

## Introduction

This document is addressed to application developers. It provides complete information on how to use the STM32F0x1/STM32F0x2/STM32F0x8 microcontroller memory and peripherals.

It applies to the STM32F031x4/x6, STM32F051x4/x6/x8, STM32F071x8/xB, STM32F091xB/xC, STM32F042x4/x6, STM32F072x8/xB, STM32F038x6, STM32F048x6, STM32F058x8, STM32F078xB and STM32F098xC devices.

For the purpose of this manual, STM32F0x1/STM32F0x2/STM32F0x8 microcontrollers are referred to as "STM32F0xx".

The STM32F0xx is a family of microcontrollers with different memory sizes, packages and peripherals.

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

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

## Related documents

- Cortex®-M0 technical reference manual, available from: <http://infocenter.arm.com>
- STM32F0xx Cortex-M0 programming manual (PM0215)
- STM32F0xx datasheets available from STMicroelectronics website: [www.st.com](http://www.st.com)

# Contents

<b>1</b>	<b>Documentation conventions</b>	<b>42</b>
1.1	General information	42
1.2	List of abbreviations for registers	42
1.3	Glossary	43
1.4	Availability of peripherals	43
<b>2</b>	<b>System and memory overview</b>	<b>44</b>
2.1	System architecture	44
2.2	Memory organization	46
2.2.1	Introduction	46
2.2.2	Memory map and register boundary addresses	47
2.3	Embedded SRAM	52
2.4	Flash memory overview	53
2.5	Boot configuration	54
<b>3</b>	<b>Embedded flash memory</b>	<b>56</b>
3.1	Flash main features	56
3.2	Flash memory functional description	56
3.2.1	Flash memory organization	56
3.2.2	Flash program and erase operations	59
3.3	Memory protection	66
3.3.1	Read protection	66
3.3.2	Write protection	68
3.3.3	Option byte write protection	68
3.4	Flash interrupts	69
3.5	Flash register description	69
3.5.1	Flash access control register (FLASH_ACR)	69
3.5.2	Flash key register (FLASH_KEYR)	70
3.5.3	Flash option key register (FLASH_OPTKEYR)	70
3.5.4	Flash status register (FLASH_SR)	71
3.5.5	Flash control register (FLASH_CR)	71
3.5.6	Flash address register (FLASH_AR)	73
3.5.7	Flash Option byte register (FLASH_OBR)	73

3.5.8	Write protection register (FLASH_WRPR) . . . . .	74
3.5.9	Flash register map . . . . .	75
<b>4</b>	<b>Option bytes . . . . .</b>	<b>76</b>
4.1	Option byte description . . . . .	77
4.1.1	User and read protection option byte . . . . .	77
4.1.2	User data option byte . . . . .	78
4.1.3	Write protection option byte . . . . .	79
4.1.4	Option byte map . . . . .	80
<b>5</b>	<b>Power control (PWR) . . . . .</b>	<b>81</b>
5.1	Power supplies . . . . .	81
5.1.1	Independent A/D and D/A converter supply and reference voltage . . . . .	82
5.1.2	Independent I/O supply rail . . . . .	82
5.1.3	Battery backup domain . . . . .	82
5.1.4	Voltage regulator . . . . .	83
5.2	Power supply supervisor . . . . .	83
5.2.1	Power on reset (POR) / power down reset (PDR) . . . . .	83
5.2.2	Programmable voltage detector (PWD) . . . . .	84
5.3	Low-power modes . . . . .	85
5.3.1	Slowing down system clocks . . . . .	86
5.3.2	Peripheral clock gating . . . . .	87
5.3.3	Sleep mode . . . . .	87
5.3.4	Stop mode . . . . .	88
5.3.5	Standby mode . . . . .	90
5.3.6	Auto-wake-up from low-power mode . . . . .	91
5.4	Power control registers . . . . .	92
5.4.1	Power control register (PWR_CR) . . . . .	92
5.4.2	Power control/status register (PWR_CSR) . . . . .	93
5.4.3	PWR register map . . . . .	94
<b>6</b>	<b>Reset and clock control (RCC) . . . . .</b>	<b>95</b>
6.1	Reset . . . . .	95
6.1.1	Power reset . . . . .	95
6.1.2	System reset . . . . .	95
6.1.3	RTC domain reset . . . . .	96
6.2	Clocks . . . . .	97

---

6.2.1	HSE clock . . . . .	101
6.2.2	HSI clock . . . . .	102
6.2.3	HSI48 clock . . . . .	103
6.2.4	PLL . . . . .	103
6.2.5	LSE clock . . . . .	104
6.2.6	LSI clock . . . . .	104
6.2.7	System clock (SYSCLK) selection . . . . .	105
6.2.8	Clock security system (CSS) . . . . .	105
6.2.9	ADC clock . . . . .	105
6.2.10	RTC clock . . . . .	106
6.2.11	Independent watchdog clock . . . . .	106
6.2.12	Clock-out capability . . . . .	106
6.2.13	Internal/external clock measurement with TIM14 . . . . .	107
6.3	Low-power modes . . . . .	108
6.4	RCC registers . . . . .	110
6.4.1	Clock control register (RCC_CR) . . . . .	110
6.4.2	Clock configuration register (RCC_CFGR) . . . . .	111
6.4.3	Clock interrupt register (RCC_CIR) . . . . .	114
6.4.4	APB peripheral reset register 2 (RCC_APB2RSTR) . . . . .	117
6.4.5	APB peripheral reset register 1 (RCC_APB1RSTR) . . . . .	119
6.4.6	AHB peripheral clock enable register (RCC_AHBENR) . . . . .	121
6.4.7	APB peripheral clock enable register 2 (RCC_APB2ENR) . . . . .	123
6.4.8	APB peripheral clock enable register 1 (RCC_APB1ENR) . . . . .	124
6.4.9	RTC domain control register (RCC_BDCR) . . . . .	127
6.4.10	Control/status register (RCC_CSR) . . . . .	128
6.4.11	AHB peripheral reset register (RCC_AHBRSTR) . . . . .	130
6.4.12	Clock configuration register 2 (RCC_CFGR2) . . . . .	131
6.4.13	Clock configuration register 3 (RCC_CFGR3) . . . . .	132
6.4.14	Clock control register 2 (RCC_CR2) . . . . .	133
6.4.15	RCC register map . . . . .	136
<b>7</b>	<b>Clock recovery system (CRS) . . . . .</b>	<b>138</b>
7.1	Introduction . . . . .	138
7.2	CRS main features . . . . .	138
7.3	CRS implementation . . . . .	138
7.4	CRS functional description . . . . .	139

7.4.1	CRS block diagram .....	139
7.4.2	Synchronization input .....	139
7.4.3	Frequency error measurement .....	140
7.4.4	Frequency error evaluation and automatic trimming .....	140
7.4.5	CRS initialization and configuration .....	141
7.5	CRS low-power modes .....	142
7.6	CRS interrupts .....	142
7.7	CRS registers .....	143
7.7.1	CRS control register (CRS_CR) .....	143
7.7.2	CRS configuration register (CRS_CFGR) .....	144
7.7.3	CRS interrupt and status register (CRS_ISR) .....	145
7.7.4	CRS interrupt flag clear register (CRS_ICR) .....	147
7.7.5	CRS register map .....	147
<b>8</b>	<b>General-purpose I/Os (GPIO) .....</b>	<b>149</b>
8.1	Introduction .....	149
8.2	GPIO main features .....	149
8.3	GPIO functional description .....	149
8.3.1	General-purpose I/O (GPIO) .....	151
8.3.2	I/O pin alternate function multiplexer and mapping .....	151
8.3.3	I/O port control registers .....	152
8.3.4	I/O port data registers .....	152
8.3.5	I/O data bitwise handling .....	152
8.3.6	GPIO locking mechanism .....	153
8.3.7	I/O alternate function input/output .....	153
8.3.8	External interrupt/wake-up lines .....	153
8.3.9	Input configuration .....	154
8.3.10	Output configuration .....	154
8.3.11	Alternate function configuration .....	155
8.3.12	Analog configuration .....	156
8.3.13	Using the HSE or LSE oscillator pins as GPIOs .....	157
8.3.14	Using the GPIO pins in the RTC supply domain .....	157
8.4	GPIO registers .....	158
8.4.1	GPIO port mode register (GPIOx_MODER) (x =A to F) .....	158
8.4.2	GPIO port output type register (GPIOx_OTYPER) (x = A to F) .....	158

8.4.3	GPIO port output speed register (GPIO <sub>x</sub> _OSPEEDR) (x = A to F) . . . . .	159
8.4.4	GPIO port pull-up/pull-down register (GPIO <sub>x</sub> _PUPDR) (x = A to F) . . . . .	159
8.4.5	GPIO port input data register (GPIO <sub>x</sub> _IDR) (x = A to F) . . . . .	160
8.4.6	GPIO port output data register (GPIO <sub>x</sub> _ODR) (x = A to F) . . . . .	160
8.4.7	GPIO port bit set/reset register (GPIO <sub>x</sub> _BSRR) (x = A to F) . . . . .	161
8.4.8	GPIO port configuration lock register (GPIO <sub>x</sub> _LCKR) (x = A to B) . . . . .	161
8.4.9	GPIO alternate function low register (GPIO <sub>x</sub> _AFRL) (x = A to F) . . . . .	162
8.4.10	GPIO alternate function high register (GPIO <sub>x</sub> _AFRH) (x = A to F) . . . . .	163
8.4.11	GPIO port bit reset register (GPIO <sub>x</sub> _BRR) (x = A to F) . . . . .	163
8.4.12	GPIO register map . . . . .	164
<b>9</b>	<b>System configuration controller (SYSCFG) . . . . .</b>	<b>166</b>
9.1	SYSCFG registers . . . . .	166
9.1.1	SYSCFG configuration register 1 (SYSCFG_CFGR1) . . . . .	166
9.1.2	SYSCFG external interrupt configuration register 1 (SYSCFG_EXTICR1) . . . . .	169
9.1.3	SYSCFG external interrupt configuration register 2 (SYSCFG_EXTICR2) . . . . .	170
9.1.4	SYSCFG external interrupt configuration register 3 (SYSCFG_EXTICR3) . . . . .	170
9.1.5	SYSCFG external interrupt configuration register 4 (SYSCFG_EXTICR4) . . . . .	171
9.1.6	SYSCFG configuration register 2 (SYSCFG_CFGR2) . . . . .	172
9.1.7	SYSCFG interrupt line 0 status register (SYSCFG_ITLINE0) . . . . .	172
9.1.8	SYSCFG interrupt line 1 status register (SYSCFG_ITLINE1) . . . . .	173
9.1.9	SYSCFG interrupt line 2 status register (SYSCFG_ITLINE2) . . . . .	173
9.1.10	SYSCFG interrupt line 3 status register (SYSCFG_ITLINE3) . . . . .	174
9.1.11	SYSCFG interrupt line 4 status register (SYSCFG_ITLINE4) . . . . .	174
9.1.12	SYSCFG interrupt line 5 status register (SYSCFG_ITLINE5) . . . . .	174
9.1.13	SYSCFG interrupt line 6 status register (SYSCFG_ITLINE6) . . . . .	175
9.1.14	SYSCFG interrupt line 7 status register (SYSCFG_ITLINE7) . . . . .	175
9.1.15	SYSCFG interrupt line 8 status register (SYSCFG_ITLINE8) . . . . .	176
9.1.16	SYSCFG interrupt line 9 status register (SYSCFG_ITLINE9) . . . . .	176

9.1.17	SYSCFG interrupt line 10 status register (SYSCFG_ITLINE10) . . . . .	176
9.1.18	SYSCFG interrupt line 11 status register (SYSCFG_ITLINE11) . . . . .	177
9.1.19	SYSCFG interrupt line 12 status register (SYSCFG_ITLINE12) . . . . .	177
9.1.20	SYSCFG interrupt line 13 status register (SYSCFG_ITLINE13) . . . . .	178
9.1.21	SYSCFG interrupt line 14 status register (SYSCFG_ITLINE14) . . . . .	178
9.1.22	SYSCFG interrupt line 15 status register (SYSCFG_ITLINE15) . . . . .	178
9.1.23	SYSCFG interrupt line 16 status register (SYSCFG_ITLINE16) . . . . .	179
9.1.24	SYSCFG interrupt line 17 status register (SYSCFG_ITLINE17) . . . . .	179
9.1.25	SYSCFG interrupt line 18 status register (SYSCFG_ITLINE18) . . . . .	179
9.1.26	SYSCFG interrupt line 19 status register (SYSCFG_ITLINE19) . . . . .	180
9.1.27	SYSCFG interrupt line 20 status register (SYSCFG_ITLINE20) . . . . .	180
9.1.28	SYSCFG interrupt line 21 status register (SYSCFG_ITLINE21) . . . . .	180
9.1.29	SYSCFG interrupt line 22 status register (SYSCFG_ITLINE22) . . . . .	181
9.1.30	SYSCFG interrupt line 23 status register (SYSCFG_ITLINE23) . . . . .	181
9.1.31	SYSCFG interrupt line 24 status register (SYSCFG_ITLINE24) . . . . .	181
9.1.32	SYSCFG interrupt line 25 status register (SYSCFG_ITLINE25) . . . . .	182
9.1.33	SYSCFG interrupt line 26 status register (SYSCFG_ITLINE26) . . . . .	182
9.1.34	SYSCFG interrupt line 27 status register (SYSCFG_ITLINE27) . . . . .	182
9.1.35	SYSCFG interrupt line 28 status register (SYSCFG_ITLINE28) . . . . .	183
9.1.36	SYSCFG interrupt line 29 status register (SYSCFG_ITLINE29) . . . . .	183
9.1.37	SYSCFG interrupt line 30 status register (SYSCFG_ITLINE30) . . . . .	184
9.1.38	SYSCFG register maps . . . . .	185
<b>10</b>	<b>Direct memory access controller (DMA) . . . . .</b>	<b>188</b>
10.1	Introduction . . . . .	188
10.2	DMA main features . . . . .	188
10.3	DMA implementation . . . . .	189
10.3.1	DMA1 and DMA2 . . . . .	189
10.3.2	DMA request mapping . . . . .	189
10.4	DMA functional description . . . . .	194
10.4.1	DMA block diagram . . . . .	194
10.4.2	DMA transfers . . . . .	195
10.4.3	DMA arbitration . . . . .	196
10.4.4	DMA channels . . . . .	197
10.4.5	DMA data width, alignment and endianness . . . . .	200
10.4.6	DMA error management . . . . .	202
10.5	DMA interrupts . . . . .	202

---

10.6	DMA registers . . . . .	202
10.6.1	DMA interrupt status register (DMA_ISR) . . . . .	203
10.6.2	DMA interrupt flag clear register (DMA_IFCR) . . . . .	205
10.6.3	DMA channel x configuration register (DMA_CCRx) . . . . .	206
10.6.4	DMA channel x number of data to transfer register (DMA_CNDTRx) . . . . .	209
10.6.5	DMA channel x peripheral address register (DMA_CPARx) . . . . .	210
10.6.6	DMA channel x memory address register (DMA_CMARx) . . . . .	210
10.6.7	DMA channel selection register (DMA_CSELR) . . . . .	212
10.6.8	DMA register map . . . . .	212
<b>11</b>	<b>Interrupts and events . . . . .</b>	<b>215</b>
11.1	Nested vectored interrupt controller (NVIC) . . . . .	215
11.1.1	NVIC main features . . . . .	215
11.1.2	SysTick calibration value register . . . . .	215
11.1.3	Interrupt and exception vectors . . . . .	215
11.2	Extended interrupts and events controller (EXTI) . . . . .	217
11.2.1	Main features . . . . .	217
11.2.2	Block diagram . . . . .	218
11.2.3	Event management . . . . .	218
11.2.4	Functional description . . . . .	218
11.2.5	External and internal interrupt/event line mapping . . . . .	220
11.3	EXTI registers . . . . .	221
11.3.1	Interrupt mask register (EXTI_IMR) . . . . .	221
11.3.2	Event mask register (EXTI_EMR) . . . . .	222
11.3.3	Rising trigger selection register (EXTI_RTSR) . . . . .	222
11.3.4	Falling trigger selection register (EXTI_FTSR) . . . . .	223
11.3.5	Software interrupt event register (EXTI_SWIER) . . . . .	223
11.3.6	Pending register (EXTI_PR) . . . . .	224
11.3.7	EXTI register map . . . . .	225
<b>12</b>	<b>Cyclic redundancy check calculation unit (CRC) . . . . .</b>	<b>226</b>
12.1	Introduction . . . . .	226
12.2	CRC main features . . . . .	226
12.3	CRC implementation . . . . .	226
12.4	CRC functional description . . . . .	227
12.4.1	CRC block diagram . . . . .	227

12.4.2	CRC internal signals . . . . .	227
12.4.3	CRC operation . . . . .	227
12.5	CRC registers . . . . .	229
12.5.1	CRC data register (CRC_DR) . . . . .	229
12.5.2	CRC independent data register (CRC_IDR) . . . . .	229
12.5.3	CRC control register (CRC_CR) . . . . .	230
12.5.4	CRC initial value (CRC_INIT) . . . . .	231
12.5.5	CRC polynomial (CRC_POL) . . . . .	231
12.5.6	CRC register map . . . . .	232
<b>13</b>	<b>Analog-to-digital converter (ADC) . . . . .</b>	<b>233</b>
13.1	Introduction . . . . .	233
13.2	ADC main features . . . . .	234
13.3	ADC functional description . . . . .	235
13.3.1	ADC pins and internal signals . . . . .	235
13.3.2	Calibration (ADCAL) . . . . .	236
13.3.3	ADC on-off control (ADEN, ADDIS, ADRDY) . . . . .	237
13.3.4	ADC clock (CKMODE) . . . . .	239
13.3.5	Configuring the ADC . . . . .	240
13.3.6	Channel selection (CHSEL, SCANDIR) . . . . .	240
13.3.7	Programmable sampling time (SMP) . . . . .	241
13.3.8	Single conversion mode (CONT = 0) . . . . .	241
13.3.9	Continuous conversion mode (CONT = 1) . . . . .	242
13.3.10	Starting conversions (ADSTART) . . . . .	242
13.3.11	Timings . . . . .	243
13.3.12	Stopping an ongoing conversion (ADSTP) . . . . .	244
13.4	Conversion on external trigger and trigger polarity (EXTSEL, EXTEN) .	244
13.4.1	Discontinuous mode (DISCEN) . . . . .	245
13.4.2	Programmable resolution (RES) - Fast conversion mode . . . . .	245
13.4.3	End of conversion, end of sampling phase (EOC, EOSMP flags) . . . . .	246
13.4.4	End of conversion sequence (EOS flag) . . . . .	246
13.4.5	Example timing diagrams (single/continuous modes hardware/software triggers) . . . . .	247
13.5	Data management . . . . .	249
13.5.1	Data register and data alignment (ADC_DR, ALIGN) . . . . .	249
13.5.2	ADC overrun (OVR, OVRMOD) . . . . .	249
13.5.3	Managing a sequence of data converted without using the DMA . . . . .	250

---

13.5.4	Managing converted data without using the DMA without overrun . . . . .	250
13.5.5	Managing converted data using the DMA . . . . .	250
13.6	Low-power features . . . . .	252
13.6.1	Wait mode conversion . . . . .	252
13.6.2	Auto-off mode (AUTOFF) . . . . .	253
13.7	Analog window watchdog . . . . .	254
13.7.1	Description of the analog watchdog . . . . .	254
13.7.2	ADC_AWD1_OUT output signal generation . . . . .	255
13.7.3	Analog watchdog threshold control . . . . .	257
13.8	Temperature sensor and internal reference voltage . . . . .	258
13.9	Battery voltage monitoring . . . . .	260
13.10	ADC interrupts . . . . .	261
13.11	ADC registers . . . . .	262
13.11.1	ADC interrupt and status register (ADC_ISR) . . . . .	262
13.11.2	ADC interrupt enable register (ADC_IER) . . . . .	263
13.11.3	ADC control register (ADC_CR) . . . . .	265
13.11.4	ADC configuration register 1 (ADC_CFGR1) . . . . .	267
13.11.5	ADC configuration register 2 (ADC_CFGR2) . . . . .	271
13.11.6	ADC sampling time register (ADC_SMPR) . . . . .	271
13.11.7	ADC watchdog threshold register (ADC_TR) . . . . .	272
13.11.8	ADC channel selection register (ADC_CHSELR) . . . . .	272
13.11.9	ADC data register (ADC_DR) . . . . .	273
13.11.10	ADC common configuration register (ADC_CCR) . . . . .	274
13.12	ADC register map . . . . .	274
<b>14</b>	<b>Digital-to-analog converter (DAC) . . . . .</b>	<b>276</b>
14.1	Introduction . . . . .	276
14.2	DAC main features . . . . .	276
14.3	DAC output buffer enable . . . . .	277
14.4	DAC channel enable . . . . .	278
14.5	Single mode functional description . . . . .	278
14.5.1	DAC data format . . . . .	278
14.5.2	DAC channel conversion . . . . .	278
14.5.3	DAC output voltage . . . . .	279
14.5.4	DAC trigger selection . . . . .	280

14.6	Dual-mode functional description (STM32F07x and STM32F09x devices) . . . . .	280
14.6.1	DAC data format . . . . .	280
14.6.2	DAC channel conversion in dual mode . . . . .	281
14.6.3	Description of dual conversion modes . . . . .	281
14.6.4	DAC output voltage . . . . .	285
14.6.5	DAC trigger selection . . . . .	286
14.7	Noise generation(STM32F07x and STM32F09x devices) . . . . .	286
14.8	Triangle-wave generation (STM32F07x and STM32F09x devices) . . . . .	287
14.9	DMA request . . . . .	288
14.10	DAC registers . . . . .	289
14.10.1	DAC control register (DAC_CR) . . . . .	289
14.10.2	DAC software trigger register (DAC_SWTRIGR) . . . . .	293
14.10.3	DAC channel1 12-bit right-aligned data holding register (DAC_DHR12R1) . . . . .	293
14.10.4	DAC channel1 12-bit left-aligned data holding register (DAC_DHR12L1) . . . . .	294
14.10.5	DAC channel1 8-bit right-aligned data holding register (DAC_DHR8R1) . . . . .	294
14.10.6	DAC channel2 12-bit right-aligned data holding register (DAC_DHR12R2) . . . . .	294
14.10.7	DAC channel2 12-bit left-aligned data holding register (DAC_DHR12L2) . . . . .	295
14.10.8	DAC channel2 8-bit right-aligned data holding register (DAC_DHR8R2) . . . . .	295
14.10.9	Dual DAC 12-bit right-aligned data holding register (DAC_DHR12RD) . . . . .	296
14.10.10	Dual DAC 12-bit left-aligned data holding register (DAC_DHR12LD) . . . . .	296
14.10.11	Dual DAC 8-bit right-aligned data holding register (DAC_DHR8RD) . . . . .	296
14.10.12	DAC channel1 data output register (DAC_DOR1) . . . . .	297
14.10.13	DAC channel2 data output register (DAC_DOR2) . . . . .	297
14.10.14	DAC status register (DAC_SR) . . . . .	297
14.10.15	DAC register map . . . . .	299
15	<b>Comparator (COMP) . . . . .</b>	<b>301</b>
15.1	Introduction . . . . .	301

---

15.2	COMP main features . . . . .	301
15.3	COMP functional description . . . . .	302
15.3.1	COMP block diagram . . . . .	302
15.3.2	COMP pins and internal signals . . . . .	302
15.3.3	COMP reset and clocks . . . . .	303
15.3.4	Comparator LOCK mechanism . . . . .	303
15.3.5	Hysteresis . . . . .	303
15.3.6	Power mode . . . . .	304
15.4	COMP interrupts . . . . .	304
15.5	COMP registers . . . . .	304
15.5.1	COMP control and status register (COMP_CSR) . . . . .	304
15.5.2	COMP register map . . . . .	308
<b>16</b>	<b>Touch sensing controller (TSC) . . . . .</b>	<b>309</b>
16.1	Introduction . . . . .	309
16.2	TSC main features . . . . .	309
16.3	TSC functional description . . . . .	310
16.3.1	TSC block diagram . . . . .	310
16.3.2	Surface charge transfer acquisition overview . . . . .	310
16.3.3	Reset and clocks . . . . .	312
16.3.4	Charge transfer acquisition sequence . . . . .	313
16.3.5	Spread spectrum feature . . . . .	314
16.3.6	Max count error . . . . .	314
16.3.7	Sampling capacitor I/O and channel I/O mode selection . . . . .	315
16.3.8	Acquisition mode . . . . .	316
16.3.9	I/O hysteresis and analog switch control . . . . .	316
16.4	TSC low-power modes . . . . .	317
16.5	TSC interrupts . . . . .	317
16.6	TSC registers . . . . .	318
16.6.1	TSC control register (TSC_CR) . . . . .	318
16.6.2	TSC interrupt enable register (TSC_IER) . . . . .	320
16.6.3	TSC interrupt clear register (TSC_ICR) . . . . .	321
16.6.4	TSC interrupt status register (TSC_ISR) . . . . .	322
16.6.5	TSC I/O hysteresis control register (TSC_IOHCR) . . . . .	322
16.6.6	TSC I/O analog switch control register (TSC_IOASCR) . . . . .	323

16.6.7	TSC I/O sampling control register (TSC_IOSCR) . . . . .	323
16.6.8	TSC I/O channel control register (TSC_IOCCR) . . . . .	324
16.6.9	TSC I/O group control status register (TSC_IOGCSR) . . . . .	324
16.6.10	TSC I/O group x counter register (TSC_IOGxCR) . . . . .	325
16.6.11	TSC register map . . . . .	326
<b>17</b>	<b>Advanced-control timers (TIM1) . . . . .</b>	<b>328</b>
17.1	TIM1 introduction . . . . .	328
17.2	TIM1 main features . . . . .	328
17.3	TIM1 functional description . . . . .	330
17.3.1	Time-base unit . . . . .	330
17.3.2	Counter modes . . . . .	332
17.3.3	Repetition counter . . . . .	342
17.3.4	Clock sources . . . . .	344
17.3.5	Capture/compare channels . . . . .	347
17.3.6	Input capture mode . . . . .	350
17.3.7	PWM input mode . . . . .	351
17.3.8	Forced output mode . . . . .	352
17.3.9	Output compare mode . . . . .	352
17.3.10	PWM mode . . . . .	353
17.3.11	Complementary outputs and dead-time insertion . . . . .	357
17.3.12	Using the break function . . . . .	359
17.3.13	Clearing the OCxREF signal on an external event . . . . .	362
17.3.14	6-step PWM generation . . . . .	364
17.3.15	One-pulse mode . . . . .	365
17.3.16	Encoder interface mode . . . . .	366
17.3.17	Timer input XOR function . . . . .	369
17.3.18	Interfacing with Hall sensors . . . . .	369
17.3.19	TIMx and external trigger synchronization . . . . .	371
17.3.20	Timer synchronization . . . . .	374
17.3.21	Debug mode . . . . .	374
17.4	TIM1 registers . . . . .	375
17.4.1	TIM1 control register 1 (TIM1_CR1) . . . . .	375
17.4.2	TIM1 control register 2 (TIM1_CR2) . . . . .	376
17.4.3	TIM1 slave mode control register (TIM1_SMCR) . . . . .	378
17.4.4	TIM1 DMA/interrupt enable register (TIM1_DIER) . . . . .	381
17.4.5	TIM1 status register (TIM1_SR) . . . . .	383

---

17.4.6	TIM1 event generation register (TIM1_EGR) . . . . .	384
17.4.7	TIM1 capture/compare mode register 1 (TIM1_CCMR1) . . . . .	386
17.4.8	TIM1 capture/compare mode register 2 (TIM1_CCMR2) . . . . .	389
17.4.9	TIM1 capture/compare enable register (TIM1_CCER) . . . . .	391
17.4.10	TIM1 counter (TIM1_CNT) . . . . .	394
17.4.11	TIM1 prescaler (TIM1_PSC) . . . . .	395
17.4.12	TIM1 auto-reload register (TIM1_ARR) . . . . .	395
17.4.13	TIM1 repetition counter register (TIM1_RCR) . . . . .	395
17.4.14	TIM1 capture/compare register 1 (TIM1_CCR1) . . . . .	396
17.4.15	TIM1 capture/compare register 2 (TIM1_CCR2) . . . . .	396
17.4.16	TIM1 capture/compare register 3 (TIM1_CCR3) . . . . .	397
17.4.17	TIM1 capture/compare register 4 (TIM1_CCR4) . . . . .	398
17.4.18	TIM1 break and dead-time register (TIM1_BDTR) . . . . .	398
17.4.19	TIM1 DMA control register (TIM1_DCR) . . . . .	400
17.4.20	TIM1 DMA address for full transfer (TIM1_DMAR) . . . . .	401
17.4.21	TIM1 register map . . . . .	402
<b>18</b>	<b>General-purpose timers (TIM2 and TIM3) . . . . .</b>	<b>404</b>
18.1	TIM2 and TIM3 introduction . . . . .	404
18.2	TIM2 and TIM3 main features . . . . .	404
18.3	TIM2 and TIM3 functional description . . . . .	405
18.3.1	Time-base unit . . . . .	405
18.3.2	Counter modes . . . . .	407
18.3.3	Clock sources . . . . .	418
18.3.4	Capture/compare channels . . . . .	421
18.3.5	Input capture mode . . . . .	423
18.3.6	PWM input mode . . . . .	425
18.3.7	Forced output mode . . . . .	426
18.3.8	Output compare mode . . . . .	426
18.3.9	PWM mode . . . . .	427
18.3.10	One-pulse mode . . . . .	431
18.3.11	Clearing the OCxREF signal on an external event . . . . .	432
18.3.12	Encoder interface mode . . . . .	433
18.3.13	Timer input XOR function . . . . .	435
18.3.14	Timers and external trigger synchronization . . . . .	436
18.3.15	Timer synchronization . . . . .	439
18.3.16	Debug mode . . . . .	445

18.4	TIM2 and TIM3 registers .....	446
18.4.1	TIM2 and TIM3 control register 1 (TIM2_CR1 and TIM3_CR1) .....	446
18.4.2	TIM2 and TIM3 control register 2 (TIM2_CR2 and TIM3_CR2) .....	448
18.4.3	TIM2 and TIM3 slave mode control register (TIM2_SMCR and TIM3_SMCR) .....	449
18.4.4	TIM2 and TIM3 DMA/Interrupt enable register (TIM2_DIER and TIM3_DIER) .....	451
18.4.5	TIM2 and TIM3 status register (TIM2_SR and TIM3_SR) .....	452
18.4.6	TIM2 and TIM3 event generation register (TIM2_EGR and TIM3_EGR) .....	455
18.4.7	TIM2 and TIM3 capture/compare mode register 1 (TIM2_CCMR1 and TIM3_CCMR1) .....	456
18.4.8	TIM2 and TIM3 capture/compare mode register 2 (TIM2_CCMR2 and TIM3_CCMR2) .....	459
18.4.9	TIM2 and TIM3 capture/compare enable register (TIM2_CCER and TIM3_CCER) .....	460
18.4.10	TIM2 and TIM3 counter (TIM2_CNT and TIM3_CNT) .....	462
18.4.11	TIM2 and TIM3 prescaler (TIM2_PSC and TIM3_PSC) .....	462
18.4.12	TIM2 and TIM3 auto-reload register (TIM2_ARR and TIM3_ARR) ..	463
18.4.13	TIM2 and TIM3 capture/compare register 1 (TIM2_CCR1 and TIM3_CCR1) .....	463
18.4.14	TIM2 and TIM3 capture/compare register 2 (TIM2_CCR2 and TIM3_CCR2) .....	464
18.4.15	TIM2 and TIM3 capture/compare register 3 (TIM2_CCR3 and TIM3_CCR3) .....	464
18.4.16	TIM2 and TIM3 capture/compare register 4 (TIM2_CCR4 and TIM3_CCR4) .....	466
18.4.17	TIM2 and TIM3 DMA control register (TIM2_DCR and TIM3_DCR) ..	466
18.4.18	TIM2 and TIM3 DMA address for full transfer (TIM2_DMAR and TIM3_DMAR) .....	467
18.4.19	TIM2 and TIM3 register map .....	469
<b>19</b>	<b>General-purpose timer (TIM14) .....</b>	<b>471</b>
19.1	TIM14 introduction .....	471
19.2	TIM14 main features .....	471
19.3	TIM14 functional description .....	472
19.3.1	Time-base unit .....	472
19.3.2	Counter operation .....	473
19.3.3	Clock source .....	476
19.3.4	Capture/compare channels .....	476

---

19.3.5	Input capture mode .....	478
19.3.6	Forced output mode .....	479
19.3.7	Output compare mode .....	479
19.3.8	PWM mode .....	481
19.3.9	Debug mode .....	482
19.4	TIM14 registers .....	482
19.4.1	TIM14 control register 1 (TIM14_CR1) .....	482
19.4.2	TIM14 interrupt enable register (TIM14_DIER) .....	483
19.4.3	TIM14 status register (TIM14_SR) .....	484
19.4.4	TIM14 event generation register (TIM14_EGR) .....	484
19.4.5	TIM14 capture/compare mode register 1 [alternate] (TIM14_CCMR1) .....	485
19.4.6	TIM14 capture/compare mode register 1 [alternate] (TIM14_CCMR1) .....	486
19.4.7	TIM14 capture/compare enable register (TIM14_CCER) .....	487
19.4.8	TIM14 counter (TIM14_CNT) .....	488
19.4.9	TIM14 prescaler (TIM14_PSC) .....	489
19.4.10	TIM14 auto-reload register (TIM14_ARR) .....	489
19.4.11	TIM14 capture/compare register 1 (TIM14_CCR1) .....	489
19.4.12	TIM14 option register (TIM14_OR) .....	490
19.4.13	TIM14 register map .....	490
20	<b>General-purpose timers (TIM15/16/17) .....</b>	<b>492</b>
20.1	TIM15/16/17 introduction .....	492
20.2	TIM15 main features .....	492
20.3	TIM16 and TIM17 main features .....	494
20.4	TIM15/16/17 functional description .....	495
20.4.1	Time-base unit .....	495
20.4.2	Counter operation .....	497
20.4.3	Repetition counter .....	501
20.4.4	Clock sources .....	502
20.4.5	Capture/compare channels .....	504
20.4.6	Input capture mode .....	507
20.4.7	PWM input mode (only for TIM15) .....	508
20.4.8	Forced output mode .....	509
20.4.9	Output compare mode .....	509
20.4.10	PWM mode .....	510
20.4.11	Complementary outputs and dead-time insertion .....	511
20.4.12	Using the break function .....	514

20.4.13	One-pulse mode . . . . .	517
20.4.14	TIM15 external trigger synchronization . . . . .	518
20.4.15	Timer synchronization (TIM15) . . . . .	521
20.4.16	Debug mode . . . . .	521
20.5	TIM15 registers . . . . .	522
20.5.1	TIM15 control register 1 (TIM15_CR1) . . . . .	522
20.5.2	TIM15 control register 2 (TIM15_CR2) . . . . .	523
20.5.3	TIM15 slave mode control register (TIM15_SMCR) . . . . .	524
20.5.4	TIM15 DMA/interrupt enable register (TIM15_DIER) . . . . .	526
20.5.5	TIM15 status register (TIM15_SR) . . . . .	527
20.5.6	TIM15 event generation register (TIM15_EGR) . . . . .	529
20.5.7	TIM15 capture/compare mode register 1 [alternate] (TIM15_CCMR1)	530
20.5.8	TIM15 capture/compare mode register 1 [alternate] (TIM15_CCMR1)	531
20.5.9	TIM15 capture/compare enable register (TIM15_CCER) . . . . .	533
20.5.10	TIM15 counter (TIM15_CNT) . . . . .	536
20.5.11	TIM15 prescaler (TIM15_PSC) . . . . .	536
20.5.12	TIM15 auto-reload register (TIM15_ARR) . . . . .	536
20.5.13	TIM15 repetition counter register (TIM15_RCR) . . . . .	537
20.5.14	TIM15 capture/compare register 1 (TIM15_CCR1) . . . . .	537
20.5.15	TIM15 capture/compare register 2 (TIM15_CCR2) . . . . .	537
20.5.16	TIM15 break and dead-time register (TIM15_BDTR) . . . . .	538
20.5.17	TIM15 DMA control register (TIM15_DCR) . . . . .	540
20.5.18	TIM15 DMA address for full transfer (TIM15_DMAR) . . . . .	541
20.5.19	TIM15 register map . . . . .	541
20.6	TIM16/TIM17 registers . . . . .	542
20.6.1	TIMx control register 1 (TIMx_CR1)(x = 16 to 17) . . . . .	542
20.6.2	TIMx control register 2 (TIMx_CR2)(x = 16 to 17) . . . . .	544
20.6.3	TIMx DMA/interrupt enable register (TIMx_DIER)(x = 16 to 17) . . . . .	544
20.6.4	TIMx status register (TIMx_SR)(x = 16 to 17) . . . . .	545
20.6.5	TIMx event generation register (TIMx_EGR)(x = 16 to 17) . . . . .	546
20.6.6	TIMx capture/compare mode register 1 [alternate] (TIMx_CCMR1)(x = 16 to 17) . . . . .	547
20.6.7	TIMx capture/compare mode register 1 [alternate] (TIMx_CCMR1)(x = 16 to 17) . . . . .	548
20.6.8	TIMx capture/compare enable register (TIMx_CCER)(x = 16 to 17) . . . . .	550
20.6.9	TIMx counter (TIMx_CNT)(x = 16 to 17) . . . . .	553
20.6.10	TIMx prescaler (TIMx_PSC)(x = 16 to 17) . . . . .	553

---

20.6.11	TIMx auto-reload register (TIMx_ARR)(x = 16 to 17) . . . . .	553
20.6.12	TIMx repetition counter register (TIMx_RCR)(x = 16 to 17) . . . . .	554
20.6.13	TIMx capture/compare register 1 (TIMx_CCR1)(x = 16 to 17) . . . . .	554
20.6.14	TIMx break and dead-time register (TIMx_BDTR)(x = 16 to 17) . . . . .	554
20.6.15	TIMx DMA control register (TIMx_DCR)(x = 16 to 17) . . . . .	556
20.6.16	TIMx DMA address for full transfer (TIMx_DMAR)(x = 16 to 17) . . . . .	557
20.6.17	TIM16/TIM17 register map . . . . .	558
<b>21</b>	<b>Basic timer (TIM6/TIM7) . . . . .</b>	<b>560</b>
21.1	TIM6/TIM7 introduction . . . . .	560
21.2	TIM6/TIM7 main features . . . . .	560
21.3	TIM6/TIM7 functional description . . . . .	561
21.3.1	Time-base unit . . . . .	561
21.3.2	Counter modes . . . . .	563
21.3.3	Clock source . . . . .	567
21.3.4	Debug mode . . . . .	567
21.4	TIM6/TIM7 registers . . . . .	568
21.4.1	TIM6/TIM7 control register 1 (TIMx_CR1) . . . . .	568
21.4.2	TIM6/TIM7 control register 2 (TIMx_CR2) . . . . .	569
21.4.3	TIM6/TIM7 DMA/Interrupt enable register (TIMx_DIER) . . . . .	569
21.4.4	TIM6/TIM7 status register (TIMx_SR) . . . . .	570
21.4.5	TIM6/TIM7 event generation register (TIMx_EGR) . . . . .	570
21.4.6	TIM6/TIM7 counter (TIMx_CNT) . . . . .	570
21.4.7	TIM6/TIM7 prescaler (TIMx_PSC) . . . . .	571
21.4.8	TIM6/TIM7 auto-reload register (TIMx_ARR) . . . . .	571
21.4.9	TIM6/TIM7 register map . . . . .	572
<b>22</b>	<b>Infrared interface (IRTIM) . . . . .</b>	<b>573</b>
<b>23</b>	<b>Independent watchdog (IWDG) . . . . .</b>	<b>574</b>
23.1	Introduction . . . . .	574
23.2	IWDG main features . . . . .	574
23.3	IWDG functional description . . . . .	574
23.3.1	IWDG block diagram . . . . .	574
23.3.2	Window option . . . . .	575
23.3.3	Hardware watchdog . . . . .	576
23.3.4	Register access protection . . . . .	576

23.3.5	Debug mode .....	576
23.4	IWDG registers .....	577
23.4.1	IWDG key register (IWDG_KR) .....	577
23.4.2	IWDG prescaler register (IWDG_PR) .....	578
23.4.3	IWDG reload register (IWDG_RLR) .....	579
23.4.4	IWDG status register (IWDG_SR) .....	580
23.4.5	IWDG window register (IWDG_WINR) .....	581
23.4.6	IWDG register map .....	582
<b>24</b>	<b>System window watchdog (WWDG) .....</b>	<b>583</b>
24.1	Introduction .....	583
24.2	WWDG main features .....	583
24.3	WWDG functional description .....	583
24.3.1	WWDG block diagram .....	584
24.3.2	Enabling the watchdog .....	584
24.3.3	Controlling the down-counter .....	584
24.3.4	How to program the watchdog timeout .....	584
24.3.5	Debug mode .....	585
24.4	WWDG interrupts .....	586
24.5	WWDG registers .....	586
24.5.1	WWDG control register (WWDG_CR) .....	586
24.5.2	WWDG configuration register (WWDG_CFR) .....	587
24.5.3	WWDG status register (WWDG_SR) .....	587
24.5.4	WWDG register map .....	588
<b>25</b>	<b>Real-time clock (RTC) .....</b>	<b>589</b>
25.1	Introduction .....	589
25.2	RTC main features .....	590
25.3	RTC implementation .....	590
25.4	RTC functional description .....	591
25.4.1	RTC block diagram .....	591
25.4.2	GPIOs controlled by the RTC .....	593
25.4.3	Clock and prescalers .....	595
25.4.4	Real-time clock and calendar .....	595
25.4.5	Programmable alarm .....	596
25.4.6	Periodic auto-wake-up .....	596

---

25.4.7	RTC initialization and configuration . . . . .	597
25.4.8	Reading the calendar . . . . .	599
25.4.9	Resetting the RTC . . . . .	600
25.4.10	RTC synchronization . . . . .	600
25.4.11	RTC reference clock detection . . . . .	601
25.4.12	RTC smooth digital calibration . . . . .	601
25.4.13	Time-stamp function . . . . .	603
25.4.14	Tamper detection . . . . .	604
25.4.15	Calibration clock output . . . . .	606
25.4.16	Alarm output . . . . .	606
25.5	RTC low-power modes . . . . .	606
25.6	RTC interrupts . . . . .	607
25.7	RTC registers . . . . .	607
25.7.1	RTC time register (RTC_TR) . . . . .	607
25.7.2	RTC date register (RTC_DR) . . . . .	608
25.7.3	RTC control register (RTC_CR) . . . . .	610
25.7.4	RTC initialization and status register (RTC_ISR) . . . . .	613
25.7.5	RTC prescaler register (RTC_PRER) . . . . .	615
25.7.6	RTC wake-up timer register (RTC_WUTR) . . . . .	616
25.7.7	RTC alarm A register (RTC_ALRMAR) . . . . .	617
25.7.8	RTC write protection register (RTC_WPR) . . . . .	618
25.7.9	RTC sub second register (RTC_SSR) . . . . .	618
25.7.10	RTC shift control register (RTC_SHIFTR) . . . . .	619
25.7.11	RTC timestamp time register (RTC_TSTR) . . . . .	620
25.7.12	RTC timestamp date register (RTC_TSDR) . . . . .	621
25.7.13	RTC time-stamp sub second register (RTC_TSSSR) . . . . .	622
25.7.14	RTC calibration register (RTC_CALR) . . . . .	623
25.7.15	RTC tamper and alternate function configuration register (RTC_TAFCR) . . . . .	624
25.7.16	RTC alarm A sub second register (RTC_ALRMASSR) . . . . .	627
25.7.17	RTC backup registers (RTC_BKPxR) . . . . .	628
25.7.18	RTC register map . . . . .	628
26	<b>Inter-integrated circuit (I2C) interface . . . . .</b>	<b>631</b>
26.1	Introduction . . . . .	631
26.2	I2C main features . . . . .	631
26.3	I2C implementation . . . . .	632

26.4	I2C functional description . . . . .	632
26.4.1	I2C1 block diagram . . . . .	633
26.4.2	I2C2 block diagram . . . . .	633
26.4.3	I2C pins and internal signals . . . . .	634
26.4.4	I2C clock requirements . . . . .	635
26.4.5	Mode selection . . . . .	635
26.4.6	I2C initialization . . . . .	636
26.4.7	Software reset . . . . .	641
26.4.8	Data transfer . . . . .	642
26.4.9	I2C slave mode . . . . .	644
26.4.10	I2C master mode . . . . .	653
26.4.11	I2C_TIMINGR register configuration examples . . . . .	665
26.4.12	SMBus specific features . . . . .	666
26.4.13	SMBus initialization . . . . .	669
26.4.14	SMBus: I2C_TIMEOUTR register configuration examples . . . . .	671
26.4.15	SMBus slave mode . . . . .	671
26.4.16	Wake-up from Stop mode on address match . . . . .	678
26.4.17	Error conditions . . . . .	679
26.4.18	DMA requests . . . . .	681
26.4.19	Debug mode . . . . .	682
26.5	I2C low-power modes . . . . .	682
26.6	I2C interrupts . . . . .	683
26.7	I2C registers . . . . .	684
26.7.1	I2C control register 1 (I2C_CR1) . . . . .	684
26.7.2	I2C control register 2 (I2C_CR2) . . . . .	687
26.7.3	I2C own address 1 register (I2C_OAR1) . . . . .	689
26.7.4	I2C own address 2 register (I2C_OAR2) . . . . .	690
26.7.5	I2C timing register (I2C_TIMINGR) . . . . .	691
26.7.6	I2C timeout register (I2C_TIMEOUTR) . . . . .	692
26.7.7	I2C interrupt and status register (I2C_ISR) . . . . .	693
26.7.8	I2C interrupt clear register (I2C_ICR) . . . . .	695
26.7.9	I2C PEC register (I2C_PECR) . . . . .	696
26.7.10	I2C receive data register (I2C_RXDR) . . . . .	697
26.7.11	I2C transmit data register (I2C_TXDR) . . . . .	697
26.7.12	I2C register map . . . . .	698

<b>27</b>	<b>Universal synchronous/asynchronous receiver transmitter (USART/UART) . . . . .</b>	<b>700</b>
27.1	Introduction . . . . .	700
27.2	USART main features . . . . .	700
27.3	USART extended features . . . . .	701
27.4	USART implementation . . . . .	702
27.5	USART functional description . . . . .	703
27.5.1	USART character description . . . . .	705
27.5.2	USART transmitter . . . . .	707
27.5.3	USART receiver . . . . .	710
27.5.4	USART baud rate generation . . . . .	716
27.5.5	Tolerance of the USART receiver to clock deviation . . . . .	718
27.5.6	USART auto baud rate detection . . . . .	719
27.5.7	Multiprocessor communication using USART . . . . .	720
27.5.8	Modbus communication using USART . . . . .	722
27.5.9	USART parity control . . . . .	723
27.5.10	USART LIN (local interconnection network) mode . . . . .	724
27.5.11	USART synchronous mode . . . . .	726
27.5.12	USART Single-wire Half-duplex communication . . . . .	729
27.5.13	USART Smartcard mode . . . . .	729
27.5.14	USART IrDA SIR ENDEC block . . . . .	734
27.5.15	USART continuous communication in DMA mode . . . . .	736
27.5.16	RS232 hardware flow control and RS485 driver enable using USART . . . . .	738
27.5.17	Wake-up from Stop mode using USART . . . . .	740
27.6	USART in low-power modes . . . . .	742
27.7	USART interrupts . . . . .	742
27.8	USART registers . . . . .	744
27.8.1	USART control register 1 (USART_CR1) . . . . .	744
27.8.2	USART control register 2 (USART_CR2) . . . . .	747
27.8.3	USART control register 3 (USART_CR3) . . . . .	751
27.8.4	USART baud rate register (USART_BRR) . . . . .	755
27.8.5	USART guard time and prescaler register (USART_GTPR) . . . . .	755
27.8.6	USART receiver timeout register (USART_RTOR) . . . . .	756
27.8.7	USART request register (USART_RQR) . . . . .	757
27.8.8	USART interrupt and status register (USART_ISR) . . . . .	758

27.8.9	USART interrupt flag clear register (USART_ICR) .....	763
27.8.10	USART receive data register (USART_RDR) .....	764
27.8.11	USART transmit data register (USART_TDR) .....	764
27.8.12	USART register map .....	765
<b>28</b>	<b>Serial peripheral interface / integrated interchip sound (SPI/I2S) . . . . .</b>	<b>767</b>
28.1	Introduction .....	767
28.2	SPI main features .....	767
28.3	I2S main features .....	768
28.4	SPI/I2S implementation .....	768
28.5	SPI functional description .....	769
28.5.1	General description .....	769
28.5.2	Communications between one master and one slave .....	770
28.5.3	Standard multi-slave communication .....	772
28.5.4	Multi-master communication .....	773
28.5.5	Slave select (NSS) pin management .....	774
28.5.6	Communication formats .....	775
28.5.7	Configuration of SPI .....	777
28.5.8	Procedure for enabling SPI .....	778
28.5.9	Data transmission and reception procedures .....	778
28.5.10	SPI status flags .....	788
28.5.11	SPI error flags .....	789
28.5.12	NSS pulse mode .....	790
28.5.13	TI mode .....	790
28.5.14	CRC calculation .....	791
28.6	SPI interrupts .....	793
28.7	I2S functional description .....	794
28.7.1	I2S general description .....	794
28.7.2	I2S full duplex .....	795
28.7.3	Supported audio protocols .....	796
28.7.4	Start-up description .....	803
28.7.5	Clock generator .....	805
28.7.6	I <sup>2</sup> S master mode .....	808
28.7.7	I <sup>2</sup> S slave mode .....	809
28.7.8	I2S status flags .....	811
28.7.9	I2S error flags .....	812

---

28.7.10	DMA features . . . . .	813
28.8	I2S interrupts . . . . .	813
28.9	SPI and I2S registers . . . . .	814
28.9.1	SPI control register 1 (SPIx_CR1) . . . . .	814
28.9.2	SPI control register 2 (SPIx_CR2) . . . . .	816
28.9.3	SPI status register (SPIx_SR) . . . . .	818
28.9.4	SPI data register (SPIx_DR) . . . . .	820
28.9.5	SPI CRC polynomial register (SPIx_CRCPR) . . . . .	820
28.9.6	SPI Rx CRC register (SPIx_RXCRCR) . . . . .	820
28.9.7	SPI Tx CRC register (SPIx_TXCRCR) . . . . .	821
28.9.8	SPIx_I2S configuration register (SPIx_I2SCFGR) . . . . .	821
28.9.9	SPIx_I2S prescaler register (SPIx_I2SPR) . . . . .	823
28.9.10	SPI/I2S register map . . . . .	824
<b>29</b>	<b>Controller area network (bxCAN) . . . . .</b>	<b>825</b>
29.1	Introduction . . . . .	825
29.2	bxCAN main features . . . . .	825
29.3	bxCAN general description . . . . .	825
29.3.1	CAN 2.0B active core . . . . .	826
29.3.2	Control, status and configuration registers . . . . .	826
29.3.3	Tx mailboxes . . . . .	826
29.3.4	Acceptance filters . . . . .	826
29.4	bxCAN operating modes . . . . .	827
29.4.1	Initialization mode . . . . .	827
29.4.2	Normal mode . . . . .	828
29.4.3	Sleep mode (low-power) . . . . .	828
29.5	Test mode . . . . .	829
29.5.1	Silent mode . . . . .	829
29.5.2	Loop back mode . . . . .	830
29.5.3	Loop back combined with silent mode . . . . .	830
29.6	Behavior in debug mode . . . . .	831
29.7	bxCAN functional description . . . . .	831
29.7.1	Transmission handling . . . . .	831
29.7.2	Time triggered communication mode . . . . .	833
29.7.3	Reception handling . . . . .	833
29.7.4	Identifier filtering . . . . .	834

29.7.5	Message storage . . . . .	838
29.7.6	Error management . . . . .	839
29.7.7	Bit timing . . . . .	840
29.8	bxCAN interrupts . . . . .	843
29.9	CAN registers . . . . .	844
29.9.1	Register access protection . . . . .	844
29.9.2	CAN control and status registers . . . . .	844
29.9.3	CAN mailbox registers . . . . .	855
29.9.4	CAN filter registers . . . . .	860
29.9.5	bxCAN register map . . . . .	864
<b>30</b>	<b>Universal serial bus full-speed device interface (USB) . . . . .</b>	<b>868</b>
30.1	Introduction . . . . .	868
30.2	USB main features . . . . .	868
30.3	USB implementation . . . . .	868
30.4	USB functional description . . . . .	870
30.4.1	Description of USB blocks . . . . .	871
30.5	Programming considerations . . . . .	872
30.5.1	Generic USB device programming . . . . .	872
30.5.2	System and power-on reset . . . . .	873
30.5.3	Double-buffered endpoints . . . . .	878
30.5.4	Isochronous transfers . . . . .	880
30.5.5	Suspend/Resume events . . . . .	881
30.6	USB and USB SRAM registers . . . . .	884
30.6.1	Common registers . . . . .	884
30.6.2	Buffer descriptor table . . . . .	897
30.6.3	USB register map . . . . .	900
<b>31</b>	<b>HDMI-CEC controller (CEC) . . . . .</b>	<b>902</b>
31.1	Introduction . . . . .	902
31.2	HDMI-CEC controller main features . . . . .	902
31.3	HDMI-CEC functional description . . . . .	903
31.3.1	HDMI-CEC pin . . . . .	903
31.3.2	HDMI-CEC block diagram . . . . .	903
31.3.3	Message description . . . . .	903
31.3.4	Bit timing . . . . .	904

---

31.4	Arbitration . . . . .	905
31.4.1	SFT option bit . . . . .	906
31.5	Error handling . . . . .	907
31.5.1	Bit error . . . . .	907
31.5.2	Message error . . . . .	907
31.5.3	Bit rising error (BRE) . . . . .	907
31.5.4	Short bit period error (SBPE) . . . . .	908
31.5.5	Long bit period error (LBPE) . . . . .	908
31.5.6	Transmission error detection (TXERR) . . . . .	909
31.6	HDMI-CEC interrupts . . . . .	911
31.7	HDMI-CEC registers . . . . .	912
31.7.1	CEC control register (CEC_CR) . . . . .	912
31.7.2	CEC configuration register (CEC_CFGR) . . . . .	913
31.7.3	CEC Tx data register (CEC_TXDR) . . . . .	915
31.7.4	CEC Rx data register (CEC_RXDR) . . . . .	915
31.7.5	CEC interrupt and status register (CEC_ISR) . . . . .	915
31.7.6	CEC interrupt enable register (CEC_IER) . . . . .	917
31.7.7	HDMI-CEC register map . . . . .	919
<b>32</b>	<b>Debug support (DBG) . . . . .</b>	<b>920</b>
32.1	Overview . . . . .	920
32.2	Reference Arm documentation . . . . .	921
32.3	Pinout and debug port pins . . . . .	921
32.3.1	SWD port pins . . . . .	922
32.3.2	SW-DP pin assignment . . . . .	922
32.3.3	Internal pull-up and pull-down on SWD pins . . . . .	922
32.4	ID codes and locking mechanism . . . . .	922
32.4.1	MCU device ID code . . . . .	923
32.5	SWD port . . . . .	924
32.5.1	SWD protocol introduction . . . . .	924
32.5.2	SWD protocol sequence . . . . .	924
32.5.3	SW-DP state machine (reset, idle states, ID code) . . . . .	925
32.5.4	DP and AP read/write accesses . . . . .	925
32.5.5	SW-DP registers . . . . .	927
32.5.6	SW-AP registers . . . . .	928
32.6	Core debug . . . . .	928

32.7	BPU (Break Point Unit) . . . . .	929
32.7.1	BPU functionality . . . . .	929
32.8	DWT (Data Watchpoint) . . . . .	929
32.8.1	DWT functionality . . . . .	929
32.8.2	DWT Program Counter Sample Register . . . . .	929
32.9	MCU debug component (DBGMCU) . . . . .	929
32.9.1	Debug support for low-power modes . . . . .	930
32.9.2	Debug support for timers, watchdog and I <sup>2</sup> C . . . . .	930
32.9.3	Debug MCU configuration register (DBGMCU_CR) . . . . .	931
32.9.4	Debug MCU APB1 freeze register (DBGMCU_APB1_FZ) . . . . .	932
32.9.5	Debug MCU APB2 freeze register (DBGMCU_APB2_FZ) . . . . .	934
32.9.6	DBG register map . . . . .	934
<b>33</b>	<b>Device electronic signature . . . . .</b>	<b>936</b>
33.1	Unique device ID register (96 bits) . . . . .	936
33.2	Flash memory size data register . . . . .	937
<b>A</b>	<b>Appendix A Code examples . . . . .</b>	<b>938</b>
A.1	Introduction . . . . .	938
A.2	Flash operation code examples . . . . .	938
A.2.1	Flash memory unlocking sequence . . . . .	938
A.2.2	Main Flash programming sequence . . . . .	938
A.2.3	Page erase sequence . . . . .	939
A.2.4	Mass erase sequence . . . . .	940
A.2.5	Option byte unlocking sequence . . . . .	940
A.2.6	Option byte programming sequence . . . . .	941
A.2.7	Option byte erasing sequence . . . . .	941
A.3	Clock controller . . . . .	942
A.3.1	HSE start sequence code example . . . . .	942
A.3.2	PLL configuration modification code example . . . . .	943
A.3.3	MCO selection code example . . . . .	943
A.3.4	Clock measurement configuration with TIM14 code example . . . . .	944
A.4	GPIO . . . . .	945
A.4.1	Lock sequence code example . . . . .	945
A.4.2	Alternate function selection sequence code example . . . . .	945
A.4.3	Analog GPIO configuration code example . . . . .	946

---

A.5	DMA . . . . .	946
	A.5.1    DMA channel configuration sequence code example . . . . .	946
A.6	Interrupts and event . . . . .	947
	A.6.1    NVIC initialization example . . . . .	947
	A.6.2    External interrupt selection code example . . . . .	947
A.7	ADC . . . . .	948
	A.7.1    ADC calibration code example . . . . .	948
	A.7.2    ADC enable sequence code example . . . . .	948
	A.7.3    ADC disable sequence code example . . . . .	949
	A.7.4    ADC clock selection code example . . . . .	949
	A.7.5    Single conversion sequence code example - Software trigger . . . . .	950
	A.7.6    Continuous conversion sequence code example - Software trigger . . . . .	950
	A.7.7    Single conversion sequence code example - Hardware trigger . . . . .	951
	A.7.8    Continuous conversion sequence code example - Hardware trigger . . . . .	951
	A.7.9    DMA one shot mode sequence code example . . . . .	952
	A.7.10    DMA circular mode sequence code example . . . . .	952
	A.7.11    Wait mode sequence code example . . . . .	952
	A.7.12    Auto Off and no wait mode sequence code example . . . . .	953
	A.7.13    Auto Off and wait mode sequence code example . . . . .	953
	A.7.14    Analog watchdog code example . . . . .	953
	A.7.15    Temperature configuration code example . . . . .	954
	A.7.16    Temperature computation code example . . . . .	954
A.8	DAC . . . . .	954
	A.8.1    Independent trigger without wave generation code example . . . . .	954
	A.8.2    Independent trigger with single LFSR generation code example . . . . .	955
	A.8.3    Independent trigger with different LFSR generation code example . . . . .	955
	A.8.4    Independent trigger with single triangle generation code example . . . . .	956
	A.8.5    Independent trigger with different triangle generation code example . . . . .	956
	A.8.6    Simultaneous software start code example . . . . .	956
	A.8.7    Simultaneous trigger without wave generation code example . . . . .	957
	A.8.8    Simultaneous trigger with single LFSR generation code example . . . . .	957
	A.8.9    Simultaneous trigger with different LFSR generation code example . . . . .	957
	A.8.10    Simultaneous trigger with single triangle generation code example . . . . .	958
	A.8.11    Simultaneous trigger with different triangle generation code example . . . . .	958
	A.8.12    DMA initialization code example . . . . .	959
A.9	Timers . . . . .	960

A.9.1	Upcounter on TI2 rising edge code example . . . . .	960
A.9.2	Up counter on each 2 ETR rising edges code example . . . . .	961
A.9.3	Input capture configuration code example . . . . .	961
A.9.4	Input capture data management code example . . . . .	962
A.9.5	PWM input configuration code example . . . . .	963
A.9.6	PWM input with DMA configuration code example . . . . .	963
A.9.7	Output compare configuration code example . . . . .	964
A.9.8	Edge-aligned PWM configuration example . . . . .	964
A.9.9	Center-aligned PWM configuration example . . . . .	965
A.9.10	ETR configuration to clear OCxREF code example . . . . .	966
A.9.11	Encoder interface code example . . . . .	966
A.9.12	Reset mode code example . . . . .	967
A.9.13	Gated mode code example . . . . .	967
A.9.14	Trigger mode code example . . . . .	968
A.9.15	External clock mode 2 + trigger mode code example . . . . .	968
A.9.16	One-Pulse mode code example . . . . .	969
A.9.17	Timer prescaling another timer code example . . . . .	969
A.9.18	Timer enabling another timer code example . . . . .	970
A.9.19	Master and slave synchronization code example . . . . .	971
A.9.20	Two timers synchronized by an external trigger code example . . . . .	972
A.9.21	DMA burst feature code example . . . . .	973
A.10	IRTIM code example . . . . .	974
A.10.1	TIM16 and TIM17 configuration code example . . . . .	974
A.10.2	IRQHandler for IRTIM code example . . . . .	975
A.11	bxCAN code example . . . . .	976
A.11.1	bxCAN initialization mode code example . . . . .	976
A.11.2	bxCAN transmit code example . . . . .	976
A.11.3	bxCAN receive code example . . . . .	977
A.12	DBG code example . . . . .	977
A.12.1	DBG read device ID code example . . . . .	977
A.12.2	DBG debug in Low-power mode code example . . . . .	977
A.13	HDMI-CEC code example . . . . .	977
A.13.1	HDMI-CEC configure CEC code example . . . . .	977
A.13.2	HDMI-CEC transmission with interrupt enabled code example . . . . .	978
A.13.3	HDMI-CEC interrupt management code example . . . . .	978
A.14	I2C code example . . . . .	978

---

A.14.1	I2C configured in master mode to receive code example . . . . .	978
A.14.2	I2C configured in master mode to transmit code example . . . . .	979
A.14.3	I2C configured in slave mode code example . . . . .	979
A.14.4	I2C master transmitter code example . . . . .	979
A.14.5	I2C master receiver code example . . . . .	979
A.14.6	I2C slave transmitter code example . . . . .	980
A.14.7	I2C slave receiver code example . . . . .	980
A.14.8	I2C configured in master mode to transmit with DMA code example . .	980
A.14.9	I2C configured in slave mode to receive with DMA code example . .	981
A.15	IWDG code example . . . . .	981
A.15.1	IWDG configuration code example . . . . .	981
A.15.2	IWDG configuration with window code example . . . . .	982
A.16	RTC code example . . . . .	982
A.16.1	RTC calendar configuration code example . . . . .	982
A.16.2	RTC alarm configuration code example . . . . .	983
A.16.3	RTC WUT configuration code example . . . . .	983
A.16.4	RTC read calendar code example . . . . .	983
A.16.5	RTC calibration code example . . . . .	984
A.16.6	RTC tamper and time stamp configuration code example . . . . .	984
A.16.7	RTC tamper and time stamp code example . . . . .	985
A.16.8	RTC clock output code example . . . . .	985
A.17	SPI code example . . . . .	985
A.17.1	SPI master configuration code example . . . . .	985
A.17.2	SPI slave configuration code example . . . . .	986
A.17.3	SPI full duplex communication code example . . . . .	986
A.17.4	SPI interrupt code example . . . . .	986
A.17.5	SPI master configuration with DMA code example . . . . .	986
A.17.6	SPI slave configuration with DMA code example . . . . .	987
A.18	TSC code example . . . . .	987
A.18.1	TSC configuration code example . . . . .	987
A.18.2	TSC interrupt code example . . . . .	987
A.19	USART code example . . . . .	988
A.19.1	USART transmitter configuration code example . . . . .	988
A.19.2	USART transmit byte code example . . . . .	988
A.19.3	USART transfer complete code example . . . . .	988
A.19.4	USART receiver configuration code example . . . . .	988

A.19.5	USART receive byte code example . . . . .	988
A.19.6	USART LIN mode code example . . . . .	989
A.19.7	USART synchronous mode code example . . . . .	989
A.19.8	USART single-wire half-duplex code example . . . . .	990
A.19.9	USART smartcard mode code example . . . . .	990
A.19.10	USART IrDA mode code example . . . . .	991
A.19.11	USART DMA code example . . . . .	991
A.19.12	USART hardware flow control code example . . . . .	992
A.20	WWDG code example. . . . .	992
A.20.1	WWDG configuration code example. . . . .	992
	<b>Important security notice . . . . .</b>	<b>993</b>
	<b>Revision history . . . . .</b>	<b>994</b>

## List of tables

Table 1.	STM32F0xx peripheral register boundary addresses .....	47
Table 2.	STM32F0xx memory boundary addresses .....	51
Table 3.	Boot modes .....	54
Table 4.	Flash memory organization (STM32F03x, STM32F04x and STM32F05x devices).....	57
Table 5.	Flash memory organization (STM32F07x, STM32F09x devices).....	58
Table 6.	Flash memory read protection status .....	66
Table 7.	Access status versus protection level and execution modes .....	67
Table 8.	Flash interrupt request .....	69
Table 9.	Flash interface - Register map and reset values .....	75
Table 10.	Option byte format .....	76
Table 11.	Option byte organization.....	76
Table 12.	Option byte map and ST production values .....	80
Table 13.	Low-power mode summary .....	86
Table 14.	Sleep-now.....	88
Table 15.	Sleep-on-exit.....	88
Table 16.	Stop mode .....	89
Table 17.	Standby mode.....	90
Table 18.	PWR register map and reset values.....	94
Table 19.	RCC register map and reset values .....	136
Table 20.	CRS features .....	138
Table 21.	Effect of low-power modes on CRS .....	142
Table 22.	Interrupt control bits .....	142
Table 23.	CRS register map and reset values .....	147
Table 24.	Port bit configuration table .....	150
Table 25.	GPIO register map and reset values .....	164
Table 26.	SYSCFG register map and reset values.....	185
Table 27.	SYSCFG register map and reset values for STM32F09x devices .....	185
Table 28.	DMA1 and DMA2 implementation .....	189
Table 29.	DMA requests for each channel on STM32F03x/04x/05x devices .....	189
Table 30.	DMA requests for each channel on STM32F07x devices.....	190
Table 31.	DMA1 requests for each channel on STM32F09x devices.....	193
Table 32.	DMA2 requests for each channel on STM32F09x devices.....	194
Table 33.	Programmable data width and endian behavior (when PINC = MINC = 1) .....	201
Table 34.	DMA interrupt requests.....	202
Table 35.	DMA register map and reset values .....	212
Table 36.	Vector table.....	215
Table 37.	External interrupt/event controller register map and reset values.....	225
Table 38.	CRC features .....	226
Table 39.	CRC internal input/output signals .....	227
Table 40.	CRC register map and reset values .....	232
Table 41.	ADC input/output pins.....	235
Table 42.	ADC internal input/output signals .....	236
Table 43.	External triggers .....	236
Table 44.	Latency between trigger and start of conversion .....	240
Table 45.	Configuring the trigger polarity .....	244
Table 46.	tSAR timings depending on resolution .....	246
Table 47.	Analog watchdog comparison.....	255

Table 48.	Analog watchdog channel selection . . . . .	255
Table 49.	ADC interrupts . . . . .	261
Table 50.	ADC register map and reset values . . . . .	274
Table 51.	DAC pins . . . . .	277
Table 52.	External triggers . . . . .	280
Table 53.	DAC register map and reset values . . . . .	299
Table 54.	COMP register map and reset values . . . . .	308
Table 55.	Acquisition sequence summary . . . . .	312
Table 56.	Spread spectrum deviation versus AHB clock frequency . . . . .	314
Table 57.	I/O state depending on its mode and IODEF bit value . . . . .	315
Table 58.	Effect of low-power modes on TSC . . . . .	317
Table 59.	Interrupt control bits . . . . .	317
Table 60.	TSC register map and reset values . . . . .	326
Table 61.	Counting direction versus encoder signals . . . . .	367
Table 62.	TIMx Internal trigger connection . . . . .	380
Table 63.	Output control bits for complementary OC <sub>x</sub> and OC <sub>xN</sub> channels with break feature . . . . .	393
Table 64.	TIM1 register map and reset values . . . . .	402
Table 65.	Counting direction versus encoder signals . . . . .	434
Table 66.	TIM2 and TIM3 internal trigger connection . . . . .	451
Table 67.	Output control bit for standard OC <sub>x</sub> channels . . . . .	462
Table 68.	TIM2 and TIM3 register map and reset values . . . . .	469
Table 69.	Output control bit for standard OC <sub>x</sub> channels . . . . .	488
Table 70.	TIM14 register map and reset values . . . . .	490
Table 71.	TIMx Internal trigger connection . . . . .	525
Table 72.	Output control bits for complementary OC <sub>x</sub> and OC <sub>xN</sub> channels with break feature . . . . .	535
Table 73.	TIM15 register map and reset values . . . . .	541
Table 74.	Output control bits for complementary OC <sub>x</sub> and OC <sub>xN</sub> channels with break feature . . . . .	552
Table 75.	TIM16/TIM17 register map and reset values . . . . .	558
Table 76.	TIM6/TIM7 register map and reset values . . . . .	572
Table 77.	IWDG register map and reset values . . . . .	582
Table 78.	WWDG register map and reset values . . . . .	588
Table 79.	STM32F0xx RTC implementation . . . . .	590
Table 80.	RTC pin PC13 configuration . . . . .	594
Table 81.	LSE pin PC14 configuration . . . . .	594
Table 82.	LSE pin PC15 configuration . . . . .	594
Table 83.	Effect of low-power modes on RTC . . . . .	606
Table 84.	Interrupt control bits . . . . .	607
Table 85.	RTC register map and reset values . . . . .	628
Table 86.	STM32F0xx I2C implementation . . . . .	632
Table 87.	I2C input/output pins . . . . .	634
Table 88.	I2C internal input/output signals . . . . .	635
Table 89.	Comparison of analog vs. digital filters . . . . .	637
Table 90.	I2C-SMBus specification data setup and hold times . . . . .	640
Table 91.	I2C configuration . . . . .	644
Table 92.	I2C-SMBus specification clock timings . . . . .	655
Table 93.	Examples of timing settings for fI2CCLK = 8 MHz . . . . .	665
Table 94.	Examples of timing settings for fI2CCLK = 16 MHz . . . . .	665
Table 95.	Examples of timing settings for fI2CCLK = 48 MHz . . . . .	666
Table 96.	SMBus timeout specifications . . . . .	668
Table 97.	SMBus with PEC configuration . . . . .	669
Table 98.	Examples of TIMEOUTA settings for various I2CCLK frequencies . . . . .	

---

Table 99.	(max $t_{TIMEOUT} = 25$ ms) . . . . .	671
Table 100.	Examples of TIMEOUTB settings for various I2CCLK frequencies . . . . .	671
Table 101.	Examples of TIMEOUTA settings for various I2CCLK frequencies (max $t_{IDLE} = 50$ $\mu$ s) . . . . .	671
Table 102.	Effect of low-power modes on the I2C . . . . .	682
Table 103.	I2C Interrupt requests . . . . .	683
Table 104.	I2C register map and reset values . . . . .	698
Table 105.	STM32F0xx USART features . . . . .	702
Table 106.	Noise detection from sampled data . . . . .	714
Table 107.	Error calculation for programmed baud rates at $f_{CK} = 48$ MHz in both cases of oversampling by 16 or by 8 . . . . .	717
Table 108.	Tolerance of the USART receiver when BRR [3:0] = 0000 . . . . .	719
Table 109.	Tolerance of the USART receiver when BRR [3:0] is different from 0000 . . . . .	719
Table 110.	Frame formats . . . . .	723
Table 111.	Effect of low-power modes on the USART . . . . .	742
Table 112.	USART interrupt requests . . . . .	742
Table 113.	USART register map and reset values . . . . .	765
Table 114.	STM32F0xx SPI and SPI/I2S implementation . . . . .	768
Table 115.	SPI interrupt requests . . . . .	793
Table 116.	SPI/I2S register map and reset values . . . . .	807
Table 117.	Transmit mailbox mapping . . . . .	813
Table 118.	Receive mailbox mapping . . . . .	824
Table 119.	bxCAN register map and reset values . . . . .	839
Table 120.	STM32F0xx USB implementation . . . . .	864
Table 121.	Double-buffering buffer flag definition . . . . .	868
Table 122.	Bulk double-buffering memory buffers usage . . . . .	879
Table 123.	Isochronous memory buffers usage . . . . .	879
Table 124.	Resume event detection . . . . .	881
Table 125.	Reception status encoding . . . . .	882
Table 126.	Endpoint type encoding . . . . .	895
Table 127.	Endpoint kind meaning . . . . .	895
Table 128.	Transmission status encoding . . . . .	896
Table 129.	Definition of allocated buffer memory . . . . .	899
Table 130.	USB register map and reset values . . . . .	900
Table 131.	HDMI pin . . . . .	903
Table 132.	Error handling timing parameters . . . . .	909
Table 133.	TXERR timing parameters . . . . .	910
Table 134.	HDMI-CEC interrupts . . . . .	911
Table 135.	HDMI-CEC register map and reset values . . . . .	919
Table 136.	SW debug port pins . . . . .	922
Table 137.	DEV_ID and REV_ID field values . . . . .	923
Table 138.	Packet request (8-bits) . . . . .	924
Table 139.	ACK response (3 bits) . . . . .	925
Table 140.	DATA transfer (33 bits) . . . . .	925
Table 141.	SW-DP registers . . . . .	927
Table 142.	32-bit debug port registers addressed through the shifted value A[3:2] . . . . .	928
Table 143.	Core debug registers . . . . .	928
Table 144.	DBG register map and reset values . . . . .	935
Table 145.	Document revision history . . . . .	994

## List of figures

Figure 1.	System architecture . . . . .	44
Figure 2.	Memory map . . . . .	47
Figure 3.	Programming procedure . . . . .	61
Figure 4.	Flash memory Page erase procedure . . . . .	63
Figure 5.	Flash memory mass erase procedure . . . . .	64
Figure 6.	Power supply overview . . . . .	81
Figure 7.	Power on reset/power down reset waveform . . . . .	84
Figure 8.	PVD thresholds . . . . .	85
Figure 9.	Simplified diagram of the reset circuit . . . . .	96
Figure 10.	Clock tree (STM32F03x and STM32F05x devices) . . . . .	99
Figure 11.	Clock tree (STM32F04x, STM32F07x and STM32F09x devices) . . . . .	100
Figure 12.	HSE/ LSE clock sources . . . . .	101
Figure 13.	Frequency measurement with TIM14 in capture mode . . . . .	107
Figure 14.	CRS block diagram . . . . .	139
Figure 15.	CRS counter behavior . . . . .	140
Figure 16.	Basic structure of an I/O port bit . . . . .	150
Figure 17.	Input floating / pull up / pull down configurations . . . . .	154
Figure 18.	Output configuration . . . . .	155
Figure 19.	Alternate function configuration . . . . .	156
Figure 20.	High impedance-analog configuration . . . . .	156
Figure 21.	DMAx request routing architecture on STM32F09x devices . . . . .	192
Figure 22.	DMA block diagram . . . . .	195
Figure 23.	Extended interrupts and events controller (EXTI) block diagram . . . . .	218
Figure 24.	External interrupt/event GPIO mapping . . . . .	220
Figure 25.	CRC calculation unit block diagram . . . . .	227
Figure 26.	ADC block diagram . . . . .	235
Figure 27.	ADC calibration . . . . .	237
Figure 28.	Enabling/disabling the ADC . . . . .	238
Figure 29.	ADC clock scheme . . . . .	239
Figure 30.	Analog to digital conversion time . . . . .	243
Figure 31.	ADC conversion timings . . . . .	243
Figure 32.	Stopping an ongoing conversion . . . . .	244
Figure 33.	Single conversions of a sequence, software trigger . . . . .	247
Figure 34.	Continuous conversion of a sequence, software trigger . . . . .	247
Figure 35.	Single conversions of a sequence, hardware trigger . . . . .	248
Figure 36.	Continuous conversions of a sequence, hardware trigger . . . . .	248
Figure 37.	Data alignment and resolution . . . . .	249
Figure 38.	Example of overrun (OVR) . . . . .	250
Figure 39.	Wait mode conversion (continuous mode, software trigger) . . . . .	252
Figure 40.	Behavior with WAIT = 0, AUTOFF = 1 . . . . .	253
Figure 41.	Behavior with WAIT = 1, AUTOFF = 1 . . . . .	254
Figure 42.	Analog watchdog guarded area . . . . .	255
Figure 43.	ADC_AWD1_OUT signal generation . . . . .	256
Figure 44.	ADC_AWD1_OUT signal generation (AWD flag not cleared by software) . . . . .	257
Figure 45.	ADC1_AWD_OUT signal generation (on a single channel) . . . . .	257
Figure 46.	Analog watchdog threshold update . . . . .	258
Figure 47.	Temperature sensor and VREFINT channel block diagram . . . . .	259
Figure 48.	DAC block diagram . . . . .	277

---

Figure 49.	Data registers in single DAC channel mode . . . . .	278
Figure 50.	Timing diagram for conversion with trigger disabled TEN = 0 . . . . .	279
Figure 51.	Data registers in dual DAC channel mode . . . . .	281
Figure 52.	DAC LFSR register calculation algorithm . . . . .	286
Figure 53.	DAC conversion (SW trigger enabled) with LFSR wave generation . . . . .	286
Figure 54.	DAC triangle wave generation . . . . .	287
Figure 55.	DAC conversion (SW trigger enabled) with triangle wave generation . . . . .	287
Figure 56.	Comparator 1 and 2 block diagrams . . . . .	302
Figure 57.	Comparator hysteresis . . . . .	303
Figure 58.	TSC block diagram . . . . .	310
Figure 59.	Surface charge transfer analog I/O group structure . . . . .	311
Figure 60.	Sampling capacitor voltage variation . . . . .	312
Figure 61.	Charge transfer acquisition sequence . . . . .	313
Figure 62.	Spread spectrum variation principle . . . . .	314
Figure 63.	Advanced-control timer block diagram . . . . .	329
Figure 64.	Counter timing diagram with prescaler division change from 1 to 2 . . . . .	331
Figure 65.	Counter timing diagram with prescaler division change from 1 to 4 . . . . .	331
Figure 66.	Counter timing diagram, internal clock divided by 1 . . . . .	333
Figure 67.	Counter timing diagram, internal clock divided by 2 . . . . .	333
Figure 68.	Counter timing diagram, internal clock divided by 4 . . . . .	334
Figure 69.	Counter timing diagram, internal clock divided by N . . . . .	334
Figure 70.	Counter timing diagram, update event when ARPE=0 (TIMx_ARR not preloaded) . . . . .	335
Figure 71.	Counter timing diagram, update event when ARPE=1 (TIMx_ARR preloaded) . . . . .	335
Figure 72.	Counter timing diagram, internal clock divided by 1 . . . . .	336
Figure 73.	Counter timing diagram, internal clock divided by 2 . . . . .	337
Figure 74.	Counter timing diagram, internal clock divided by 4 . . . . .	337
Figure 75.	Counter timing diagram, internal clock divided by N . . . . .	337
Figure 76.	Counter timing diagram, update event when repetition counter is not used . . . . .	338
Figure 77.	Counter timing diagram, internal clock divided by 1, TIMx_ARR = 0x6 . . . . .	339
Figure 78.	Counter timing diagram, internal clock divided by 2 . . . . .	340
Figure 79.	Counter timing diagram, internal clock divided by 4, TIMx_ARR=0x36 . . . . .	340
Figure 80.	Counter timing diagram, internal clock divided by N . . . . .	341
Figure 81.	Counter timing diagram, update event with ARPE=1 (counter underflow) . . . . .	341
Figure 82.	Counter timing diagram, Update event with ARPE=1 (counter overflow) . . . . .	342
Figure 83.	Update rate examples depending on mode and TIMx_RCR register settings . . . . .	343
Figure 84.	Control circuit in normal mode, internal clock divided by 1 . . . . .	344
Figure 85.	TI2 external clock connection example . . . . .	345
Figure 86.	Control circuit in external clock mode 1 . . . . .	346
Figure 87.	External trigger input block . . . . .	346
Figure 88.	Control circuit in external clock mode 2 . . . . .	347
Figure 89.	Capture/compare channel (example: channel 1 input stage) . . . . .	348
Figure 90.	Capture/compare channel 1 main circuit . . . . .	348
Figure 91.	Output stage of capture/compare channel (channel 1 to 3) . . . . .	349
Figure 92.	Output stage of capture/compare channel (channel 4) . . . . .	349
Figure 93.	PWM input mode timing . . . . .	351
Figure 94.	Output compare mode, toggle on OC1 . . . . .	353
Figure 95.	Edge-aligned PWM waveforms (ARR=8) . . . . .	355
Figure 96.	Center-aligned PWM waveforms (ARR=8) . . . . .	356
Figure 97.	Complementary output with dead-time insertion . . . . .	358
Figure 98.	Dead-time waveforms with delay greater than the negative pulse . . . . .	358

---

Figure 99. Dead-time waveforms with delay greater than the positive pulse.	358
Figure 100. Output behavior in response to a break	361
Figure 101. Clearing TIMx OCxREF	363
Figure 102. 6-step generation, COM example (OSSR=1)	364
Figure 103. Example of one pulse mode	365
Figure 104. Example of counter operation in encoder interface mode	368
Figure 105. Example of encoder interface mode with TI1FP1 polarity inverted	368
Figure 106. Example of hall sensor interface	370
Figure 107. Control circuit in reset mode	371
Figure 108. Control circuit in gated mode	372
Figure 109. Control circuit in trigger mode	373
Figure 110. Control circuit in external clock mode 2 + trigger mode	374
Figure 111. General-purpose timer block diagram (TIM2 and TIM3)	405
Figure 112. Counter timing diagram with prescaler division change from 1 to 2	406
Figure 113. Counter timing diagram with prescaler division change from 1 to 4	407
Figure 114. Counter timing diagram, internal clock divided by 1	408
Figure 115. Counter timing diagram, internal clock divided by 2	408
Figure 116. Counter timing diagram, internal clock divided by 4	409
Figure 117. Counter timing diagram, internal clock divided by N	409
Figure 118. Counter timing diagram, Update event when ARPE=0 (TIMx_ARR not preloaded)	410
Figure 119. Counter timing diagram, Update event when ARPE=1 (TIMx_ARR preloaded)	410
Figure 120. Counter timing diagram, internal clock divided by 1	411
Figure 121. Counter timing diagram, internal clock divided by 2	412
Figure 122. Counter timing diagram, internal clock divided by 4	412
Figure 123. Counter timing diagram, internal clock divided by N	413
Figure 124. Counter timing diagram, Update event when repetition counter is not used	413
Figure 125. Counter timing diagram, internal clock divided by 1, TIMx_ARR=0x6	415
Figure 126. Counter timing diagram, internal clock divided by 2	415
Figure 127. Counter timing diagram, internal clock divided by 4, TIMx_ARR=0x36	416
Figure 128. Counter timing diagram, internal clock divided by N	416
Figure 129. Counter timing diagram, Update event with ARPE=1 (counter underflow)	417
Figure 130. Counter timing diagram, Update event with ARPE=1 (counter overflow)	417
Figure 131. Control circuit in normal mode, internal clock divided by 1	418
Figure 132. TI2 external clock connection example	419
Figure 133. Control circuit in external clock mode 1	420
Figure 134. External trigger input block	420
Figure 135. Control circuit in external clock mode 2	421
Figure 136. Capture/compare channel (example: channel 1 input stage)	422
Figure 137. Capture/compare channel 1 main circuit	422
Figure 138. Output stage of capture/compare channel (channel 1)	423
Figure 139. PWM input mode timing	425
Figure 140. Output compare mode, toggle on OC1	427
Figure 141. Edge-aligned PWM waveforms (ARR=8)	428
Figure 142. Center-aligned PWM waveforms (ARR=8)	430
Figure 143. Example of one-pulse mode	431
Figure 144. Clearing TIMx OCxREF	433
Figure 145. Example of counter operation in encoder interface mode	435
Figure 146. Example of encoder interface mode with TI1FP1 polarity inverted	435
Figure 147. Control circuit in reset mode	436
Figure 148. Control circuit in gated mode	437

---

Figure 149. Control circuit in trigger mode . . . . .	438
Figure 150. Control circuit in external clock mode 2 + trigger mode . . . . .	439
Figure 151. Master/Slave timer example . . . . .	440
Figure 152. Gating timer 2 with OC1REF of timer 1 . . . . .	441
Figure 153. Gating timer 2 with Enable of timer 1 . . . . .	442
Figure 154. Triggering timer 2 with update of timer 1 . . . . .	443
Figure 155. Triggering timer 2 with Enable of timer 1 . . . . .	444
Figure 156. Triggering timer 1 and 2 with timer 1 TI1 input . . . . .	445
Figure 157. General-purpose timer block diagram (TIM14) . . . . .	472
Figure 158. Counter timing diagram with prescaler division change from 1 to 2 . . . . .	473
Figure 159. Counter timing diagram with prescaler division change from 1 to 4 . . . . .	473
Figure 160. Counter timing diagram, internal clock divided by 1 . . . . .	474
Figure 161. Counter timing diagram, internal clock divided by 2 . . . . .	474
Figure 162. Counter timing diagram, internal clock divided by 4 . . . . .	475
Figure 163. Counter timing diagram, internal clock divided by N . . . . .	475
Figure 164. Counter timing diagram, update event when ARPE=0 (TIMx_ARR not preloaded) . . . . .	475
Figure 165. Counter timing diagram, update event when ARPE=1 (TIMx_ARR preloaded) . . . . .	476
Figure 166. Control circuit in normal mode, internal clock divided by 1 . . . . .	476
Figure 167. Capture/compare channel (example: channel 1 input stage) . . . . .	477
Figure 168. Capture/compare channel 1 main circuit . . . . .	477
Figure 169. Output stage of capture/compare channel (channel 1) . . . . .	478
Figure 170. Output compare mode, toggle on OC1 . . . . .	481
Figure 171. Edge-aligned PWM waveforms (ARR=8) . . . . .	482
Figure 172. TIM15 block diagram . . . . .	493
Figure 173. TIM16 and TIM17 block diagram . . . . .	495
Figure 174. Counter timing diagram with prescaler division change from 1 to 2 . . . . .	496
Figure 175. Counter timing diagram with prescaler division change from 1 to 4 . . . . .	497
Figure 176. Counter timing diagram, internal clock divided by 1 . . . . .	498
Figure 177. Counter timing diagram, internal clock divided by 2 . . . . .	499
Figure 178. Counter timing diagram, internal clock divided by 4 . . . . .	499
Figure 179. Counter timing diagram, internal clock divided by N . . . . .	500
Figure 180. Counter timing diagram, update event when ARPE=0 (TIMx_ARR not preloaded) . . . . .	500
Figure 181. Counter timing diagram, update event when ARPE=1 (TIMx_ARR preloaded) . . . . .	501
Figure 182. Update rate examples depending on mode and TIMx_RCR register settings . . . . .	502
Figure 183. Control circuit in normal mode, internal clock divided by 1 . . . . .	503
Figure 184. TI2 external clock connection example . . . . .	503
Figure 185. Control circuit in external clock mode 1 . . . . .	504
Figure 186. Capture/compare channel (example: channel 1 input stage) . . . . .	505
Figure 187. Capture/compare channel 1 main circuit . . . . .	505
Figure 188. Output stage of capture/compare channel (channel 1) . . . . .	506
Figure 189. Output stage of capture/compare channel (channel 2 for TIM15) . . . . .	506
Figure 190. PWM input mode timing . . . . .	508
Figure 191. Output compare mode, toggle on OC1 . . . . .	510
Figure 192. Edge-aligned PWM waveforms (ARR=8) . . . . .	511
Figure 193. Complementary output with dead-time insertion . . . . .	512
Figure 194. Dead-time waveforms with delay greater than the negative pulse . . . . .	513
Figure 195. Dead-time waveforms with delay greater than the positive pulse . . . . .	513
Figure 196. Output behavior in response to a break . . . . .	516

---

Figure 197. Example of One-pulse mode .....	517
Figure 198. Control circuit in reset mode .....	519
Figure 199. Control circuit in gated mode .....	520
Figure 200. Control circuit in trigger mode .....	521
Figure 201. Basic timer block diagram .....	560
Figure 202. Counter timing diagram with prescaler division change from 1 to 2 .....	562
Figure 203. Counter timing diagram with prescaler division change from 1 to 4 .....	562
Figure 204. Counter timing diagram, internal clock divided by 1 .....	563
Figure 205. Counter timing diagram, internal clock divided by 2 .....	564
Figure 206. Counter timing diagram, internal clock divided by 4 .....	564
Figure 207. Counter timing diagram, internal clock divided by N .....	565
Figure 208. Counter timing diagram, update event when ARPE = 0 (TIMx_ARR not preloaded) .....	565
Figure 209. Counter timing diagram, update event when ARPE=1 (TIMx_ARR preloaded) .....	566
Figure 210. Control circuit in normal mode, internal clock divided by 1 .....	567
Figure 211. IRTIM internal hardware connections with TIM16 and TIM17 .....	573
Figure 212. Independent watchdog block diagram .....	574
Figure 213. Watchdog block diagram .....	584
Figure 214. Window watchdog timing diagram .....	585
Figure 215. RTC block diagram in STM32F03x, STM32F04x and STM32F05x devices .....	591
Figure 216. RTC block diagram for STM32F07x and STM32F09x devices .....	592
Figure 217. I2C1 block diagram .....	633
Figure 218. I2C2 block diagram .....	634
Figure 219. I2C bus protocol .....	636
Figure 220. Setup and hold timings .....	638
Figure 221. I2C initialization flow .....	641
Figure 222. Data reception .....	642
Figure 223. Data transmission .....	643
Figure 224. Slave initialization flow .....	646
Figure 225. Transfer sequence flow for I2C slave transmitter, NOSTRETCH = 0 .....	648
Figure 226. Transfer sequence flow for I2C slave transmitter, NOSTRETCH = 1 .....	649
Figure 227. Transfer bus diagrams for I2C slave transmitter (mandatory events only) .....	650
Figure 228. Transfer sequence flow for slave receiver with NOSTRETCH = 0 .....	651
Figure 229. Transfer sequence flow for slave receiver with NOSTRETCH = 1 .....	652
Figure 230. Transfer bus diagrams for I2C slave receiver (mandatory events only) .....	652
Figure 231. Master clock generation .....	654
Figure 232. Master initialization flow .....	656
Figure 233. 10-bit address read access with HEAD10R = 0 .....	656
Figure 234. 10-bit address read access with HEAD10R = 1 .....	657
Figure 235. Transfer sequence flow for I2C master transmitter for $N \leq 255$ bytes .....	658
Figure 236. Transfer sequence flow for I2C master transmitter for $N > 255$ bytes .....	659
Figure 237. Transfer bus diagrams for I2C master transmitter (mandatory events only) .....	660
Figure 238. Transfer sequence flow for I2C master receiver for $N \leq 255$ bytes .....	662
Figure 239. Transfer sequence flow for I2C master receiver for $N > 255$ bytes .....	663
Figure 240. Transfer bus diagrams for I2C master receiver (mandatory events only) .....	664
Figure 241. Timeout intervals for $t_{LOW:SEXT}$ , $t_{LOW:MEXT}$ .....	668

---

Figure 242. Transfer sequence flow for SMBus slave transmitter N bytes + PEC .....	672
Figure 243. Transfer bus diagrams for SMBus slave transmitter (SBC = 1) .....	672
Figure 244. Transfer sequence flow for SMBus slave receiver N bytes + PEC .....	674
Figure 245. Bus transfer diagrams for SMBus slave receiver (SBC = 1) .....	675
Figure 246. Bus transfer diagrams for SMBus master transmitter .....	676
Figure 247. Bus transfer diagrams for SMBus master receiver .....	678
Figure 248. USART block diagram .....	704
Figure 249. Word length programming .....	706
Figure 250. Configurable stop bits .....	708
Figure 251. TC/TXE behavior when transmitting .....	709
Figure 252. Start bit detection when oversampling by 16 or 8 .....	710
Figure 253. Data sampling when oversampling by 16 .....	714
Figure 254. Data sampling when oversampling by 8 .....	714
Figure 255. Mute mode using Idle line detection .....	721
Figure 256. Mute mode using address mark detection .....	722
Figure 257. Break detection in LIN mode (11-bit break length - LBDL bit is set) .....	725
Figure 258. Break detection in LIN mode vs. Framing error detection .....	726
Figure 259. USART example of synchronous transmission .....	727
Figure 260. USART data clock timing diagram (M bits = 00) .....	727
Figure 261. USART data clock timing diagram (M bits = 01) .....	728
Figure 262. RX data setup/hold time .....	728
Figure 263. ISO 7816-3 asynchronous protocol .....	730
Figure 264. Parity error detection using the 1.5 stop bits .....	731
Figure 265. IrDA SIR ENDEC- block diagram .....	735
Figure 266. IrDA data modulation (3/16) -Normal Mode .....	735
Figure 267. Transmission using DMA .....	737
Figure 268. Reception using DMA .....	738
Figure 269. Hardware flow control between 2 USARTs .....	738
Figure 270. RS232 RTS flow control .....	739
Figure 271. RS232 CTS flow control .....	740
Figure 272. USART interrupt mapping diagram .....	743
Figure 273. SPI block diagram .....	769
Figure 274. Full-duplex single master/ single slave application .....	770
Figure 275. Half-duplex single master/ single slave application .....	771
Figure 276. Simplex single master/single slave application (master in transmit-only/ slave in receive-only mode) .....	772
Figure 277. Master and three independent slaves .....	773
Figure 278. Multi-master application .....	774
Figure 279. Hardware/software slave select management .....	775
Figure 280. Data clock timing diagram .....	776
Figure 281. Data alignment when data length is not equal to 8-bit or 16-bit .....	777
Figure 282. Packing data in FIFO for transmission and reception .....	781
Figure 283. Master full-duplex communication .....	784
Figure 284. Slave full-duplex communication .....	785
Figure 285. Master full-duplex communication with CRC .....	786
Figure 286. Master full-duplex communication in packed mode .....	787
Figure 287. NSSP pulse generation in Motorola SPI master mode .....	790
Figure 288. TI mode transfer .....	791
Figure 289. I <sup>2</sup> S block diagram .....	794
Figure 290. Full-duplex communication .....	796
Figure 291. I <sup>2</sup> S Philips protocol waveforms (16/32-bit full accuracy) .....	797
Figure 292. I <sup>2</sup> S Philips standard waveforms (24-bit frame) .....	797

---

Figure 293. Transmitting 0x8EAA33 . . . . .	798
Figure 294. Receiving 0x8EAA33 . . . . .	798
Figure 295. I <sup>2</sup> S Philips standard (16-bit extended to 32-bit packet frame) . . . . .	798
Figure 296. Example of 16-bit data frame extended to 32-bit channel frame . . . . .	798
Figure 297. MSB Justified 16-bit or 32-bit full-accuracy length . . . . .	799
Figure 298. MSB justified 24-bit frame length . . . . .	799
Figure 299. MSB justified 16-bit extended to 32-bit packet frame . . . . .	800
Figure 300. LSB justified 16-bit or 32-bit full-accuracy . . . . .	800
Figure 301. LSB justified 24-bit frame length . . . . .	800
Figure 302. Operations required to transmit 0x3478AE. . . . .	801
Figure 303. Operations required to receive 0x3478AE . . . . .	801
Figure 304. LSB justified 16-bit extended to 32-bit packet frame . . . . .	801
Figure 305. Example of 16-bit data frame extended to 32-bit channel frame . . . . .	802
Figure 306. PCM standard waveforms (16-bit) . . . . .	802
Figure 307. PCM standard waveforms (16-bit extended to 32-bit packet frame) . . . . .	803
Figure 308. Start sequence in master mode . . . . .	804
Figure 309. Audio sampling frequency definition . . . . .	805
Figure 310. I <sup>2</sup> S clock generator architecture . . . . .	805
Figure 311. CAN network topology . . . . .	826
Figure 312. Single-CAN block diagram . . . . .	827
Figure 313. bxCAN operating modes . . . . .	829
Figure 314. bxCAN in silent mode . . . . .	830
Figure 315. bxCAN in loop back mode . . . . .	830
Figure 316. bxCAN in combined mode . . . . .	831
Figure 317. Transmit mailbox states . . . . .	832
Figure 318. Receive FIFO states . . . . .	833
Figure 319. Filter bank scale configuration - Register organization . . . . .	836
Figure 320. Example of filter numbering . . . . .	837
Figure 321. Filtering mechanism example . . . . .	838
Figure 322. CAN error state diagram . . . . .	839
Figure 323. Bit timing . . . . .	841
Figure 324. CAN frames . . . . .	842
Figure 325. Event flags and interrupt generation . . . . .	843
Figure 326. CAN mailbox registers . . . . .	855
Figure 327. USB peripheral block diagram . . . . .	870
Figure 328. Packet buffer areas with examples of buffer description table locations . . . . .	874
Figure 329. HDMI-CEC block diagram . . . . .	903
Figure 330. Message structure . . . . .	904
Figure 331. Blocks . . . . .	904
Figure 332. Bit timings . . . . .	905
Figure 333. Signal free time . . . . .	905
Figure 334. Arbitration phase . . . . .	906
Figure 335. SFT of three nominal bit periods . . . . .	906
Figure 336. Error bit timing . . . . .	907
Figure 337. Error handling . . . . .	908
Figure 338. TXERR detection . . . . .	910
Figure 339. Block diagram of STM32F0xx MCU and Cortex <sup>®</sup> -M0-level debug support . . . . .	920

# 1 Documentation conventions

## 1.1 General information

The STM32F0xx devices have an Arm®<sup>(a)</sup> Cortex®-M0 core.



## 1.2 List of abbreviations for registers

The following abbreviations<sup>(b)</sup> are used in register descriptions:

read/write (rw)	Software can read and write to this bit.
read-only (r)	Software can only read this bit.
write-only (w)	Software can only write to this bit. Reading this bit returns the reset value.
read/clear write0 (rc_w0)	Software can read as well as clear this bit by writing 0. Writing 1 has no effect on the bit value.
read/clear write1 (rc_w1)	Software can read as well as clear this bit by writing 1. Writing 0 has no effect on the bit value.
read/clear write (rc_w)	Software can read as well as clear this bit by writing to the register. The value written to this bit is not important.
read/clear by read (rc_r)	Software can read this bit. Reading this bit automatically clears it to 0. Writing this bit has no effect on the bit value.
read/set by read (rs_r)	Software can read this bit. Reading this bit automatically sets it to 1. Writing this bit has no effect on the bit value.
read/set (rs)	Software can read as well as set this bit. Writing 0 has no effect on the bit value.
read/write once (rwo)	Software can only write once to this bit and can also read it at any time. Only a reset can return the bit to its reset value.
toggle (t)	The software can toggle this bit by writing 1. Writing 0 has no effect.
read-only write trigger (rt_w1)	Software can read this bit. Writing 1 triggers an event but has no effect on the bit value.
Reserved (Res.)	Reserved bit, must be kept at reset value.

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

## 1.3 Glossary

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

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

## 1.4 Availability of peripherals

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

## 2 System and memory overview

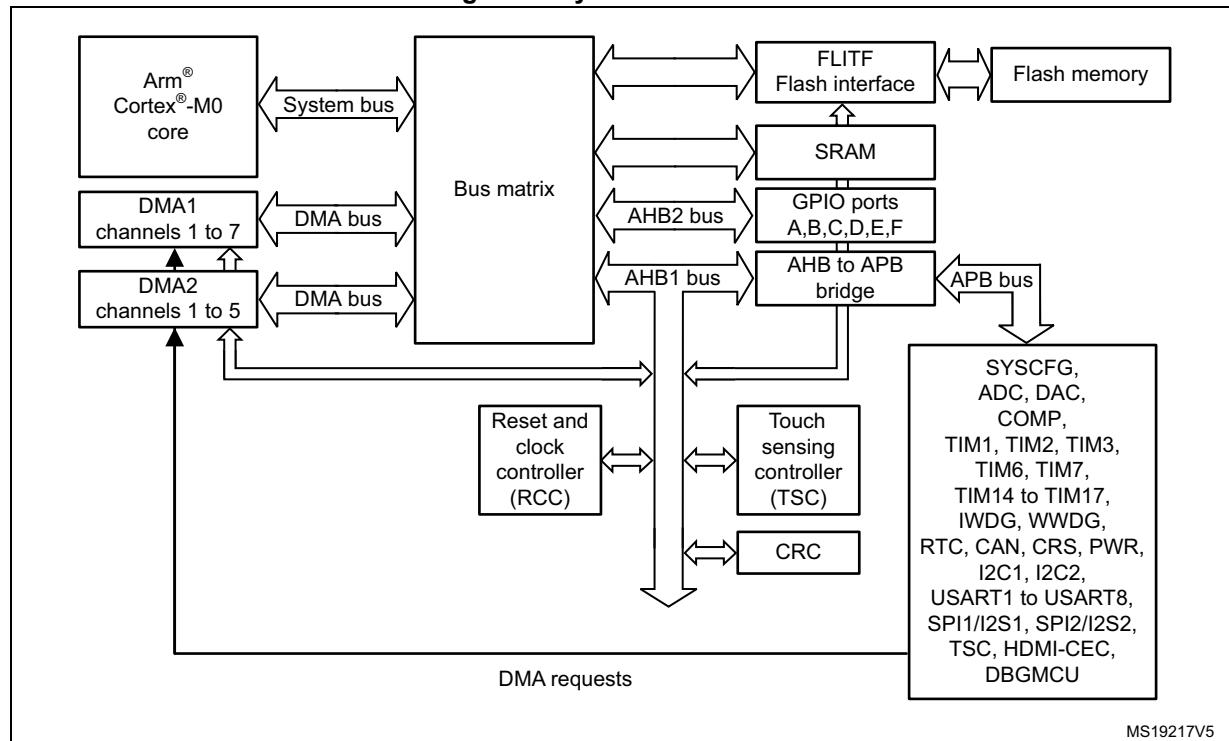
### 2.1 System architecture

The main system consists of:

- Up to three masters:
  - Cortex®-M0 core
  - General-purpose DMA1
  - General purpose DMA2 (available on STM32F09x devices only)
- Four slaves:
  - Internal SRAM
  - Internal flash memory
  - AHB1 with AHB to APB bridge, which connects all the APB peripherals
  - AHB2 dedicated to GPIO ports

These are interconnected using a multilayer AHB bus architecture, as shown in [Figure 1](#):

**Figure 1. System architecture**



MS19217V5

#### System bus

This bus connects the system bus of the Cortex®-M0 core (peripherals bus) to a BusMatrix which manages the arbitration between the core and the DMA.

## DMA bus

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

## BusMatrix

The BusMatrix manages the access arbitration between the core system bus and the DMA master bus. The arbitration uses a Round Robin algorithm. The BusMatrix is composed of up to three masters (CPU, DMA1, DMA2) and four slaves (FLITF, SRAM, AHB1 with AHB to APB bridge and AHB2).

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

## AHB to APB bridge (APB)

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

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

After each device reset, all peripheral clocks are disabled (except for the SRAM and flash). Before using a peripheral you have to enable its clock in the RCC\_AHBENR, RCC\_APB2ENR or RCC\_APB1ENR register.

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

## 2.2 Memory organization

### 2.2.1 Introduction

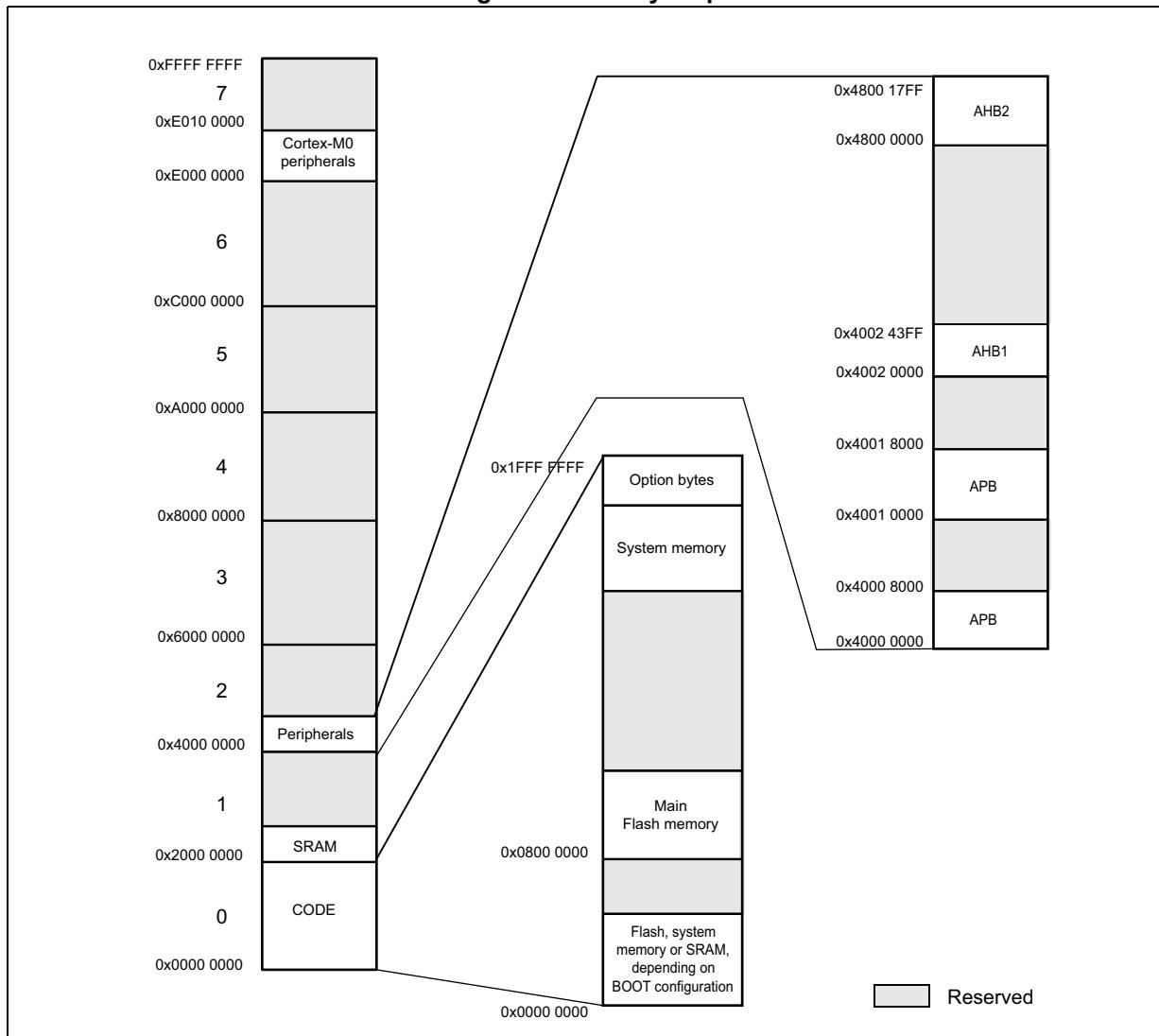
Program memory, data memory, registers and I/O ports are organized within the same linear 4-Gbyte address space.

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

The addressable memory space is divided into eight main blocks, of 512 Mbytes each.

## 2.2.2 Memory map and register boundary addresses

Figure 2. Memory map



All the memory map areas not allocated to on-chip memories and peripherals are considered “Reserved”. For the detailed mapping of available memory and register areas, refer to the following table, which gives the boundary addresses of the available peripherals.

Table 1. STM32F0xx peripheral register boundary addresses

Bus	Boundary address	Size	Peripheral	Peripheral register map
-	0xE000 0000 - 0xE00F FFFF	1MB	Cortex®-M0 internal peripherals	-
-	0x4800 1800 - 0x5FFF FFFF	~384 MB	Reserved	-

**Table 1. STM32F0xx peripheral register boundary addresses (continued)**

Bus	Boundary address	Size	Peripheral	Peripheral register map
AHB2	0x4800 1400 - 0x4800 17FF	1KB	GPIOF	<a href="#">Section 8.4.12 on page 164</a>
	0x4800 1000 - 0x4800 13FF	1KB	GPIOE	<a href="#">Section 8.4.12 on page 164</a>
	0x4800 0C00 - 0x4800 0FFF	1KB	GPIOD	<a href="#">Section 8.4.12 on page 164</a>
	0x4800 0800 - 0x4800 0BFF	1KB	GPIOC	<a href="#">Section 8.4.12 on page 164</a>
	0x4800 0400 - 0x4800 07FF	1KB	GPIOB	<a href="#">Section 8.4.12 on page 164</a>
	0x4800 0000 - 0x4800 03FF	1KB	GPIOA	<a href="#">Section 8.4.12 on page 164</a>
-	0x4002 4400 - 0x47FF FFFF	~128 MB	Reserved	-
AHB1	0x4002 4000 - 0x4002 43FF	1 KB	TSC	<a href="#">Section 16.6.11 on page 326</a>
	0x4002 3400 - 0x4002 3FFF	3 KB	Reserved	-
	0x4002 3000 - 0x4002 33FF	1 KB	CRC	<a href="#">Section 12.5.6 on page 232</a>
	0x4002 2400 - 0x4002 2FFF	3 KB	Reserved	-
	0x4002 2000 - 0x4002 23FF	1 KB	FLASH interface	<a href="#">Section 3.5.9 on page 75</a>
	0x4002 1400 - 0x4002 1FFF	3 KB	Reserved	-
	0x4002 1000 - 0x4002 13FF	1 KB	RCC	<a href="#">Section 6.4.15 on page 136</a>
	0x4002 0800 - 0x4002 0FFF	2 KB	Reserved	-
	0x4002 0400 - 0x4002 07FF	1 KB	DMA2	<a href="#">Section 10.6.8 on page 212</a>
-	0x4001 8000 - 0x4001 FFFF	32 KB	Reserved	-

**Table 1. STM32F0xx peripheral register boundary addresses (continued)**

Bus	Boundary address	Size	Peripheral	Peripheral register map
APB	0x4001 5C00 - 0x4001 7FFF	9 KB	Reserved	-
	0x4001 5800 - 0x4001 5BFF	1 KB	DBGMCU	<a href="#">Section 32.9.6 on page 934</a>
	0x4001 4C00 - 0x4001 57FF	3 KB	Reserved	-
	0x4001 4800 - 0x4001 4BFF	1 KB	TIM17	<a href="#">Section 20.6.17 on page 558</a>
	0x4001 4400 - 0x4001 47FF	1 KB	TIM16	<a href="#">Section 20.6.17 on page 558</a>
	0x4001 4000 - 0x4001 43FF	1 KB	TIM15	<a href="#">Section 20.5.19 on page 541</a>
	0x4001 3C00 - 0x4001 3FFF	1 KB	Reserved	-
	0x4001 3800 - 0x4001 3BFF	1 KB	USART1	<a href="#">Section 27.8.12 on page 765</a>
	0x4001 3400 - 0x4001 37FF	1 KB	Reserved	-
	0x4001 3000 - 0x4001 33FF	1 KB	SPI1/I2S1	<a href="#">Section 28.9.10 on page 824</a>
	0x4001 2C00 - 0x4001 2FFF	1 KB	TIM1	<a href="#">Section 17.4.21 on page 402</a>
	0x4001 2800 - 0x4001 2BFF	1 KB	Reserved	-
	0x4001 2400 - 0x4001 27FF	1 KB	ADC	<a href="#">Section 13.12 on page 274</a>
	0x4001 2000 - 0x4001 23FF	1 KB	Reserved	-
	0x4001 1C00 - 0x4001 1FFF	1 KB	USART8	<a href="#">Section 27.8.12 on page 765</a>
	0x4001 1800 - 0x4001 1BFF	1 KB	USART7	<a href="#">Section 27.8.12 on page 765</a>
	0x4001 1400 - 0x4001 17FF	1 KB	USART6	<a href="#">Section 27.8.12 on page 765</a>
	0x4001 0800 - 0x4001 13FF	3 KB	Reserved	-
	0x4001 0400 - 0x4001 07FF	1 KB	EXTI	<a href="#">Section 11.3.7 on page 225</a>
	0x4001 0000 - 0x4001 03FF	1 KB	SYSCFG COMP	<a href="#">Section 9.1.38 on page 185</a>
				<a href="#">Section 15.5.2 on page 308</a>
-	0x4000 8000 - 0x4000 FFFF	32 KB	Reserved	-

**Table 1. STM32F0xx peripheral register boundary addresses (continued)**

Bus	Boundary address	Size	Peripheral	Peripheral register map
APB	0x4000 7C00 - 0x4000 7FFF	1 KB	Reserved	-
	0x4000 7800 - 0x4000 7BFF	1 KB	CEC	<a href="#">Section 31.7.7 on page 919</a>
	0x4000 7400 - 0x4000 77FF	1 KB	DAC	<a href="#">Section 14.10.15 on page 299</a>
	0x4000 7000 - 0x4000 73FF	1 KB	PWR	<a href="#">Section 5.4.3 on page 94</a>
	0x4000 6C00 - 0x4000 6FFF	1 KB	CRS	<a href="#">Section 7.7.5 on page 147</a>
	0x4000 6800 - 0x4000 6BFF	1 KB	Reserved	-
	0x4000 6400 - 0x4000 67FF	1 KB	CAN	<a href="#">Section 29.9.5 on page 864</a>
	0x4000 6000 - 0x4000 63FF	1 KB	USB/CAN SRAM	<a href="#">Section 30.6.3 on page 900</a>
	0x4000 5C00 - 0x4000 5FFF	1 KB	USB	<a href="#">Section 30.6.3 on page 900</a>
	0x4000 5800 - 0x4000 5BFF	1 KB	I2C2	<a href="#">Section 26.7.12 on page 698</a>
	0x4000 5400 - 0x4000 57FF	1 KB	I2C1	<a href="#">Section 26.7.12 on page 698</a>
	0x4000 5000 - 0x4000 53FF	1 KB	USART5	<a href="#">Section 27.8.12 on page 765</a>
	0x4000 4C00 - 0x4000 4FFF	1 KB	USART4	<a href="#">Section 27.8.12 on page 765</a>
	0x4000 4800 - 0x4000 4BFF	1 KB	USART3	<a href="#">Section 27.8.12 on page 765</a>
	0x4000 4400 - 0x4000 47FF	1 KB	USART2	<a href="#">Section 27.8.12 on page 765</a>
	0x4000 3C00 - 0x4000 43FF	2 KB	Reserved	-
	0x4000 3800 - 0x4000 3BFF	1 KB	SPI2	<a href="#">Section 28.9.10 on page 824</a>
	0x4000 3400 - 0x4000 37FF	1 KB	Reserved	-
	0x4000 3000 - 0x4000 33FF	1 KB	IWDG	<a href="#">Section 23.4.6 on page 582</a>
	0x4000 2C00 - 0x4000 2FFF	1 KB	WWDG	<a href="#">Section 24.5.4 on page 588</a>
	0x4000 2800 - 0x4000 2BFF	1 KB	RTC	<a href="#">Section 25.7.18 on page 628</a>
	0x4000 2400 - 0x4000 27FF	1 KB	Reserved	-
	0x4000 2000 - 0x4000 23FF	1 KB	TIM14	<a href="#">Section 19.4.13 on page 490</a>
	0x4000 1800 - 0x4000 1FFF	2 KB	Reserved	-
	0x4000 1400 - 0x4000 17FF	1 KB	TIM7	<a href="#">Section 21.4.9 on page 572</a>
	0x4000 1000 - 0x4000 13FF	1 KB	TIM6	<a href="#">Section 21.4.9 on page 572</a>
	0x4000 0800 - 0x4000 0FFF	2 KB	Reserved	-
	0x4000 0400 - 0x4000 07FF	1 KB	TIM3	<a href="#">Section 18.4.19 on page 469</a>
	0x4000 0000 - 0x4000 03FF	1 KB	TIM2	<a href="#">Section 18.4.19 on page 469</a>

**Table 2. STM32F0xx memory boundary addresses**

Device	Boundary address	Size	Memory Area	Register description
STM32F03x	0x2000 1000 - 0x3FFF FFFF	~512 MB	Reserved	-
	0x2000 0000 - 0x2000 0FFF	4 KB	SRAM	<a href="#">Section 2.3 on page 52</a>
	0x1FFF FC00 - 0x1FFF FFFF	1 KB	Reserved	-
	0x1FFF F800 - 0x1FFF FBFF	1 KB	Option bytes	<a href="#">Section 4 on page 76</a>
	0x1FFF EC00 - 0x1FFF F7FF	3 KB	System memory	-
	0x0800 8000 - 0x1FFF EBFF	~384 MB	Reserved	-
	0x0800 0000 - 0x0800 7FFF	32 KB	Main flash memory	<a href="#">Section 3 on page 56</a>
	0x0000 8000 - 0x07FF FFFF	~128 MB	Reserved	-
	0x0000 0000 - 0x0000 7FFF	32 KB	Main flash memory, system memory or SRAM depending on BOOT configuration	-
STM32F04x	0x2000 1800 - 0x3FFF FFFF	~512 MB	Reserved	-
	0x2000 0000 - 0x2000 17FF	6 KB	SRAM	<a href="#">Section 2.3 on page 52</a>
	0x1FFF FC00 - 0x1FFF FFFF	1 KB	Reserved	-
	0x1FFF F800 - 0x1FFF FBFF	1 KB	Option bytes	<a href="#">Section 4 on page 76</a>
	0x1FFF C400 - 0x1FFF F7FF	13 KB	System memory	-
	0x0801 8000 - 0x1FFF C7FF	~384 MB	Reserved	-
	0x0800 0000 - 0x0801 7FFF	32 KB	Main flash memory	<a href="#">Section 3 on page 56</a>
	0x0001 8000 - 0x07FF FFFF	~128 MB	Reserved	-
	0x0000 0000 - 0x0000 7FFF	32 KB	Main flash memory, system memory or SRAM depending on BOOT configuration	-
STM32F05x	0x2000 2000 - 0x3FFF FFFF	~512 MB	Reserved	-
	0x2000 0000 - 0x2000 1FFF	8 KB	SRAM	<a href="#">Section 2.3 on page 52</a>
	0x1FFF FC00 - 0x1FFF FFFF	1 KB	Reserved	-
	0x1FFF F800 - 0x1FFF FBFF	1 KB	Option bytes	<a href="#">Section 4 on page 76</a>
	0x1FFF EC00 - 0x1FFF F7FF	3 KB	System memory	-
	0x0801 0000 - 0x1FFF EBFF	~384 MB	Reserved	-
	0x0800 0000 - 0x0800 FFFF	64 KB	Main flash memory	<a href="#">Section 3 on page 56</a>
	0x0001 0000 - 0x07FF FFFF	~128 MB	Reserved	-
	0x0000 0000 - 0x0000 FFFF	64 KB	Main flash memory, system memory or SRAM depending on BOOT configuration	-

**Table 2. STM32F0xx memory boundary addresses (continued)**

Device	Boundary address	Size	Memory Area	Register description
STM32F07x	0x2000 4000 - 0x3FFF FFFF	~512 MB	Reserved	-
	0x2000 0000 - 0x2000 3FFF	16 KB	SRAM	<a href="#">Section 2.3 on page 52</a>
	0x1FFF F800 - 0x1FFF FFFF	2 KB	Option bytes	<a href="#">Section 4 on page 76</a>
	0x1FFF C800 - 0x1FFF F7FF	12 KB	System memory	-
	0x0802 0000 - 0x1FFF C7FF	~384 MB	Reserved	-
	0x0800 0000 - 0x0801 FFFF	128 KB	Main flash memory	<a href="#">Section 3 on page 56</a>
	0x0002 0000 - 0x07FF FFFF	~128 MB	Reserved	-
	0x0000 0000 - 0x0001 FFFF	128 KB	Main flash memory, system memory or SRAM depending on BOOT configuration	-
STM32F09x	0x2000 8000 - 0x3FFF FFFF	~512 MB	Reserved	-
	0x2000 0000 - 0x2000 7FFF	32 KB	SRAM	<a href="#">Section 2.3 on page 52</a>
	0x1FFF F800 - 0x1FFF FFFF	2 KB	Option bytes	<a href="#">Section 4 on page 76</a>
	0x1FFF D800 - 0x1FFF F7FF	8 KB	System memory	-
	0x0804 0000 - 0x1FFF D7FF	~384 MB	Reserved	-
	0x0800 0000 - 0x0803 FFFF	256 KB	Main flash memory	<a href="#">Section 3 on page 56</a>
	0x0004 0000 - 0x07FF FFFF	~128 MB	Reserved	-
	0x0000 0000 - 0x0003 FFFF	256 KB	Main flash memory, system memory or SRAM depending on BOOT configuration	-

## 2.3 Embedded SRAM

STM32F03x devices feature 4 Kbytes of static SRAM. STM32F04x devices feature

6 Kbytes of static SRAM. STM32F05x devices feature 8 Kbytes of static SRAM.

STM32F07xS devices feature 16 Kbytes of static SRAM. STM32F09x devices feature 32 Kbytes of static SRAM.

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

### Parity check

The user can enable the parity check using the option bit RAM\_PARITY\_CHECK in the user option byte (refer to [Section 4: Option bytes](#)).

The data bus width is 36 bits because 4 bits are available for parity check (1 bit per byte) in order to increase memory robustness, as required for instance by Class B or SIL norms.

The parity bits are computed and stored when writing into the SRAM. Then, they are automatically checked when reading. If one bit fails, an NMI is generated. The same error can also be linked to the BRK\_IN Break input of TIM1/15/16/17, with the

SRAM\_PARITY\_LOCK control bit in the [SYSCFG configuration register 2 \(SYSCFG\\_CFGR2\)](#). The SRAM Parity Error flag (SRAM\_PEF) is available in the [SYSCFG configuration register 2 \(SYSCFG\\_CFGR2\)](#).

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

## 2.4 Flash memory overview

The flash memory is composed of two distinct physical areas:

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

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

## 2.5 Boot configuration

In the STM32F0xx, three different boot modes can be selected through the BOOT0 pin and boot configuration bits nBOOT1, BOOT\_SEL and nBOOT0 in the User option byte, as shown in the following table.

**Table 3. Boot modes<sup>(1)</sup>**

Boot mode configuration				Mode
nBOOT1 bit	BOOT0 pin	BOOT_SEL bit	nBOOT0 bit	
x	0	1	x	<b>Main flash memory</b> is selected as boot area <sup>(2)</sup>
1	1	1	x	<b>System memory</b> is selected as boot area
0	1	1	x	<b>Embedded SRAM</b> is selected as boot area
x	x	0	1	<b>Main flash memory</b> is selected as boot area
1	x	0	0	<b>System memory</b> is selected as boot area
0	x	0	0	<b>Embedded SRAM</b> is selected as boot area

1. Grey options are available on STM32F04x and STM32F09x devices only.
2. For STM32F04x and STM32F09x devices, see also Empty check description.

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

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

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

- Boot from main flash memory: the main flash memory is aliased in the boot memory space (0x0000 0000), but still accessible from its original memory space (0x0800 0000). In other words, the flash memory contents can be accessed starting from address 0x0000 0000 or 0x0800 0000.
- Boot from system memory: the system memory is aliased in the boot memory space (0x0000 0000), but still accessible from its original memory space (0x1FFF EC00 on STM32F03x and STM32F05x devices, 0x1FFF C400 on STM32F04x devices, 0x1FFF C800 on STM32F07x and 0x1FFF D800 on STM32F09x devices).
- Boot from the embedded SRAM: the SRAM is aliased in the boot memory space (0x0000 0000), but it is still accessible from its original memory space (0x2000 0000).

### Empty check

On STM32F04x and STM32F09x devices only, internal empty check flag is implemented to allow easy programming of virgin devices by the boot loader. This flag is used when BOOT0 pin is defining Main flash memory as the target boot area. When the flag is set, the device is considered as empty and System memory (boot loader) is selected instead of the main flash as a boot area to allow user to program the flash memory.

This flag is updated only during the loading of option bytes: it is set when the content of the address 0x08000 0000 is read as 0xFFFF FFFF, otherwise it is cleared. It means a power reset or setting of OBL\_LAUNCH bit in FLASH\_CR register is needed to clear this flag after programming of a virgin device to execute user code after System reset.

**Note:** *If the device is programmed for a first time but the option bytes are not reloaded, the device still selects System memory as a boot area after a System reset. In the STM32F04x, the boot loader code is able to detect this situation. It then changes the boot memory mapping to main flash and performs a jump to user code programmed there. In the STM32F09x, a POR must be performed or the Option bytes reloaded before applying the system reset.*

### Physical remap

Once the boot mode is selected, the application software can modify the memory accessible in the code area. This modification is performed by programming the MEM\_MODE bits in the [SYSCFG configuration register 1 \(SYSCFG\\_CFGR1\)](#). Unlike Cortex® M3 and M4, the M0 CPU does not support the vector table relocation. For application code which is located in a different address than 0x0800 0000, some additional code must be added in order to be able to serve the application interrupts. A solution is to relocate by software the vector table to the internal SRAM:

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

This operation should be done at the initialization phase of the application. Please refer to AN4065 and attached IAP code from [www.st.com](http://www.st.com) for more details.

### Embedded boot loader

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

- USART on pins PA9/PA10, PA14/PA15 or PA2/PA3
- I2C on pins PB6/PB7 (STM32F04xxx, STM32F07xxx and STM32F09xxx devices only)
- USB DFU interface (STM32F04xxx and STM32F07xxx devices only)

For further details, refer to the application note AN2606.

## 3 Embedded flash memory

### 3.1 Flash main features

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

Flash memory interface features:

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

### 3.2 Flash memory functional description

#### 3.2.1 Flash memory organization

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

The memory organization of STM32F03x, STM32F04x and STM32F05x devices is based on a main flash memory block containing up to 64 pages of 1 Kbyte or up to 16 sectors of 4 Kbytes (4 pages). The sector is the granularity of the write protection (see [Section 3.3](#)).

The memory organization of STM32F07x and STM32F09x devices is based on a main flash memory block containing up to 128 pages of 2 Kbytes or up to 64 sectors of 4 Kbytes (2 pages). The sector is the granularity of the write protection (see [Section 3.3](#)).

The information block is divided into two parts:

1. System memory: used to boot the device in System memory boot mode. The area is reserved for use by STMicroelectronics and contains the boot loader, which is used to reprogram the flash memory through the selected communication interface. It is programmed by ST when the device is manufactured, and protected against spurious write/erase operations. For further details refer to AN2606.
2. Option byte

**Table 4. Flash memory organization (STM32F03x, STM32F04x and STM32F05x devices)**

Flash area	Flash memory addresses	Size (byte)	Name	Description
Main flash memory	0x0800 0000 - 0x0800 03FF	1 Kbyte	Page 0	Sector 0
	0x0800 0400 - 0x0800 07FF	1 Kbyte	Page 1	
	0x0800 0800 - 0x0800 0BFF	1 Kbyte	Page 2	
	0x0800 0C00 - 0x0800 0FFF	1 Kbyte	Page 3	
	.	.	.	.
	.	.	.	.
	.	.	.	.
	0x0800 7000 - 0x0800 73FF	1 Kbyte	Page 28	Sector 7 <sup>(1)</sup>
	0x0800 7400 - 0x0800 77FF	1 Kbyte	Page 29	
	0x0800 7800 - 0x0800 7BFF	1 Kbyte	Page 30	
	0x0800 7C00 - 0x0800 7FFF	1 Kbyte	Page 31	
	.	.	.	.
	.	.	.	.
	.	.	.	.
Information block	0x0800 F000 - 0x0800 F3FF	1 Kbyte	Page 60	Sector 15
	0x0800 F400 - 0x0800 F7FF	1 Kbyte	Page 61	
	0x0800 F800 - 0x0800 FBFF	1 Kbyte	Page 62	
	0x0800 FC00 - 0x0800 FFFF	1 Kbyte	Page 63	
	0x1FFF EC00 - 0x1FFF F7FF	3 Kbyte <sup>(2)</sup>	-	System memory
	0x1FFF C400 - 0x1FFF F7FF	13 Kbyte <sup>(3)</sup>	-	System memory
	0x1FFF F800 - 0x1FFF F80F	2 x 8 byte	-	Option byte

1. Main flash memory space of STM32F03x and STM32F04x devices is limited to sector 7.

2. STM32F03x and STM32F05x devices.

3. STM32F04x devices.

**Table 5. Flash memory organization (STM32F07x, STM32F09x devices)**

Flash area	Flash memory addresses	Size (byte)	Name	Description
Main flash memory	0x0800 0000 - 0x0800 07FF	2 Kbytes	Page 0	Sector 0
	0x0800 0800 - 0x0800 0FFF	2 Kbytes	Page 1	
	.	.	.	.
	.	.	.	.
	0x0801 F000 - 0x0801 F7FF	2 Kbytes	Page 62	Sector 31 <sup>(1)</sup>
	0x0801 F800 - 0x0801 FFFF	2 Kbytes	Page 63	
	.	.	.	.
	0x0803 F000 - 0x0803 F7FF	2 Kbytes	Page 126	-
Information block	0x0803 F800 - 0x0803 FFFF	2 Kbytes	Page 127	-
	0xFFFF C800 - 0xFFFF F7FF	12 Kbytes <sup>(2)</sup>	-	System memory
	0xFFFF D800 - 0xFFFF F7FF	8 Kbytes <sup>(3)</sup>	-	System memory
	0xFFFF F800 - 0xFFFF F80F	2 x 8 byte	-	Option byte

1. The main flash memory space of STM32F07x is limited to sector 31.

2. STM32F07x devices only.

3. STM32F09x devices only.

## Read operations

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

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

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

## Instruction fetch

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

## Prefetch buffer

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

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

However the prefetch buffer has an impact on the performance only when the wait state number is 1. In the other case (no wait state) the performance remains the same whatever the prefetch buffer status. There could be some impacts on the power consumption but this is strongly dependent from the actual application code.

### Prefetch controller

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

The prefetch buffer is usually switched on/off during the initialization routine, while the microcontroller is running on the internal 8 MHz RC (HSI) oscillator.

### Access latency

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

## 3.2.2 Flash program and erase operations

The STM32F0xx embedded flash memory can be programmed using in-circuit programming or in-application programming.

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

In contrast to the ICP method, **in-application programming (IAP)** can use any communication interface supported by the microcontroller (I/Os, USB, CAN, USART, I<sup>2</sup>C, SPI, etc.) to download programming data into memory. IAP allows the user to re-program the flash memory while the application is running. Nevertheless, part of the application has to have been previously programmed in the flash memory using ICP.

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

- Key register (FLASH\_KEYR)
- Option byte key register (FLASH\_OPTKEYR)
- Flash control register (FLASH\_CR)
- Flash status register (FLASH\_SR)
- Flash address register (FLASH\_AR)
- Option byte register (FLASH\_OBR)
- Write protection register (FLASH\_WRPR)

An ongoing flash memory operation does not block the CPU as long as the CPU does not access the flash memory.

On the contrary, during a program/erase operation to the flash memory, any attempt to read the flash memory stalls the bus. The read operation proceeds correctly once the

program/erase operation has completed. This means that code or data fetches cannot be made while a program/erase operation is ongoing.

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

### Unlocking the flash memory

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

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

Any wrong sequence locks up the FLASH\_CR register until the next reset.

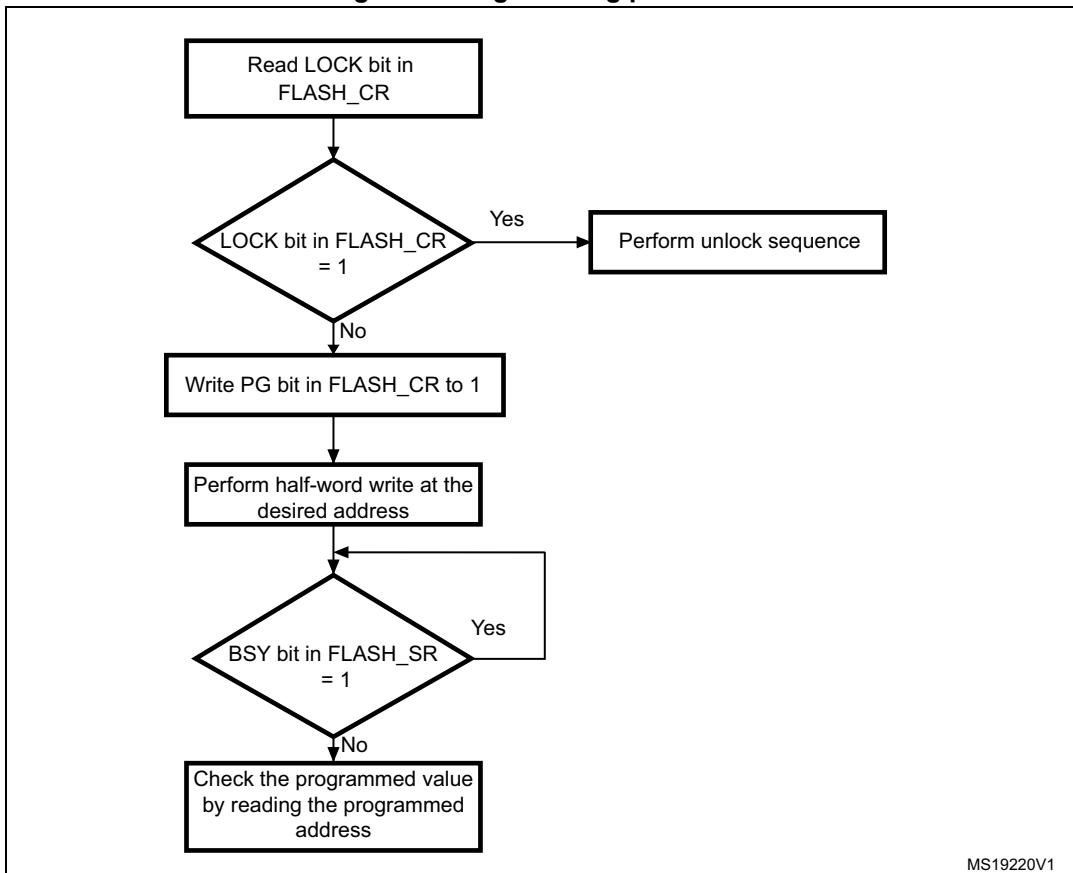
In the case of a wrong key sequence, a bus error is detected and a Hard Fault interrupt is generated. This is done after the first write cycle if KEY1 does not match, or during the second write cycle if KEY1 has been correctly written but KEY2 does not match.

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

For code example refer to the Appendix section [A.2.1: Flash memory unlocking sequence](#).

### Main flash memory programming

The main flash memory can be programmed 16 bits at a time. The program operation is started when the CPU writes a half-word into a main flash memory address with the PG bit of the FLASH\_CR register set. Any attempt to write data that are not half-word long results in a bus error generating a Hard Fault interrupt.

**Figure 3. Programming procedure**

MS19220V1

The flash memory interface preliminarily reads the value at the addressed main flash memory location and checks that it has been erased. If not, the program operation is skipped and a warning is issued by the PGERR bit in FLASH\_SR register. The only exception to this is when 0x0000 is programmed. In this case, the location is correctly programmed to 0x0000 and the PGERR bit is not set.

If the addressed main flash memory location is write-protected by the FLASH\_WRPR register, the program operation is skipped and a warning is issued by the WRPRTERR bit in the FLASH\_SR register. The end of the program operation is indicated by the EOP bit in the FLASH\_SR register.

The main flash memory programming sequence in standard mode is as follows:

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

**Note:** *The registers are not accessible in write mode when the BSY bit of the FLASH\_SR register is set.*

For code example refer to the Appendix section [A.2.2: Main Flash programming sequence](#).

### Flash memory erase

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

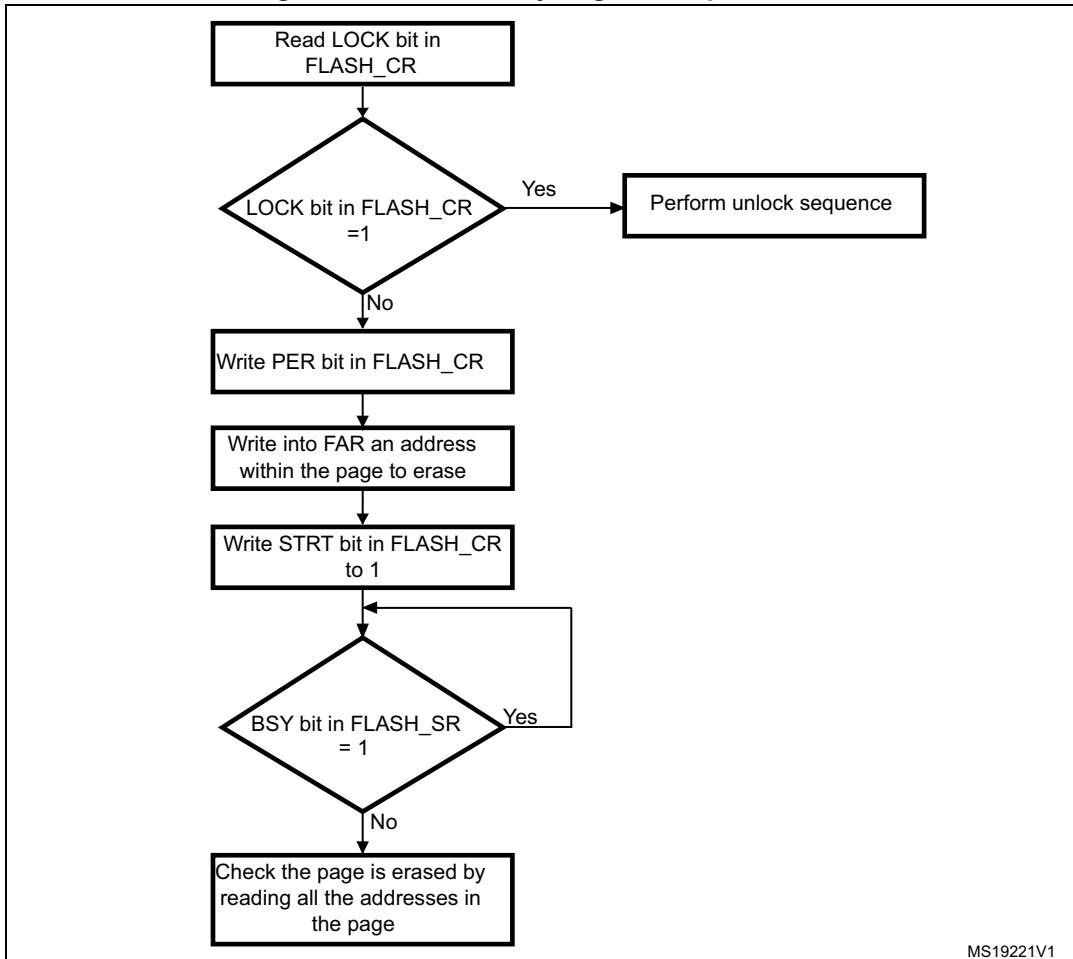
#### Page erase

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

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

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

For code example refer to the Appendix section [A.2.3: Page erase sequence](#).

**Figure 4. Flash memory Page erase procedure**

## Mass erase

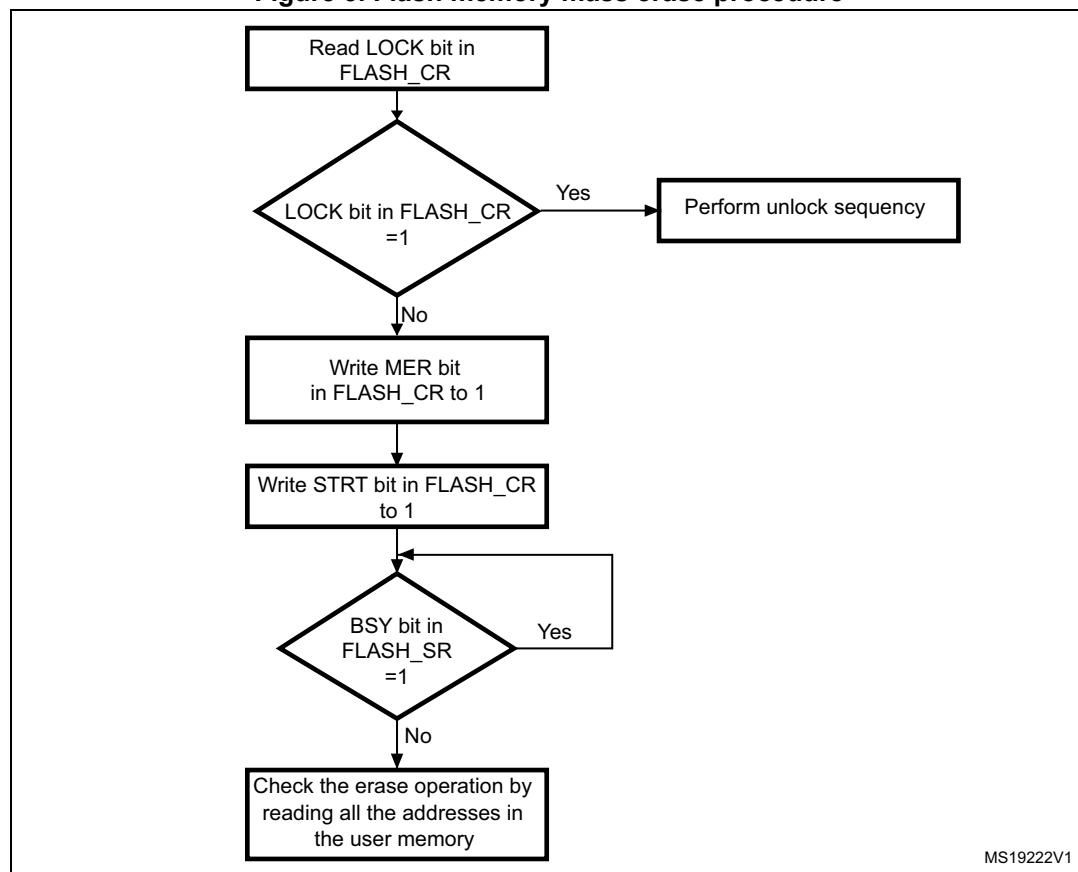
The mass erase command can be used to completely erase the pages of the main flash memory. The information block is unaffected by this procedure. The following sequence is recommended:

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

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

For code example refer to the Appendix section [A.2.4: Mass erase sequence](#).

**Figure 5. Flash memory mass erase procedure**



MS19222V1

## Option byte programming

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

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

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

The sequence is as follows:

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

For code example refer to the Appendix section [A.2.6: Option byte programming sequence](#).

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

## Erase procedure

The option byte erase sequence is as follows:

1. Check that no flash memory operation is ongoing by reading the BSY bit in the FLASH\_SR register
2. Unlock the OPTWRE bit in the FLASH\_CR register
3. Set the OPTER bit in the FLASH\_CR register
4. Set the STRT bit in the FLASH\_CR register
5. Wait for the BSY bit to be reset
6. Read the erased option byte and verify

For code example refer to the Appendix section [A.2.7: Option byte erasing sequence](#).

### 3.3 Memory protection

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

#### 3.3.1 Read protection

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

*Note:* If the read protection is set while the debugger is still connected through SWD, apply a POR (power-on reset) instead of a system reset.

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

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

**Table 6. Flash memory read protection status**

RDP byte value	RDP complement value	Read protection level
0xAA	0x55	Level 0 (ST production configuration)
Any value except 0xAA or 0xCC	Any value (not necessarily complementary) except 0x55 and 0x33	Level 1
0xCC	0x33	Level 2

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

#### Level 0: no protection

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

The option byte are as well accessible by all operations.

#### Level 1: read protection

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

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

attempted program/erase operation sets the PGERR flag of flash status register (FLASH\_SR).

When the RPD is reprogrammed to the value 0xAA to move back to Level 0, a mass erase of the main flash memory is performed and the backup registers (RTC\_BKPxR in the RTC) are reset.

### Level 2: no debug

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

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

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

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

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

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

Area	Protection level	User execution			Debug / Boot from RAM / Boot from System memory		
		Read	Write	Erase	Read	Write	Erase
Main flash memory	1	Yes	Yes	Yes	No	No	No <sup>(4)</sup>
	2	Yes	Yes	Yes	N/A <sup>(1)</sup>	N/A <sup>(1)</sup>	N/A <sup>(1)</sup>
System memory <sup>(2)</sup>	1	Yes <sup>(3)</sup>	No	No	Yes	No	No
	2	Yes <sup>(3)</sup>	No	No	N/A <sup>(1)</sup>	N/A <sup>(1)</sup>	N/A <sup>(1)</sup>
Option byte	1	Yes	Yes <sup>(4)</sup>	Yes	Yes <sup>(5)</sup>	Yes <sup>(4)(5)</sup>	Yes
	2	Yes	Yes <sup>(6)</sup>	No	N/A <sup>(1)</sup>	N/A <sup>(1)</sup>	N/A <sup>(1)</sup>
Backup registers <sup>(7)</sup>	1	Yes	Yes	N/A	No	No	No
	2	Yes	Yes	N/A	N/A <sup>(1)</sup>	N/A <sup>(1)</sup>	N/A <sup>(1)</sup>

1. When the protection level 2 is active, the Debug port, the boot from RAM and the boot from System memory are disabled.
2. The system memory is only read-accessible, whatever the protection level (0, 1 or 2) and execution mode.
3. Under RDP level 1 and level 2, the bootloader can be accessed by a jump to it. However it does not execute any command other than GetCmd/GetID/GetVersion.
4. The main flash memory is erased when the RDP option byte is changed from level 1 to level 0 (0xAA).
5. When the RDP level 1 is active, the embedded boot loader does not allow to read or write the Option byte, except to remove the RDP protection (move from level 1 to level 0).
6. All option byte can be programmed, except the RDP byte.
7. These registers are reset when moving from RDP level 1 to level 0.

### Changing read protection level

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

By programming the 0xCC value in the RDP byte, it is possible to go to level 2 either directly from level 0 or from level 1.

On the contrary, the change to level 0 (no protection) is not possible without a main flash memory Mass erase operation. This Mass erase is generated as soon as 0xAA is programmed in the RDP byte.

**Note:** *When the mass erase command is used, the backup registers (RTC\_BKPxR in the RTC) are also reset.*

*To validate the protection level change, the option bytes must be reloaded through the “OBL\_LAUNCH” bit in flash control register.*

### 3.3.2 Write protection

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

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

#### Write unprotection

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

- Case 1: Read protection disabled after the write unprotection:
  - Erase the entire option byte area by using the OPTER bit in the flash memory control register (FLASH\_CR).
  - Program the code 0xAA in the RDP byte to unprotect the memory. This operation forces a mass erase of the main flash memory.
  - Set the OBL\_LAUNCH bit in the flash control register (FLASH\_CR) to reload the option byte (and the new WRP[1:0] byte), and to disable the write protection.
- Case 2: Read protection maintained active after the write unprotection, useful for in-application programming with a user boot loader:
  - Erase the entire option byte area by using the OPTER bit in the flash memory control register (FLASH\_CR).
  - Set the OBL\_LAUNCH bit in the flash control register (FLASH\_CR) to reload the option byte (and the new WRP[1:0] byte), and to disable the write protection.

### 3.3.3 Option byte write protection

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

## 3.4 Flash interrupts

**Table 8. Flash interrupt request**

Interrupt event	Event flag	Enable control bit
End of operation	EOP	EOPIE
Write protection error	WRPRTERR	ERRIE
Programming error	PGERR	ERRIE

## 3.5 Flash register description

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

### 3.5.1 Flash access control register (FLASH\_ACR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	PRFT BS	PRFT BE	Res.	LATENCY[2:0]											
										r	rw		rw	rw	rw

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

Bit 5 **PRFTBS**: Prefetch buffer status

This bit provides the status of the prefetch buffer.

0: Prefetch buffer is disabled

1: Prefetch buffer is enabled

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

Bit 4 **PRFTBE**: Prefetch buffer enable

0: Prefetch is disabled

1: Prefetch is enabled

Bit 3 Reserved, must be kept at reset value.

Bits 2:0 **LATENCY[2:0]**: Latency

These bits represent the ratio of the SYSCLK (system clock) period to the flash access time.

000: Zero wait state, if SYSCLK  $\leq$  24 MHz

001: One wait state, if 24 MHz  $<$  SYSCLK  $\leq$  48 MHz

### 3.5.2 Flash key register (FLASH\_KEYR)

Address offset: 0x04

Reset value: 0XXXXX XXXX

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

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

Bits 31:0 **FKEY**: Flash key

These bits represent the keys to unlock the flash.

### 3.5.3 Flash option key register (FLASH\_OPTKEYR)

Address offset: 0x08

Reset value: 0XXXXX XXXX

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

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

Bits 31:0 **OPTKEY**: Option byte key

These bits represent the keys to unlock the OPTWRE.

### 3.5.4 Flash status register (FLASH\_SR)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	EOP	WRPRT ERR	Res.	PG ERR	Res.	BSY									
										rc_w1	rc_w1		rc_w1		r

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

Bit 5 **EOP**: End of operation

Set by hardware when a flash operation (programming / erase) is completed.

Reset by writing 1.

*Note: EOP is asserted at the end of each successful program or erase operation*

Bit 4 **WRPRTERR**: Write protection error

Set by hardware when programming a write-protected address of the flash memory.

Reset by writing 1.

Bit 3 Reserved, must be kept at reset value.

Bit 2 **PGERR**: Programming error

Set by hardware when an address to be programmed contains a value different from '0xFFFF' before programming.

Reset by writing 1.

*Note: The STRT bit in the FLASH\_CR register should be reset before starting a programming operation.*

Bit 1 Reserved, must be kept at reset value.

Bit 0 **BSY**: Busy

This indicates that a flash operation is in progress. This is set on the beginning of a flash operation and reset when the operation finishes or when an error occurs.

### 3.5.5 Flash control register (FLASH\_CR)

Address offset: 0x10

Reset value: 0x0000 0080

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

Bits 31:14 Reserved, must be kept at reset value.

Bit 13 **OBL\_LAUNCH**: Force option byte loading

When set to 1, this bit forces the option byte reloading. This operation generates a system reset.

- 0: Inactive
- 1: Active

Bit 12 **EOPIE**: End of operation interrupt enable

This bit enables the interrupt generation when the EOP bit in the FLASH\_SR register goes to 1.

- 0: Interrupt generation disabled
- 1: Interrupt generation enabled

Bit 11 Reserved, must be kept at reset value

Bit 10 **ERRIE**: Error interrupt enable

This bit enables the interrupt generation on an error when PGERR / WRPRTERR are set in the FLASH\_SR register.

- 0: Interrupt generation disabled
- 1: Interrupt generation enabled

Bit 9 **OPTWRE**: Option byte write enable

When set, the option byte can be programmed. This bit is set on writing the correct key sequence to the FLASH\_OPTKEYR register.

This bit can be reset by software

Bit 8 Reserved, must be kept at reset value.

Bit 7 **LOCK**: Lock

Write to 1 only. When it is set, it indicates that the flash is locked. This bit is reset by hardware after detecting the unlock sequence.

In the event of unsuccessful unlock operation, this bit remains set until the next reset.

Bit 6 **STRT**: Start

This bit triggers an ERASE operation when set. This bit is set only by software and reset when the BSY bit is reset.

Bit 5 **OPTER**: Option byte erase

Option byte erase chosen.

Bit 4 **OPTPG**: Option byte programming

Option byte programming chosen.

Bit 3 Reserved, must be kept at reset value.

Bit 2 **MER**: Mass erase

Erase of all user pages chosen.

Bit 1 **PER**: Page erase

Page erase chosen.

Bit 0 **PG**: Programming

Flash programming chosen.

### 3.5.6 Flash address register (FLASH\_AR)

Address offset: 0x14

Reset value: 0x0000 0000

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

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

Bits 31:0 **FAR**: Flash address

Chooses the address to program when programming is selected, or a page to erase when Page erase is selected.

*Note:* Write access to this register is blocked when the BSY bit in the FLASH\_SR register is set.

### 3.5.7 Flash Option byte register (FLASH\_OBR)

Address offset 0x1C

Reset value: 0xXXXX XX0X

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA1								DATA0							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BOOT_SEL	RAM_PARITY_CHECK	VDDA_MONITOR	nBOOT1	nBOOT0	nRST_STDBY	nRST_STOP	WDG_SW	Res.	Res.	Res.	Res.	Res.	RDPRT[1:0]	OPTERR	
r	r	r	r	r	r	r	r						r	r	r

Bits 31:24 **DATA1**

Bits 23:16 **DATA0**

Bits 15:8 User option bytes:

Bit 15: **BOOT\_SEL** (available on STM32F04x and STM32F09x devices only)

Bit 14: **RAM\_PARITY\_CHECK**

Bit 13: **VDDA\_MONITOR**

Bit 12: **nBOOT1**

Bit 11: **nBOOT0** (available on STM32F04x and STM32F09x devices only)

Bit 10: **nRST\_STDBY**

Bit 9: **nRST\_STOP**

Bit 8: **WDG\_SW**

Bits 7:3 Reserved, must be kept at reset value.

Bits 2:1 **RDPRT[1:0]**: Read protection level status

00: Read protection level 0 is enabled (ST production configuration)

01: Read protection level 1 is enabled

11: Read protection level 2 is enabled.

Bit 0 **OPTERR**: Option byte error

When set, this indicates that the loaded option byte and its complement do not match. The corresponding byte is set to 0xFF in respective FLASH\_OBR or FLASH\_WRPR register.

### 3.5.8 Write protection register (FLASH\_WRPR)

Address offset: 0x20

Reset value: 0xFFFFFFFF

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

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

Bits 31:0 **WRP**: Write protect

This register contains the write-protection option byte loaded by the OBL.

### **3.5.9 Flash register map**

**Table 9. Flash interface - Register map and reset values**

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 4 Option bytes

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

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

**Table 10. Option byte format**

31-24	23-16	15 -8	7-0
Complemented option byte 1	Option byte 1	Complemented option byte 0	Option byte 0

The organization of these bytes inside the information block is as shown in [Table 11](#).

The option byte can be read from the memory locations listed in [Table 11](#) or from the Option byte register (FLASH\_OBR).

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

**Table 11. Option byte organization**

Address	[31:24]	[23:16]	[15:8]	[7:0]
0x1FFF F800	nUSER	USER	nRDP	RDP
0x1FFF F804	nData1	Data1	nData0	Data0
0x1FFF F808	nWRP1	WRP1	nWRP0	WRP0
0x1FFF F80C	nWRP3	WRP3	nWRP2	WRP2

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

## 4.1 Option byte description

### 4.1.1 User and read protection option byte

Flash memory address: 0x1FFF F800

ST production value: 0x00FF 55AA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
nUSER										USER						
										BOOT_SEL	RAM_PARITY_CHECK	VDDA_MONITOR	nBOOT1	nBOOT0	nRST_STDBY	nRST_STOP
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
nRDP										RDP						
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 23:16 **USER**: User option byte (stored in FLASH\_OBR[15:8])

Bit 23: **BOOT\_SEL**

0: BOOT0 signal is defined by nBOOT0 option bit

1: BOOT0 signal is defined by BOOT0 pin value (legacy mode)

Available on STM32F04x and STM32F09x devices only. Considered as “1” on other devices.

Bit 22: **RAM\_PARITY\_CHECK**

0: RAM parity check enabled

1: RAM parity check disabled

Bit 21: **VDDA\_MONITOR**

0: V<sub>DDA</sub> power supply supervisor disabled

1: V<sub>DDA</sub> power supply supervisor enabled

Bit 20: **nBOOT1**

Together with the BOOT0 signal, it selects the device boot mode. Refer to [Section 2.5: Boot configuration](#) for more details.

Bit 19: **nBOOT0**

When BOOT\_SEL is cleared, nBOOT0 bit defines BOOT0 signal value used to select the device boot mode. Refer to [Section 2.5: Boot configuration](#) for more details.

Available on STM32F04x and STM32F09x devices only.

Bit 18: **nRST\_STDBY**

0: Reset generated when entering Standby mode.

1: No reset generated.

Bit 17: **nRST\_STOP**

0: Reset generated when entering Stop mode

1: No reset generated

Bit 16: **WDG\_SW**

0: Hardware watchdog

1: Software watchdog

Bits 15:8 **nRDP**: Read protection option byte complement

Bits 7:0 **RDP**: Read protection option byte

The value of this byte defines the flash memory protection level

0xAA: level 0 (ST production configuration)

0XX (except 0xAA & 0xCC): Level 1

0xCC: Level 2

*Note: Read protection level status is stored in bits RDPRT[1:0] of the Flash Option byte register (FLASH\_OBR). For more details about read protection, refer to [Section 3.3.1: Read protection](#).*

#### 4.1.2 User data option byte

Flash memory address: 0x1FFF F804

ST production value: 0x00FF 00FF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
nData1								Data1							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
nData0								Data0							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **nData1**: User data byte 1 complement

Bits 23:16 **Data1**: User data byte 1 value (stored in FLASH\_OBR[31:24])

Bits 15:8 **nData0**: User data byte 0 complement

Bits 7:0 **Data0**: User data byte 0 value (stored in FLASH\_OBR[23:16])

#### 4.1.3 Write protection option byte

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

For STM32F03x, STM32F04x, STM32F05x and STM32F07x devices, WRP bits from 0 to 31 are protecting the flash memory by sector of 4 kB.

For STM32F09x devices, WRP bits from 0 to 30 are protecting the first 124 kB by sector of 4 kB and the bit 31 is protecting the last 132 kB.

Refer to [Section 3.3.2: Write protection](#) for more details.

Flash memory address: 0x1FFF F808

ST production value: 0x00FF 00FF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
nWRP1								WRP1							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
nWRP0								WRP0							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **nWRP1**: Flash memory write protection option byte 1 complement

Bits 23:16 **WRP1**: Flash memory write protection option byte 1 value (stored in FLASH\_WRPR[15:8])

Bits 15:8 **nWRP0**: Flash memory write protection option byte 0 complement

Bits 7:0 **WRP0**: Flash memory write protection option byte 0 value (stored in FLASH\_WRPR[7:0])

**Note:** STM32F03x and STM32F04x devices embed WRP0 and nWRP0 only.

The following Option byte are available on STM32F07x and STM32F09x devices only.

Flash memory address: 0x1FFF F80C

ST production value: 0x00FF 00FF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
nWRP3								WRP3							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
nWRP2								WRP2							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **nWRP3**: Flash memory write protection option byte 1 complement

Bits 23:16 **WRP3**: Flash memory write protection option byte 1 value (stored in FLASH\_WRPR[15:8])

Bits 15:8 **nWRP2**: Flash memory write protection option byte 0 complement

Bits 7:0 **WRP2**: Flash memory write protection option byte 0 value (stored in FLASH\_WRPR[7:0])

#### 4.1.4 Option byte map

The following table summarizes the option bytes.

**Table 12. Option byte map and ST production values**

Offset	Option byte	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	User and read protection	nUSER								USER		nRDP								RDP													
	ST production value	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0	1	1	0	1	0	1	0		
0x04	User data	nData1								Data1		nData0								Data0													
	ST production value	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1		
0x08	Write protection	nWRP1								WRP1		nWRP0								WRP0													
	ST production value	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1		
0x0C	Write protection	nWRP3								WRP3		nWRP2								WRP2													
	ST production value	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1		

## 5 Power control (PWR)

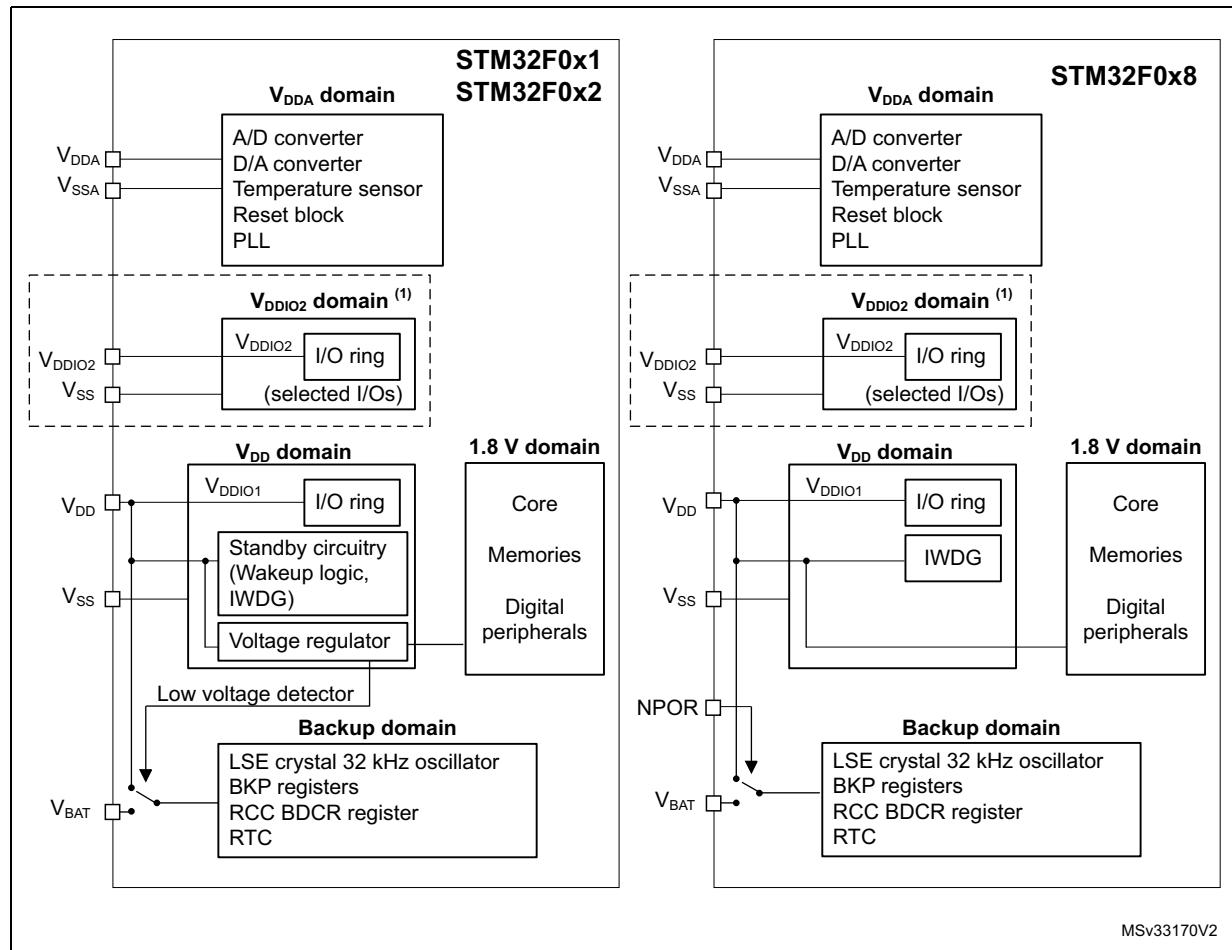
### 5.1 Power supplies

The STM32F0x1/F0x2 subfamily embeds a voltage regulator in order to supply the internal 1.8 V digital power domain, unlike the STM32F0x8 subfamily where the stable 1.8 V must be delivered by the application.

- The STM32F0x1/F0x2 devices require a 2.0 V - 3.6 V operating supply voltage ( $V_{DD}$ ) and a 2.0 V - 3.6 V analog supply voltage ( $V_{DDA}$ ).
- The STM32F0x8 devices require a 1.8 V  $\pm$  8% operating supply voltage ( $V_{DD}$ ), and a 1.65 V - 3.6 V analog supply voltage ( $V_{DDA}$ ).

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

**Figure 6. Power supply overview**



1. Available only on STM32F04x, STM32F07x and STM32F09x devices.

### 5.1.1 Independent A/D and D/A converter supply and reference voltage

To improve conversion accuracy and to extend the supply flexibility, the ADC and the DAC have an independent power supply that can be separately filtered and shielded from noise on the PCB.

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

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

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

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

### 5.1.2 Independent I/O supply rail

For better supply flexibility on STM32F04x, STM32F07x and STM32F09x devices, portions of the I/Os are supplied from a separate supply rail. Power supply for this rail can range from 1.65 to 3.6 V and is provided externally through the  $V_{DDIO2}$  pin. The  $V_{DDIO2}$  voltage level is completely independent from  $V_{DD}$  or  $V_{DDA}$ , but it must not be provided without a valid operating supply on the  $V_{DD}$  pin. Refer to the pinout diagrams or tables in related device datasheets for concerned I/Os list.

The  $V_{DDIO2}$  supply is monitored and compared with the internal reference voltage ( $V_{REFINT}$ ). When the  $V_{DDIO2}$  is below this threshold, all the I/Os supplied from this rail are disabled by hardware. The output of this comparator is connected to EXTI line 31 and it can be used to generate an interrupt.

### 5.1.3 Battery backup domain

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

The  $V_{BAT}$  pin powers the RTC unit, the LSE oscillator and the PC13 to PC15 IOs, allowing the RTC to operate even when the main power supply is turned off. The switch to the  $V_{BAT}$  supply is controlled by the Power Down Reset embedded in the Reset block or (on STM32F0x8 devices) by the external NPOR signal.

---

**Warning:** During  $t_{RSTTEMPO}$  (temporization at  $V_{DD}$  startup) or after a PDR is detected, the power switch between  $V_{BAT}$  and  $V_{DD}$  remains connected to  $V_{BAT}$ .

During the startup phase, if  $V_{DD}$  is established in less than  $t_{RSTTEMPO}$  (Refer to the datasheet for the value of  $t_{RSTTEMPO}$ ) and  $V_{DD} > V_{BAT} + 0.6$  V, a current may be injected into  $V_{BAT}$  through an internal diode connected between  $V_{DD}$  and the power switch ( $V_{BAT}$ ).

If the power supply/battery connected to the  $V_{BAT}$  pin cannot support

---

**this current injection, it is strongly recommended to connect an external low-drop diode between this power supply and the  $V_{BAT}$  pin.**

---

If no external battery is used in the application, it is recommended to connect  $V_{BAT}$  externally to  $V_{DD}$  with a 100 nF external ceramic decoupling capacitor (for more details refer to AN4080).

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

- PC13, PC14 and PC15 can be used as GPIO pins
- PC13, PC14 and PC15 can be configured by RTC or LSE (refer to [Section 25.4: RTC functional description on page 591](#))

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

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

- PC13, PC14 and PC15 can be controlled only by RTC or LSE (refer to [Section 25.4: RTC functional description on page 591](#))

## 5.1.4 Voltage regulator

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

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

**Note:** *In STM32F0x8 devices, the voltage regulator is bypassed and the microcontroller must be powered from a nominal  $V_{DD} = 1.8 \text{ V} \pm 8\%$  supply.*

## 5.2 Power supply supervisor

### 5.2.1 Power on reset (POR) / power down reset (PDR)

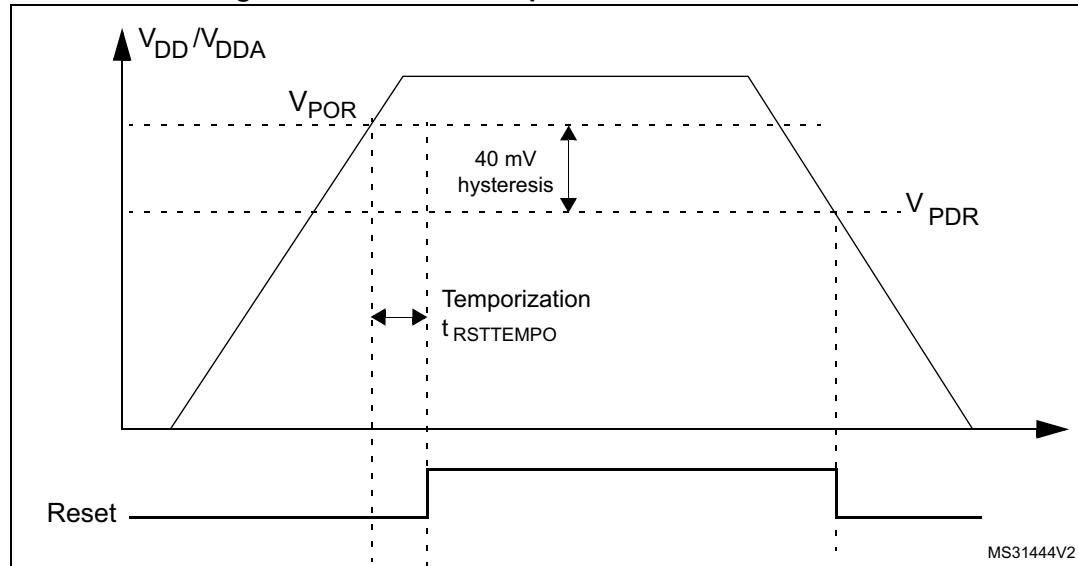
STM32F0x1xx and STM32F0x2xx devices feature integrated power-on reset (POR) and power-down reset (PDR) circuits, which are always active and ensure proper operation above a threshold of 2 V.

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

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

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

**Figure 7. Power on reset/power down reset waveform**



### External NPOR signal

In STM32F0x8 devices, the PB2 I/O (or PB1 on small packages) is not available and is replaced by the NPOR functionality used for power on reset.

To guarantee a proper power on and power down reset to the device, the NPOR pin must be held low until  $V_{DD}$  is stable or before turning off the supply. When  $V_{DD}$  is stable, the reset state can be exited by putting the NPOR pin in high impedance. The NPOR pin has an internal pull-up connected to  $V_{DDA}$ .

## 5.2.2 Programmable voltage detector (PVD)

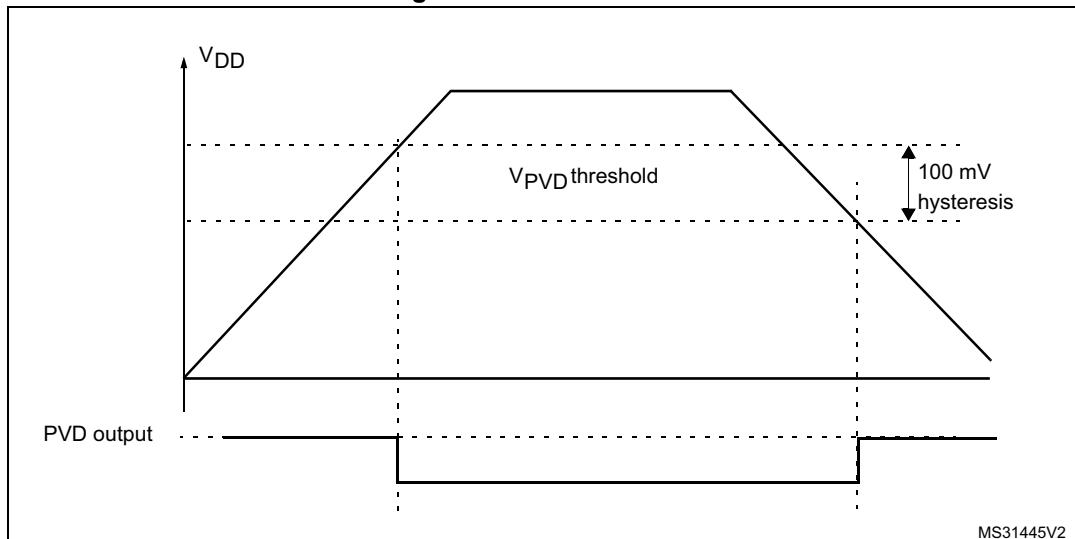
STM32F0x1xx and STM32F0x2xx can use the PVD to monitor the  $V_{DD}$  power supply by comparing it to a threshold selected by the PLS[2:0] bits in the [Power control register \(PWR\\_CR\)](#).

The PVD is enabled by setting the PVDE bit.

A PVDO flag is available, in the [Power control/status register \(PWR\\_CSR\)](#), to indicate if  $V_{DD}$  is higher or lower than the PVD threshold. This event is internally connected to the EXTI line16 and can generate an interrupt if enabled through the EXTI registers. The PVD output interrupt can be generated when  $V_{DD}$  drops below the PVD threshold and/or when  $V_{DD}$

rises above the PVD threshold depending on EXTI line16 rising/falling edge configuration. As an example the service routine could perform emergency shutdown tasks.

**Figure 8. PVD thresholds**



### 5.3 Low-power modes

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

The device features three low-power modes:

- Sleep mode (CPU clock off, all peripherals including Cortex®-M0 core peripherals like NVIC, SysTick, etc. are kept running)
- Stop mode (all clocks are stopped)
- Standby mode (1.8V domain powered-off)

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

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

**Table 13. Low-power mode summary**

Mode name	Entry	Wake-up	Effect on 1.8 V domain clocks	Effect on V <sub>DD</sub> domain clocks	Voltage regulator
Sleep (Sleep now or Sleep-on - exit)	WFI	Any interrupt	CPU clock OFF no effect on other clocks or analog clock sources	None	ON
	WFE	Wake-up event			
Stop	PDSS and LPDS bits + SLEEPDEEP bit + WFI or WFE	Any EXTI line (configured in the EXTI registers) Specific communication peripherals on reception events (CEC, USART, I2C)	All 1.8V domain clocks OFF	HSI and HSE oscillators OFF	ON or in low-power mode (depends on <i>Power control register (PWR_CR)</i> )
					OFF
Standby	PDSS bit + SLEEPDEEP bit + WFI or WFE	WKUP pin rising edge, RTC alarm, external reset in NRST pin, IWDG reset			

**Caution:** On STM32F0x8 devices, the Stop mode is available, but it is meaningless to distinguish between voltage regulator in low-power mode and voltage regulator in Run mode because the regulator is not used and the core is supplied directly from an external source. Consequently, the Standby mode is not available on those devices.

### 5.3.1 Slowing down system clocks

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

For more details refer to [Section 6.4.2: Clock configuration register \(RCC\\_CFGR\)](#).

### 5.3.2 Peripheral clock gating

In Run mode, the AHB clock (HCLK) and the APB clock (PCLK) for individual peripherals and memories can be stopped at any time to reduce power consumption.

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

Peripheral clock gating is controlled by the [AHB peripheral clock enable register \(RCC\\_AHBENR\)](#), the [APB peripheral clock enable register 2 \(RCC\\_APB2ENR\)](#) and the [APB peripheral clock enable register 1 \(RCC\\_APB1ENR\)](#).

### 5.3.3 Sleep mode

#### Entering Sleep mode

The Sleep mode is entered by executing the WFI (Wait For Interrupt) or WFE (Wait for Event) instructions. Two options are available to select the Sleep mode entry mechanism, depending on the SLEEPONEXIT bit in the Cortex®-M0 System Control register:

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

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

Refer to [Table 14](#) and [Table 15](#) for details on how to enter Sleep mode.

#### Exiting Sleep mode

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

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

- enabling an interrupt in the peripheral control register but not in the NVIC, and enabling the SEVONPEND bit in the Cortex®-M0 System Control register. When the MCU resumes from WFE, the peripheral interrupt pending bit and the peripheral NVIC IRQ channel pending bit (in the NVIC interrupt clear pending register) must be cleared.
- or configuring an external or internal EXTI line in event mode. When the CPU resumes from WFE, it is not necessary to clear the peripheral interrupt pending bit or the NVIC IRQ channel pending bit as the pending bit corresponding to the event line is not set.

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

Refer to [Table 14](#) and [Table 15](#) for more details on how to exit Sleep mode.

**Table 14. Sleep-now**

Sleep-now mode	Description
<b>Mode entry</b>	WFI (Wait for Interrupt) or WFE (Wait for Event) while: – SLEEPDEEP = 0 and – SLEEPONEXIT = 0 Refer to the Cortex®-M0 System Control register.
<b>Mode exit</b>	If WFI was used for entry: Interrupt: Refer to <a href="#">Table 36: Vector table</a> If WFE was used for entry Wake-up event: Refer to <a href="#">Section 11.2.3: Event management</a>
<b>Wake-up latency</b>	None

**Table 15. Sleep-on-exit**

Sleep-on-exit	Description
<b>Mode entry</b>	WFI (wait for interrupt) while: – SLEEPDEEP = 0 and – SLEEPONEXIT = 1 Refer to the Cortex®-M0 System Control register.
<b>Mode exit</b>	Interrupt: Refer to <a href="#">Table 36: Vector table</a> .
<b>Wake-up latency</b>	None

### 5.3.4 Stop mode

The Stop mode is based on the Cortex®-M0 deep sleep mode combined with peripheral clock gating. The voltage regulator can be configured either in normal or low-power mode. In Stop mode, all clocks in the 1.8 V domain are stopped, the PLL, the HSI and the HSE oscillators are disabled. SRAM and register contents are preserved.

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

#### Entering Stop mode

Refer to [Table 16](#) for details on how to enter the Stop mode.

To further reduce power consumption in Stop mode, the internal voltage regulator can be put in low-power mode. This is configured by the LPDS bit of the [Power control register \(PWR\\_CR\)](#).

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

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

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

- Independent watchdog (IWDG): the IWDG is started by writing to its Key register or by hardware option. Once started it cannot be stopped except by a Reset. See [Section 23.3: IWDG functional description](#).

- real-time clock (RTC): this is configured by the RTCEN bit in the [RTC domain control register \(RCC\\_BDCR\)](#)
- Internal RC oscillator (LSI): this is configured by the LSION bit in the [Control/status register \(RCC\\_CSR\)](#).
- External 32.768 kHz oscillator (LSE): this is configured by the LSEON bit in the [RTC domain control register \(RCC\\_BDCR\)](#).

The ADC or DAC can also consume power during Stop mode, unless they are disabled before entering this mode. Refer to [ADC control register \(ADC\\_CR\)](#) and [DAC control register \(DAC\\_CR\)](#) for details on how to disable them.

### Exiting Stop mode

Refer to [Table 16](#) for more details on how to exit Stop mode.

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

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

**Table 16. Stop mode**

Stop mode	Description
Mode entry	<p>WFI (Wait for Interrupt) or WFE (Wait for Event) while:</p> <ul style="list-style-type: none"> <li>Set SLEEPDEEP bit in Cortex®-M0 System Control register</li> <li>Clear PDDS bit in Power Control register (PWR_CR)</li> <li>Select the voltage regulator mode by configuring LPDS bit in PWR_CR</li> </ul> <p><b>Note:</b> To enter Stop mode, all EXTI line pending bits (in <a href="#">Pending register (EXTI_PR)</a>), all peripherals interrupt pending bits and RTC Alarm flag must be reset. Otherwise, the Stop mode entry procedure is ignored and program execution continues.</p> <p>If the application needs to disable the external oscillator (external clock) before entering Stop mode, the system clock source must be first switched to HSI and then clear the HSEON bit.</p> <p>Otherwise, if before entering Stop mode the HSEON bit is kept at 1, the security system (CSS) feature must be enabled to detect any external oscillator (external clock) failure and avoid a malfunction when entering Stop mode.</p>
Mode exit	<p>If WFI was used for entry:</p> <ul style="list-style-type: none"> <li>Any EXTI line configured in Interrupt mode (the corresponding EXTI Interrupt vector must be enabled in the NVIC).</li> <li>Some specific communication peripherals (CEC, USART, I2C) interrupts, when programmed in wake-up mode (the peripheral must be programmed in wake-up mode and the corresponding interrupt vector must be enabled in the NVIC).</li> </ul> <p>Refer to <a href="#">Table 36: Vector table</a>.</p> <p>If WFE was used for entry:</p> <p>Any EXTI line configured in event mode. Refer to <a href="#">Section 11.2.3: Event management on page 218</a></p>
Wake-up latency	HSI wake-up time + regulator wake-up time from Low-power mode

### 5.3.5 Standby mode

The Standby mode allows to achieve the lowest power consumption. It is based on the Cortex<sup>®</sup>-M0 deepsleep mode, with the voltage regulator disabled. The 1.8 V domain is consequently powered off. The PLL, the HSI oscillator and the HSE oscillator are also switched off. SRAM and register contents are lost except for registers in the RTC domain and Standby circuitry (see [Figure 6](#)).

**Caution:** The Standby mode is not available on STM32F0x8 devices.

#### Entering Standby mode

Refer to [Table 17](#) for more details on how to enter Standby mode.

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

- Independent watchdog (IWDG): the IWDG is started by writing to its Key register or by hardware option. Once started it cannot be stopped except by a reset. See [Section 23.3: IWDG functional description](#).
- Real-time clock (RTC): this is configured by the RTCEN bit in the [RTC domain control register \(RCC\\_BDCR\)](#).
- Internal RC oscillator (LSI): this is configured by the LSION bit in the [Control/status register \(RCC\\_CSR\)](#).
- External 32.768 kHz oscillator (LSE): this is configured by the LSEON bit in the [RTC domain control register \(RCC\\_BDCR\)](#).

#### Exiting Standby mode

The microcontroller exits the Standby mode when an external reset (NRST pin), an IWDG reset, a rising edge on one of the enabled WKUPx pins or an RTC event occurs. All registers are reset after wake-up from Standby except for [Power control/status register \(PWR\\_CSR\)](#).

After waking up from Standby mode, program execution restarts in the same way as after a Reset (boot pin sampling, option bytes loading, reset vector is fetched, etc.). The SBF status flag in the [Power control/status register \(PWR\\_CSR\)](#) indicates that the MCU was in Standby mode.

Refer to [Table 17](#) for more details on how to exit Standby mode.

**Table 17. Standby mode**

Standby mode	Description
<b>Mode entry</b>	WFI (Wait for Interrupt) or WFE (Wait for Event) while: – Set SLEEPDEEP in Cortex <sup>®</sup> -M0 System Control register – Set PDDS bit in Power Control register (PWR_CR) – Clear WUF bit in Power Control/Status register (PWR_CSR)
<b>Mode exit</b>	WKUP pin rising edge, RTC alarm event's rising edge, external Reset in NRST pin, IWDG Reset.
<b>Wake-up latency</b>	Reset phase

### I/O states in Standby mode

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

- Reset pad (still available)
- PC13, PC14 and PC15 if configured by RTC or LSE
- WKUPx pins

### Debug mode

By default, the debug connection is lost if the application puts the MCU in Stop or Standby mode while the debug features are used. This is due to the fact that the Cortex®-M0 core is no longer clocked.

However, by setting some configuration bits in the DBGMCU\_CR register, the software can be debugged even when using the low-power modes extensively.

## 5.3.6 Auto-wake-up from low-power mode

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

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

To wake-up from Stop mode with an RTC alarm event, it is necessary to:

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

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

## 5.4 Power control registers

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

### 5.4.1 Power control register (PWR\_CR)

Address offset: 0x00

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	Res	Res	Res	Res	Res	Res	Res	Res							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	Res	Res	Res	Res	Res	DBP	PLS[2:0]			PVDE	CSBF	CWUF	PDDS	LPDS	
						rw	rw	rw	rw	rw	rc_w1	rc_w1	rw	rw	

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

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

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

0: Access to RTC and Backup registers disabled

1: Access to RTC and Backup registers enabled

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

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

Once the PVD\_LOCK is enabled in the [SYSCFG configuration register 2 \(SYSCFG\\_CFGR2\)](#), the PLS[2:0] bits cannot be programmed anymore.

000: PVD threshold 0

001: PVD threshold 1

010: PVD threshold 2

011: PVD threshold 3

100: PVD threshold 4

101: PVD threshold 5

110: PVD threshold 6

111: PVD threshold 7

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

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

This bit is set and cleared by software. Once the PVD\_LOCK is enabled in the [SYSCFG configuration register 2 \(SYSCFG\\_CFGR2\)](#) register, the PVDE bit cannot be programmed anymore.

0: PVD disabled

1: PVD enabled

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

This bit is always read as 0.

0: No effect

1: Clear the SBF standby flag (write).

Bit 2 **CWUF**: Clear wake-up flag.

This bit is always read as 0.

0: No effect

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

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

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

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

1: Enter Standby mode when the CPU enters Deepsleep.

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

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

0: Voltage regulator on during Stop mode

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

*Note:* When a peripheral that can work in STOP mode requires a clock, the Power controller automatically switch the voltage regulator from Low-power mode to Normal mode and remains in this mode until the request disappears.

## 5.4.2 Power control/status register (PWR\_CSR)

Address offset: 0x04

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

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	Res	Res	Res	Res	Res	Res	Res	Res							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EWUP 8	EWUP 7	EWUP 6	EWUP 5	EWUP 4	EWUP 3	EWUP 2	EWUP 1	Res	Res	Res	Res	VREF INT RDY	PVDO	SBF	WUF
rw					r	r	r	r							

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

Bits 15:8 **EWUPx**: Enable WKUPx pin

These bits are set and cleared by software.

0: WKUPx pin is used for general purpose I/O. An event on the WKUPx pin does not wake-up the device from Standby mode.

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

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

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

**Bit 3 VREFINTRDY:** VREFINT reference voltage ready

This bit is set and cleared by hardware to indicate the state of the internal voltage reference VREFINT.

- 0: VREFINT is not ready
- 1: VREFINT is ready

*Note: This flag is useful only for STM32F0x8 devices where POR is provided externally (through the NPOR pin). In STM32F0x1/F0x2 devices, the internal POR waits for VREFINT to stabilize before releasing the reset.*

**Bit 2 PVDO:** PVD output

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

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

Notes:

1. The PVD is stopped by Standby mode. For this reason, this bit is equal to 0 after Standby or reset until the PVDE bit is set.
2. Once the PVD is enabled and configured in the PWR\_CR register, PVDO can be used to generate an interrupt through the external interrupt controller.

**Bit 1 SBF:** Standby flag

This bit is set by hardware when the device enters Standby mode and it is cleared only by a POR/PDR (power on reset/power down reset) or by setting the CSBF bit in the *Power control register (PWR\_CR)*

- 0: Device has not been in Standby mode
- 1: Device has been in Standby mode

**Bit 0 WUF:** Wake-up flag

This bit is set by hardware to indicate that the device received a wake-up event. It is cleared by a system reset or by setting the CWUF bit in the *Power control register (PWR\_CR)*

- 0: No wake-up event occurred
- 1: A wake-up event was received from one of the enabled WKUPx pins or from the RTC alarm.

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

### 5.4.3 PWR register map

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

**Table 18. PWR register map and reset values**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x000	PWR_CR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DBP	PLS[2:0]	PVDE	CSBF	CWUF	PDOS	LPDS																			
	Reset value																							0	0	0	0	0	0	X	0	0		
0x004	PWR_CSR	Res.	EWUP8	EWUP7	EWUP6	EWUP5	EWUP4	EWUP3	EWUP2	EWUP1	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.														
	Reset value																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 6 Reset and clock control (RCC)

### 6.1 Reset

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

#### 6.1.1 Power reset

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

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

A power reset sets all registers to their reset values except the RTC domain ([Figure 6: Power supply overview](#)).

In STM32F0x8 devices, the POR/PDR reset is not functional and the Standby mode is not available. Power reset must be provided from an external NPOR pin (active low and released by the application when all supply voltages are stabilized).

#### 6.1.2 System reset

A system reset sets all registers to their reset values except the reset flags in the clock controller CSR register and the registers in the RTC domain (see [Figure 6: Power supply overview](#)).

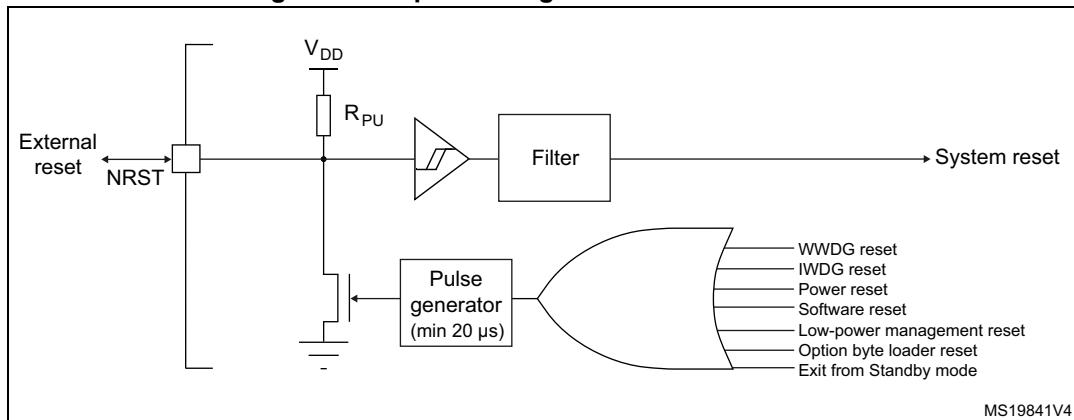
A system reset is generated when one of the following events occurs:

1. A low level on the NRST pin (external reset)
2. Window watchdog event (WWDG reset)
3. Independent watchdog event (IWDG reset)
4. A software reset (SW reset) (see [Software reset](#))
5. Low-power management reset (see [Low-power management reset](#))
6. Option byte loader reset (see [Option byte loader reset](#))
7. A power reset

The reset source can be identified by checking the reset flags in the Control/Status register, RCC\_CSR (see [Section 6.4.10: Control/status register \(RCC\\_CSR\)](#)).

These sources act on the NRST pin and it is always kept low during the delay phase. The RESET service routine vector is fixed at address 0x0000\_0004 in the memory map.

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

**Figure 9. Simplified diagram of the reset circuit**

### Software reset

The SYSRESETREQ bit in Cortex<sup>®</sup>-M0 Application Interrupt and Reset Control Register must be set to force a software reset on the device. Refer to the *Cortex™-M0 technical reference manual* for more details.

### Low-power management reset

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

1. Reset generated when entering Standby mode:

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

2. Reset when entering Stop mode:

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

For further information on the User Option Bytes, refer to [Section 4: Option bytes](#).

### Option byte loader reset

The option byte loader reset is generated when the OBL\_LAUNCH bit (bit 13) is set in the FLASH\_CR register. This bit is used to launch the option byte loading by software.

## 6.1.3 RTC domain reset

The RTC domain has two specific resets that affect only the RTC domain ([Figure 6: Power supply overview](#)).

An RTC domain reset only affects the LSE oscillator, the RTC, the Backup registers and the RCC [RTC domain control register \(RCC\\_BDCR\)](#). It is generated when one of the following events occurs.

1. Software reset, triggered by setting the BDRST bit in the [RTC domain control register \(RCC\\_BDCR\)](#).
2. V<sub>DD</sub> power-up if V<sub>BAT</sub> has been disconnected when it was low.

The Backup registers are also reset when one of the following events occurs:

1. RTC tamper detection event.
2. Change of the read out protection from level 1 to level 0.

## 6.2 Clocks

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

- HSI 8 MHz RC oscillator clock
- HSE oscillator clock
- PLL clock
- HSI48 48 MHz RC oscillator clock (available on STM32F04x, STM32F07x and STM32F09x devices only)

The devices have the following additional clock sources:

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

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

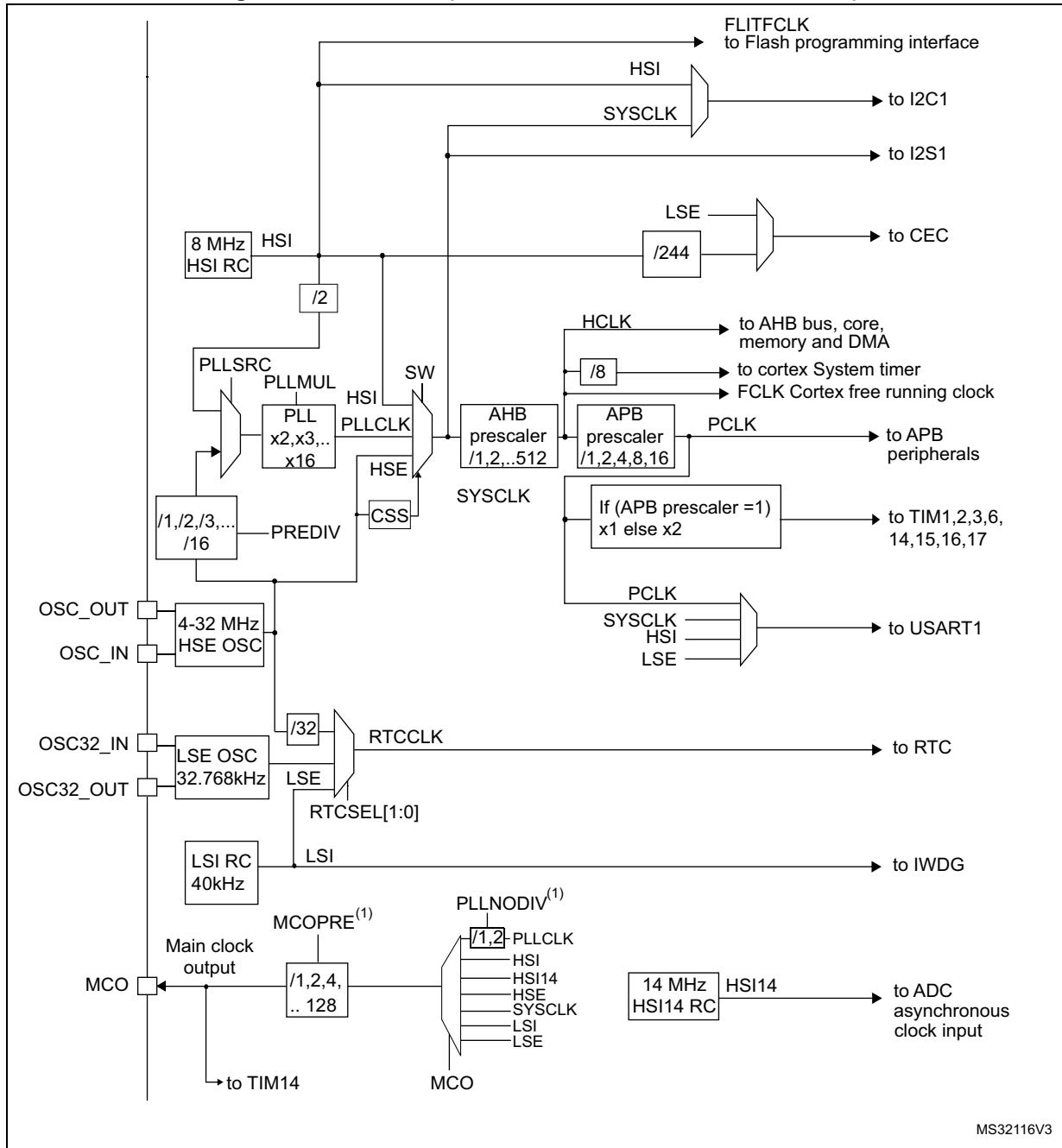
Several prescalers can be used to configure the frequency of the AHB and the APB domains. The AHB and the APB domains maximum frequency is 48 MHz.

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

- The Flash memory programming interface clock (FLITFCLK) which is always the HSI clock.
- The option byte loader clock which is always the HSI clock
- The ADC clock which is derived (selected by software) from one of the two following sources:
  - dedicated HSI14 clock, to run always at the maximum sampling rate
  - APB clock (PCLK) divided by 2 or 4
- The USART1 clock, USART2 clock (on STM32F07x and STM32F09x devices only) and USART3 clock (on STM32F09x devices only) which is derived (selected by software) from one of the four following sources:
  - system clock
  - HSI clock
  - LSE clock
  - APB clock (PCLK)
- The I2C1 clock which is derived (selected by software) from one of the two following sources:
  - system clock
  - HSI clock
- The USB clock which is derived (selected by software) from one of the two following sources:
  - PLL clock
  - HSI48 clock
- The CEC clock which is derived from the HSI clock divided by 244 or from the LSE clock.
- The I2S1 and I2S2 clock which is always the system clock.
- The RTC clock which is derived from the LSE, LSI or from the HSE clock divided by 32.
- The timer clock frequencies are automatically fixed by hardware. There are two cases:
  - if the APB prescaler is 1, the timer clock frequencies are set to the same frequency as that of the APB domain;
  - otherwise, they are set to twice (x2) the frequency of the APB domain.
- The IWDG clock which is always the LSI clock.

The RCC feeds the Cortex System Timer (SysTick) external clock with the AHB clock (HCLK) divided by 8. The SysTick can work either with this clock or directly with the Cortex clock (HCLK), configurable in the SysTick Control and Status Register.

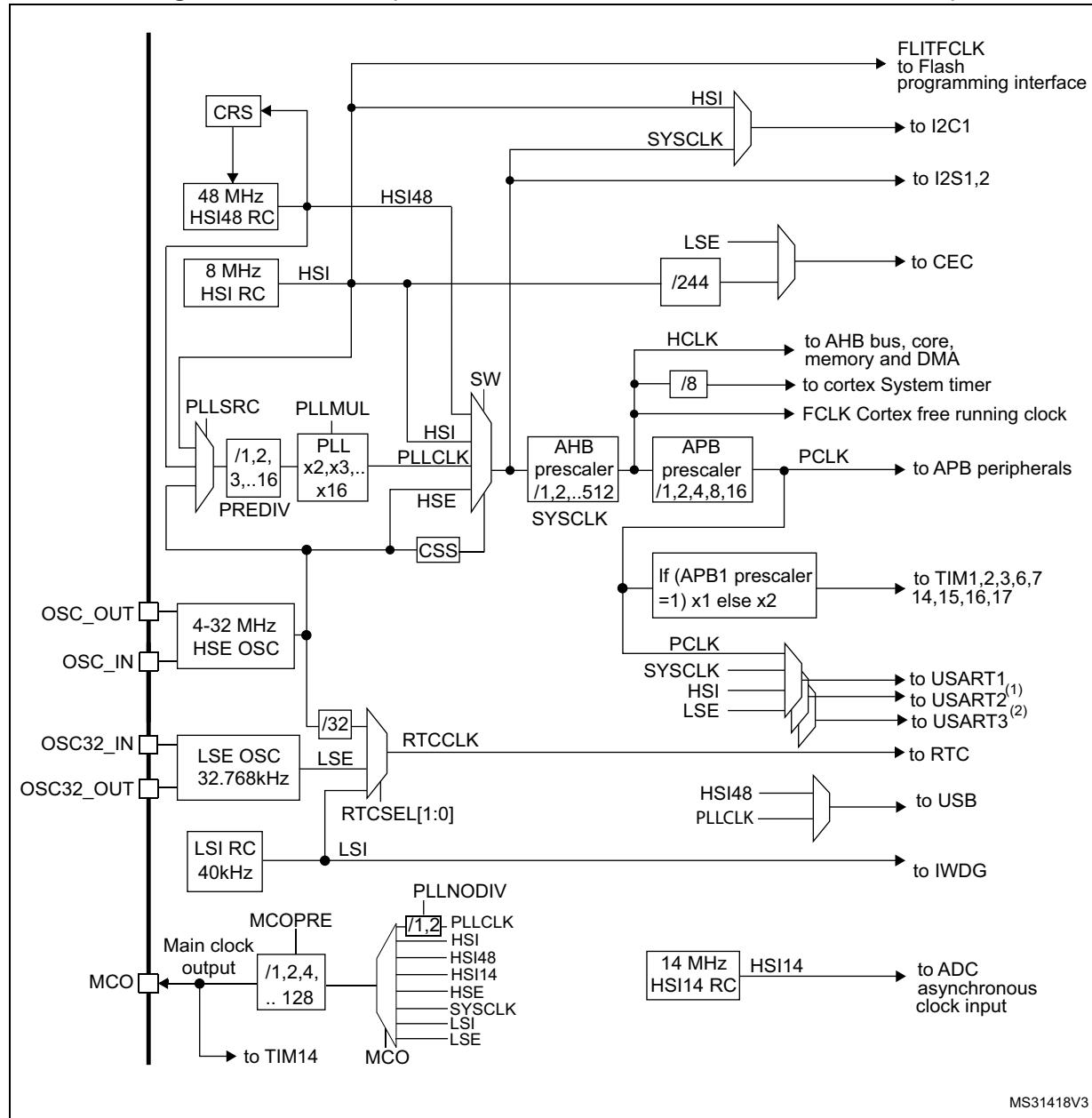
Figure 10. Clock tree (STM32F03x and STM32F05x devices)



1. Not available on STM32F05x devices.

MS32116V3

Figure 11. Clock tree (STM32F04x, STM32F07x and STM32F09x devices)



MS31418V3

1. Not available on STM32F04x devices.
2. Not available on STM32F04x and STM32F07x devices

FCLK acts as Cortex®-M0's free-running clock. For more details refer to the *Arm Cortex-M0 r0p0 technical reference manual (TRM)*.

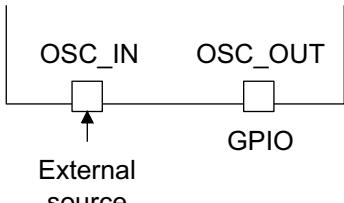
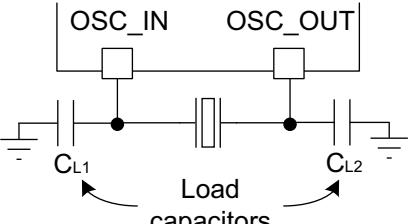
### 6.2.1 HSE clock

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

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

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

**Figure 12. HSE/ LSE clock sources**

Clock source	Hardware configuration
External clock	 <p>External source</p> <p>OSC_IN      OSC_OUT</p> <p>GPIO</p> <p>MSv31915V1</p>
Crystal/Ceramic resonators	 <p>OSC_IN      OSC_OUT</p> <p>Load capacitors</p> <p>CL1      CL2</p> <p>MSv31916V1</p>

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

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

The associated hardware configuration is shown in [Figure 12](#). Refer to the electrical characteristics section of the *datasheet* for more details.

The HSERDY flag in the [Clock control register \(RCC\\_CR\)](#) indicates if the HSE oscillator is stable or not. At startup, the clock is not released until this bit is set by hardware. An interrupt can be generated if enabled in the [Clock interrupt register \(RCC\\_CIR\)](#).

The HSE Crystal can be switched on and off using the HSEON bit in the [Clock control register \(RCC\\_CR\)](#).

For code example refer to the Appendix section [A.3.1: HSE start sequence code example](#).

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

### External source (HSE bypass)

In this mode, an external clock source must be provided. It can have a frequency of up to 32 MHz. You select this mode by setting the HSEBYP and HSEON bits in the [Clock control register \(RCC\\_CR\)](#). The external clock signal (square, sinus or triangle) with ~40-60% duty cycle depending on the frequency (refer to the *datasheet*) has to drive the OSC\_IN pin while the OSC\_OUT pin can be used a GPIO. See [Figure 12](#).

## 6.2.2 HSI clock

The HSI clock signal is generated from an internal 8 MHz RC oscillator and can be used directly as a system clock or for PLL input

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

### Calibration

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

After reset, the factory calibration value is loaded in the HSICAL[7:0] bits in the [Clock control register \(RCC\\_CR\)](#).

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

For more details on how to measure the HSI frequency variation refer to [Section 6.2.13: Internal/external clock measurement with TIM14 on page 107](#).

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

The HSI RC can be switched on and off using the HSION bit in the [Clock control register \(RCC\\_CR\)](#).

The HSI signal can also be used as a backup source (Auxiliary clock) if the HSE crystal oscillator fails. Refer to [Section 6.2.8: Clock security system \(CSS\) on page 105](#).

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

### 6.2.3 HSI48 clock

On STM32F04x, STM32F07x and STM32F09x devices only, the HSI48 clock signal is generated from an internal 48 MHz RC oscillator and can be used directly as a system clock or divided and be used as PLL input.

The internal 48MHz RC oscillator is mainly dedicated to provide a high precision clock to the USB peripheral by means of a special Clock recovery system (CRS) circuitry, which could use the USB SOF signal or the LSE or an external signal to automatically adjust the oscillator frequency on-fly, in a very small steps. This oscillator can also be used as a system clock source when the system is in run mode; it is disabled as soon as the system enters in Stop or Standby mode. When the CRS is not used, the HSI48 RC oscillator runs on its default frequency which is subject to manufacturing process variations, this is why each device is factory calibrated by ST for ~3% accuracy at  $T_A = 25^\circ\text{C}$ .

For more details on how to configure and use the CRS peripheral refer to [Section 7](#).

The HSI48RDY flag in the [Clock control register \(RCC\\_CR\)](#) indicates if the HSI48 RC is stable or not. At startup, the HSI48 RC output clock is not released until this bit is set by hardware.

The HSI48 RC can be switched on and off using the HSI48ON bit in the [Clock control register \(RCC\\_CR\)](#). This oscillator is also automatically enabled (by hardware forcing HSI48ON bit to one) as soon as it is chosen as a clock source for the USB and the peripheral is enabled.

Furthermore it is possible to drive the HSI48 clock to the MCO multiplexer and use it as a clock source for other application components.

### 6.2.4 PLL

The internal PLL can be used to multiply the HSI, a divided HSI48 or the HSE output clock frequency. Refer to [Figure 9: Simplified diagram of the reset circuit](#), [Figure 12: HSE/LSE clock sources](#) and [Clock control register \(RCC\\_CR\)](#).

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

To modify the PLL configuration, proceed as follows:

1. Disable the PLL by setting PLLON to 0.
2. Wait until PLLRDY is cleared. The PLL is now fully stopped.
3. Change the desired parameter.
4. Enable the PLL again by setting PLLON to 1.
5. Wait until PLLRDY is set.

An interrupt can be generated when the PLL is ready, if enabled in the [Clock interrupt register \(RCC\\_CIR\)](#).

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

For code example refer to the Appendix section [A.3.2: PLL configuration modification code example](#).

### 6.2.5 LSE clock

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

The LSE crystal is switched on and off using the LSEON bit in [RTC domain control register \(RCC\\_BDCR\)](#). The crystal oscillator driving strength can be changed at runtime using the LSEDRV[1:0] bits in the [RTC domain control register \(RCC\\_BDCR\)](#) to obtain the best compromise between robustness and short start-up time on one side and low-power consumption on the other.

The LSERDY flag in the [RTC domain control register \(RCC\\_BDCR\)](#) indicates whether the LSE crystal is stable or not. At startup, the LSE crystal output clock signal is not released until this bit is set by hardware. An interrupt can be generated if enabled in the [Clock interrupt register \(RCC\\_CIR\)](#).

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

#### External source (LSE bypass)

In this mode, an external clock source must be provided. It can have a frequency of up to 1 MHz. You select this mode by setting the LSEBYP and LSEON bits in the [RTC domain control register \(RCC\\_BDCR\)](#). The external clock signal (square, sinus or triangle) with ~50% duty cycle has to drive the OSC32\_IN pin while the OSC32\_OUT pin can be used as GPIO. See [Figure 12](#).

### 6.2.6 LSI clock

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

The LSI RC can be switched on and off using the LSION bit in the [Control/status register \(RCC\\_CSR\)](#).

The LSIRDY flag in the [Control/status register \(RCC\\_CSR\)](#) indicates if the LSI oscillator is stable or not. At startup, the clock is not released until this bit is set by hardware. An interrupt can be generated if enabled in the [Clock interrupt register \(RCC\\_CIR\)](#).

## 6.2.7 System clock (SYSCLK) selection

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

- HSI oscillator
- HSE oscillator
- PLL
- HSI48 oscillator (available only on STM32F04x, STM32F07x and STM32F09x devices)

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

A switch from one clock source to another occurs only if the target clock source is ready (clock stable after startup delay or PLL locked). If a clock source which is not yet ready is selected, the switch will occur when the clock source becomes ready. Status bits in the [Clock control register \(RCC\\_CR\)](#) indicate which clock(s) is (are) ready and which clock is currently used as a system clock.

## 6.2.8 Clock security system (CSS)

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

If a failure is detected on the HSE clock, the HSE oscillator is automatically disabled, a clock failure event is sent to the break input of the advanced-control timers (TIM1) and general-purpose timers (TIM15, TIM16 and TIM17) and an interrupt is generated to inform the software about the failure (clock security system interrupt, or CSSI), allowing the MCU to perform rescue operations. The CSSI is linked to the Cortex®-M0 NMI (Non-Maskable Interrupt) exception vector.

Note:

*Once the CSS is enabled and if the HSE clock fails, the CSS interrupt occurs and an NMI is automatically generated. The NMI is executed indefinitely unless the CSS interrupt pending bit is cleared. As a consequence, in the NMI ISR user must clear the CSS interrupt by setting the CSSC bit in the [Clock interrupt register \(RCC\\_CIR\)](#).*

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

## 6.2.9 ADC clock

The ADC clock selection is done inside the ADC\_CFGR2 (refer to [Section 13.11.5: ADC configuration register 2 \(ADC\\_CFGR2\) on page 271](#)). It can be either the dedicated 14 MHz RC oscillator (HSI14) connected on the ADC asynchronous clock input or PCLK divided by 2 or 4. The 14 MHz RC oscillator can be configured by software either to be turned on/off (“auto-off mode”) by the ADC interface or to be always enabled. The HSI 14 MHz RC

oscillator cannot be turned on by ADC interface when the APB clock is selected as an ADC kernel clock.

### 6.2.10 RTC clock

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

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

- If LSE is selected as RTC clock:
  - The RTC continues to work even if the  $V_{DD}$  supply is switched off, provided the  $V_{BAT}$  supply is maintained.
  - The RTC remains clocked and functional under system reset
- If LSI is selected as the RTC clock:
  - The RTC state is not guaranteed if the  $V_{DD}$  supply is powered off. Refer to [Section 6.2.6: LSI clock on page 104](#) for more details on LSI calibration.
- If the HSE clock divided by 32 is used as the RTC clock:
  - The RTC state is not guaranteed if the  $V_{DD}$  supply is powered off or if the internal voltage regulator is powered off (removing power from the 1.8 V domain).

When the RTC clock is LSE, the RTC remains clocked and functional under system reset.

### 6.2.11 Independent watchdog clock

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

### 6.2.12 Clock-out capability

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

- HSI14
- SYSCLK
- HSI
- HSE
- PLL clock divided by 2 or direct (direct connection is not available on STM32F05x devices)
- LSE
- LSI
- HSI48 (on STM32F04x, STM32F07x and STM32F09x devices only)

The selection is controlled by the MCO[3:0] bits of the *Clock configuration register (RCC\_CFGR)*.

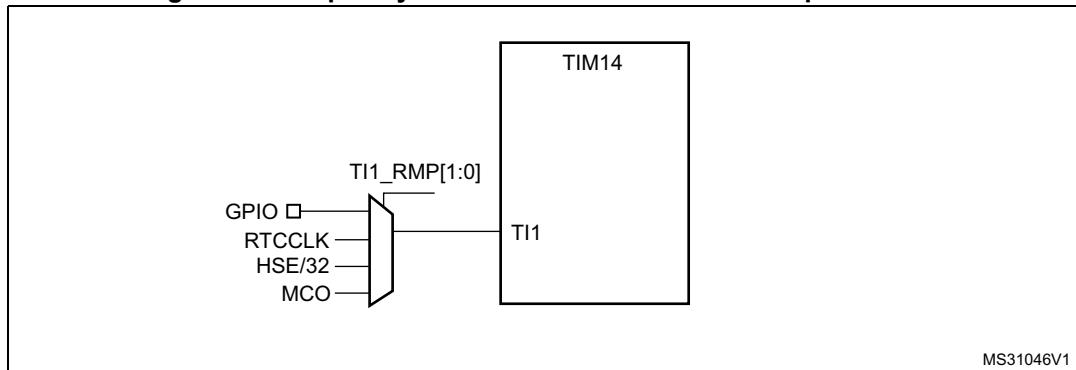
For code example refer to the Appendix section [A.3.3: MCO selection code example](#).

On STM32F03x, STM32F04x, STM32F07x and STM32F09x devices, the additional bit PLLNODIV of this register controls the divider bypass for a PLL clock input to MCO. The MCO frequency can be reduced by a configurable binary divider, controlled by the MCOPRE[2..0] bits of the [Clock configuration register \(RCC\\_CFGR\)](#).

### 6.2.13 Internal/external clock measurement with TIM14

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

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



The input capture channel of the Timer 14 can be a GPIO line or an internal clock of the MCU. This selection is performed through the TI1\_RMP [1:0] bits in the TIM14\_OR register. The possibilities available are the following ones.

- TIM14 Channel1 is connected to the GPIO. Refer to the alternate function mapping in the device datasheets.
- TIM14 Channel1 is connected to the RTCCLK.
- TIM14 Channel1 is connected to the HSE/32 Clock.
- TIM14 Channel1 is connected to the microcontroller clock output (MCO). Refer to [Section 6.2.12: Clock-out capability](#) for MCO clock configuration.

For code example refer to the Appendix section [A.3.4: Clock measurement configuration with TIM14 code example](#).

### Calibration of the HSI

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

The HSI oscillator has dedicated user-accessible calibration bits for this purpose.

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

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

### Calibration of the LSI

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

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

### Calibration of the HSI14

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

## 6.3 Low-power modes

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

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

Stop mode stops all the clocks in the core supply domain and disables the PLL and the HSI, HSI48, HSI14 and HSE oscillators.

HDMI CEC, USART1, USART2 (only on STM32F07x and STM32F09x devices), USART3 (only on STM32F09x devices) and I2C1 have the capability to enable the HSI oscillator even when the MCU is in Stop mode (if HSI is selected as the clock source for that peripheral). When the system is in Stop mode, with the regulator in LP mode, the clock request coming from any of those three peripherals moves the regulator to MR mode in order to have the proper current drive capability for the core logic. The regulator moves back to LP mode once this request is removed without waking up the MCU.

HDMI CEC, USART1, USART2 (only on STM32F07x and STM32F09x devices) and USART3 (only on STM32F09x devices) can also be driven by the LSE oscillator when the system is in Stop mode (if LSE is selected as clock source for that peripheral) and the LSE oscillator is enabled (LSEON) but they do not have the capability to turn on the LSE oscillator.

Standby mode stops all the clocks in the core supply domain and disables the PLL and the HSI, HSI48, HSI14 and HSE oscillators.

The CPU's deepsleep mode can be overridden for debugging by setting the DBG\_STOP or DBG\_STANDBY bits in the DBGMCU\_CR register.

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

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

## 6.4 RCC registers

Refer to [Section 1.2 on page 42](#) for a list of abbreviations used in register descriptions.

### 6.4.1 Clock control register (RCC\_CR)

Address offset: 0x000

Reset value: 0x0000 XX83 where X is undefined.

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	PLL RDY	POLLON	Res.	Res.	Res.	Res.	CSS ON	HSE BYP	HSE RDY	HSE ON
						r	rw					rw	rw	r	rw
HSICAL[7:0]								HSITRIM[4:0]							
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw		r	rw

Bits 31:26 Reserved, must be kept at reset value.

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

Set by hardware to indicate that the PLL is locked.

0: PLL unlocked

1: PLL locked

Bit 24 **POLLON**: PLL enable

Set and cleared by software to enable PLL.

Cleared by hardware when entering Stop or Standby mode. This bit can not be reset if the PLL clock is used as system clock or is selected to become the system clock.

0: PLL OFF

1: PLL ON

Bits 23:20 Reserved, must be kept at reset value.

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

Set and cleared by software to enable the clock security system. When CSSON is set, the clock detector is enabled by hardware when the HSE oscillator is ready, and disabled by hardware if a HSE clock failure is detected.

0: Clock security system disabled (clock detector OFF).

1: Clock security system enabled (clock detector ON if the HSE is ready, OFF if not).

Bit 18 **HSEBYP**: HSE crystal oscillator bypass

Set and cleared by software to bypass the oscillator with an external clock. The external clock must be enabled with the HSEON bit set, to be used by the device. The HSEBYP bit can be written only if the HSE oscillator is disabled.

0: HSE crystal oscillator not bypassed

1: HSE crystal oscillator bypassed with external clock

Bit 17 **HSERDY**: HSE clock ready flag

Set by hardware to indicate that the HSE oscillator is stable. This bit needs 6 cycles of the HSE oscillator clock to fall down after HSEON reset.

- 0: HSE oscillator not ready
- 1: HSE oscillator ready

Bit 16 **HSEON**: HSE clock enable

Set and cleared by software.

Cleared by hardware to stop the HSE oscillator when entering Stop or Standby mode. This bit cannot be reset if the HSE oscillator is used directly or indirectly as the system clock.

- 0: HSE oscillator OFF
- 1: HSE oscillator ON

Bits 15:8 **HSICAL[7:0]**: HSI clock calibration

These bits are initialized automatically at startup. They are adjusted by SW through the HSITRIM setting.

Bits 7:3 **HSITRIM[4:0]**: HSI clock trimming

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

The default value is 16, which, when added to the HSICAL value, should trim the HSI to 8 MHz  $\pm 1\%$ . The trimming step is around 40 kHz between two consecutive HSICAL steps.

*Note: Increased value in the register results to higher clock frequency.*

## Bit 2 Reserved, must be kept at reset value.

Bit 1 **HSIRDY**: HSI clock ready flag

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

- 0: HSI oscillator not ready
- 1: HSI oscillator ready

Bit 0 **HSION**: HSI clock enable

Set and cleared by software.

Set by hardware to force the HSI oscillator ON when leaving Stop or Standby mode or in case of failure of the HSE crystal oscillator used directly or indirectly as system clock. This bit cannot be reset if the HSI is used directly or indirectly as system clock or is selected to become the system clock.

- 0: HSI oscillator OFF
- 1: HSI oscillator ON

## 6.4.2 Clock configuration register (RCC\_CFGR)

Address offset: 0x04

Reset value: 0x0000 0000

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

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

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PLL NODIV		MCOPRE[2:0]			MCO[3:0]				Res.	Res.	PLLMUL[3:0]				PLL XTPRE	PLL SRC[1]
rw	rw	rw	rw	rw	rw	rw	rw				rw	rw	rw	rw	rw	rw
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PLL SRC[0]	ADC PRE	Res.	Res.	Res.	PPRE[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]		
rw	rw				rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw	

Bit 31 **PLLNODIV:** PLL clock not divided for MCO (not available on STM32F05x devices)

This bit is set and cleared by software. It switches off divider by 2 for PLL connection to MCO.

- 0: PLL is divided by 2 for MCO
- 1: PLL is not divided for MCO

Bits 30:28 **MCOPRE[2:0]:** Microcontroller clock output prescaler (not available on STM32F05x devices)

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

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

Bits 27:24 **MCO[3:0]:** Microcontroller clock output

Set and cleared by software.

- 0000: MCO output disabled, no clock on MCO
- 0001: Internal RC 14 MHz (HSI14) oscillator clock selected
- 0010: Internal low speed (LSI) oscillator clock selected
- 0011: External low speed (LSE) oscillator clock selected
- 0100: System clock selected
- 0101: Internal RC 8 MHz (HSI) oscillator clock selected
- 0110: External 4-32 MHz (HSE) oscillator clock selected
- 0111: PLL clock selected (divided by 1 or 2, depending on PLLNODIV)
- 1000: Internal RC 48 MHz (HSI48) oscillator clock selected

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

Bits 23:22 Reserved, must be kept at reset value.

Bits 21:18 **PLLMUL[3:0]**: PLL multiplication factor

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

Caution: The PLL output frequency must not exceed 48 MHz.

- 0000: PLL input clock x 2
- 0001: PLL input clock x 3
- 0010: PLL input clock x 4
- 0011: PLL input clock x 5
- 0100: PLL input clock x 6
- 0101: PLL input clock x 7
- 0110: PLL input clock x 8
- 0111: PLL input clock x 9
- 1000: PLL input clock x 10
- 1001: PLL input clock x 11
- 1010: PLL input clock x 12
- 1011: PLL input clock x 13
- 1100: PLL input clock x 14
- 1101: PLL input clock x 15
- 1110: PLL input clock x 16
- 1111: PLL input clock x 16

Bit 17 **PLLXTPRE**: HSE divider for PLL input clock

This bit is the same bit as bit PREDIV[0] from RCC\_CFGR2. Refer to RCC\_CFGR2 PREDIV bits description for its meaning.

Bits 16:15 **PLLSRC[1:0]**: PLL input clock source

Set and cleared by software to select PLL or PREDIV clock source. These bits can be written only when PLL is disabled.

- 00: HSI/2 selected as PLL input clock (PREDIV forced to divide by 2 on STM32F04x, STM32F07x and STM32F09x devices)
- 01: HSI/PREDIV selected as PLL input clock
- 10: HSE/PREDIV selected as PLL input clock
- 11: HSI48/PREDIV selected as PLL input clock

Bit PLLSRC[0] is available only on STM32F04x, STM32F07x and STM32F09x devices, otherwise it is reserved (with value zero).

Bit 14 **ADCPRE**: ADC prescaler

Obsolete setting. Proper ADC clock selection is done inside the ADC\_CFGR2 (refer to [Section 13.11.5: ADC configuration register 2 \(ADC\\_CFGR2\) on page 271](#)).

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

Bits 10:8 **PPRE[2:0]**: PCLK prescaler

Set and cleared by software to control the division factor of the APB clock (PCLK).

- 0xx: HCLK not divided
- 100: HCLK divided by 2
- 101: HCLK divided by 4
- 110: HCLK divided by 8
- 111: HCLK divided by 16

**Bits 7:4 HPRE[3:0]: HCLK prescaler**

Set and cleared by software to control the division factor of the AHB clock.

- 0xxx: SYSCLK not divided
- 1000: SYSCLK divided by 2
- 1001: SYSCLK divided by 4
- 1010: SYSCLK divided by 8
- 1011: SYSCLK divided by 16
- 1100: SYSCLK divided by 64
- 1101: SYSCLK divided by 128
- 1110: SYSCLK divided by 256
- 1111: SYSCLK divided by 512

**Bits 3:2 SWS[1:0]: System clock switch status**

Set and cleared by hardware to indicate which clock source is used as system clock.

- 00: HSI oscillator used as system clock
- 01: HSE oscillator used as system clock
- 10: PLL used as system clock
- 11: HSI48 oscillator used as system clock (when available)

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

Set and cleared by software to select SYSCLK source.

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

- 00: HSI selected as system clock
- 01: HSE selected as system clock
- 10: PLL selected as system clock
- 11: HSI48 selected as system clock (when available)

### 6.4.3 Clock interrupt register (RCC\_CIR)

Address offset: 0x08

Reset value: 0x0000 0000

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CSSC	HSI48 RDYC	HSI14 RDYC	PLL RDYC	HSE RDYC	HSI RDYC	LSE RDYC	LSI RDYC
								w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	HSI48 RDYIE	HSI14 RDYIE	PLL RDYIE	HSE RDYIE	HSI RDYIE	LSE RDYIE	LSI RDYIE	CSSF	HSI48 RDYF	HSI14 RDYF	PLL RDYF	HSE RDYF	HSI RDYF	LSE RDYF	LSI RDYF
	rw	rw	rw	rw	rw	rw	rw	r	r	r	r	r	r	r	r

Bits 31:24 Reserved, must be kept at reset value.

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

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

0: No effect

1: Clear CSSF flag

Bit 22 **HSI48RDYC:** HSI48 Ready Interrupt Clear

This bit is set by software to clear the HSI48RDYF flag.

0: No effect

1: Clear HSI48RDYF flag

Bit 21 **HSI14RDYC:** HSI14 ready interrupt clear

This bit is set by software to clear the HSI14RDYF flag.

0: No effect

1: Clear HSI14RDYF flag

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

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

0: No effect

1: Clear PLLRDYF flag

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

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

0: No effect

1: Clear HSERDYF flag

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

This bit is set software to clear the HSIRDYF flag.

0: No effect

1: Clear HSIRDYF flag

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

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

0: No effect

1: LSERDYF cleared

Bit 16 **LSIRDYC:** LSI ready interrupt clear

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

0: No effect

1: LSIRDYF cleared

## Bit 15 Reserved, must be kept at reset value.

Bit 14 **HSI48RDYIE:** HSI48 ready interrupt enable

Set and cleared by software to enable/disable interrupt caused by the HSI48 oscillator stabilization.

0: HSI48 ready interrupt disabled

1: HSI48 ready interrupt enabled

Bit 13 **HSI14RDYIE:** HSI14 ready interrupt enable

Set and cleared by software to enable/disable interrupt caused by the HSI14 oscillator stabilization.

0: HSI14 ready interrupt disabled

1: HSI14 ready interrupt enabled

Bit 12 **PLLRDYIE:** PLL ready interrupt enable

Set and cleared by software to enable/disable interrupt caused by PLL lock.

0: PLL lock interrupt disabled

1: PLL lock interrupt enabled

- Bit 11 **HSERDYIE:** HSE ready interrupt enable  
Set and cleared by software to enable/disable interrupt caused by the HSE oscillator stabilization.  
0: HSE ready interrupt disabled  
1: HSE ready interrupt enabled
- Bit 10 **HSIRDYIE:** HSI ready interrupt enable  
Set and cleared by software to enable/disable interrupt caused by the HSI oscillator stabilization.  
0: HSI ready interrupt disabled  
1: HSI ready interrupt enabled
- Bit 9 **LSERDYIE:** LSE ready interrupt enable  
Set and cleared by software to enable/disable interrupt caused by the LSE oscillator stabilization.  
0: LSE ready interrupt disabled  
1: LSE ready interrupt enabled
- Bit 8 **LSIRDYIE:** LSI ready interrupt enable  
Set and cleared by software to enable/disable interrupt caused by the LSI oscillator stabilization.  
0: LSI ready interrupt disabled  
1: LSI ready interrupt enabled
- Bit 7 **CSSF:** Clock security system interrupt flag  
Set by hardware when a failure is detected in the HSE oscillator.  
Cleared by software setting the CSSC bit.  
0: No clock security interrupt caused by HSE clock failure  
1: Clock security interrupt caused by HSE clock failure
- Bit 6 **HSI48RDYF:** HSI48 ready interrupt flag  
Set by hardware when the HSI48 becomes stable and HSI48RDYDIE is set in a response to setting the HSI48ON bit in Clock control register 2 (RCC\_CR2). When HSI48ON is not set but the HSI48 oscillator is enabled by the peripheral through a clock request, this bit is not set and no interrupt is generated.  
Cleared by software setting the HSI48RDYC bit.  
0: No clock ready interrupt caused by the HSI48 oscillator  
1: Clock ready interrupt caused by the HSI48 oscillator
- Bit 5 **HSI14RDYF:** HSI14 ready interrupt flag  
Set by hardware when the HSI14 becomes stable and HSI14RDYDIE is set in a response to setting the HSI14ON bit in [Clock control register 2 \(RCC\\_CR2\)](#). When HSI14ON is not set but the HSI14 oscillator is enabled by the peripheral through a clock request, this bit is not set and no interrupt is generated.  
Cleared by software setting the HSI14RDYC bit.  
0: No clock ready interrupt caused by the HSI14 oscillator  
1: Clock ready interrupt caused by the HSI14 oscillator
- Bit 4 **PLLRDYF:** PLL ready interrupt flag  
Set by hardware when the PLL locks and PLLRDYDIE is set.  
Cleared by software setting the PLLRDYC bit.  
0: No clock ready interrupt caused by PLL lock  
1: Clock ready interrupt caused by PLL lock

Bit 3 **HSERDYF:** HSE ready interrupt flag

Set by hardware when the HSE clock becomes stable and HSERDYDIE is set.

Cleared by software setting the HSERDYC bit.

0: No clock ready interrupt caused by the HSE oscillator

1: Clock ready interrupt caused by the HSE oscillator

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

Set by hardware when the HSI clock becomes stable and HSIRDYDIE is set in a response to setting the HSION (refer to [Clock control register \(RCC\\_CR\)](#)). When HSION is not set but the HSI oscillator is enabled by the peripheral through a clock request, this bit is not set and no interrupt is generated.

Cleared by software setting the HSIRDYC bit.

0: No clock ready interrupt caused by the HSI oscillator

1: Clock ready interrupt caused by the HSI oscillator

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

Set by hardware when the LSE clock becomes stable and LSERDYDIE is set.

Cleared by software setting the LSERDYC bit.

0: No clock ready interrupt caused by the LSE oscillator

1: Clock ready interrupt caused by the LSE oscillator

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

Set by hardware when the LSI clock becomes stable and LSIRDYDIE is set.

Cleared by software setting the LSIRDYC bit.

0: No clock ready interrupt caused by the LSI oscillator

1: Clock ready interrupt caused by the LSI oscillator

#### 6.4.4 APB peripheral reset register 2 (RCC\_APB2RSTR)

Address offset: 0x0C

Reset value: 0x0000000000

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DBGMCU RST	Res.	Res.	Res.	TIM17 RST	TIM16 RST	TIM15 RST
									rw				rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USART1 RST	Res.	SPI1 RST	TIM1 RST	Res.	ADC RST	Res.	USART8 RST	USART7R ST	USART6 RST	Res.	Res.	Res.	Res.	SYSCFG RST
	rw		rw	rw		rw		rw	rw	rw					rw

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

Bit 22 **DBGMCURST:** Debug MCU reset

Set and cleared by software.

0: No effect

1: Reset Debug MCU

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

- Bit 18 **TIM17RST:** TIM17 timer reset  
Set and cleared by software.  
0: No effect  
1: Reset TIM17 timer
- Bit 17 **TIM16RST:** TIM16 timer reset  
Set and cleared by software.  
0: No effect  
1: Reset TIM16 timer
- Bit 16 **TIM15RST:** TIM15 timer reset  
Set and cleared by software.  
0: No effect  
1: Reset TIM15 timer
- Bit 15 Reserved, must be kept at reset value.
- Bit 14 **USART1RST:** USART1 reset  
Set and cleared by software.  
0: No effect  
1: Reset USART1
- Bit 13 Reserved, must be kept at reset value.
- Bit 12 **SPI1RST:** SPI1 reset  
Set and cleared by software.  
0: No effect  
1: Reset SPI1
- Bit 11 **TIM1RST:** TIM1 timer reset  
Set and cleared by software.  
0: No effect  
1: Reset TIM1 timer
- Bit 10 Reserved, must be kept at reset value.
- Bit 9 **ADCRST:** ADC interface reset  
Set and cleared by software.  
0: No effect  
1: Reset ADC interface
- Bit 8 Reserved, must be kept at reset value.
- Bit 7 **USART8RST:** USART8 reset  
Set and cleared by software  
0: No effect  
1: Reset USART8
- Bit 6 **USART7RST:** USART7 reset  
Set and cleared by software  
0: No effect  
1: Reset USART7
- Bit 5 **USART6RST:** USART6 reset  
Set and cleared by software  
0: No effect  
1: Reset USART6

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

Bit 0 **SYSCFGRST**: SYSCFG reset

Set and cleared by software.

0: No effect

1: Reset SYSCFG

#### 6.4.5 APB peripheral reset register 1 (RCC\_APB1RSTR)

Address offset: 0x10

Reset value: 0x0000 0000

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

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	CEC RST	DAC RST	PWR RST	CRS RST	Res.	CAN RST	Res.	USB RST	I2C2 RST	I2C1 RST	USART5 RST	USART4 RST	USART3 RST	USART2 RST	Res.	
	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw	rw	rw	rw	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SPI2 RST	Res.	Res.	WWDG RST	Res.	Res.	TIM14 RST	Res.	Res.	TIM7 RST	TIM6 RST	Res.	Res.	TIM3 RST	TIM2 RST	
	rw			rw			rw			rw	rw			rw	rw	

Bit 31 Reserved, must be kept at reset value.

Bit 30 **CECRST**: HDMI CEC reset

Set and cleared by software.

0: No effect

1: Reset HDMI CEC

Bit 29 **DACRST**: DAC interface reset

Set and cleared by software.

0: No effect

1: Reset DAC interface

Bit 28 **PWRRST**: Power interface reset

Set and cleared by software.

0: No effect

1: Reset power interface

Bit 27 **CRSRST**: Clock recovery system interface reset

Set and cleared by software.

0: No effect

1: Reset CRS interface

Bit 26 Reserved, must be kept at reset value.

Bit 25 **CANRST**: CAN interface reset

Set and cleared by software.

0: No effect

1: Reset CAN interface

Bit 24 Reserved, must be kept at reset value.

- Bit 23 **USBRST:** USB interface reset  
Set and cleared by software.  
0: No effect  
1: Reset USB interface
- Bit 22 **I2C2RST:** I2C2 reset  
Set and cleared by software.  
0: No effect  
1: Reset I2C2
- Bit 21 **I2C1RST:** I2C1 reset  
Set and cleared by software.  
0: No effect  
1: Reset I2C1
- Bit 20 **USART5RST:** USART5 reset  
Set and cleared by software.  
0: No effect  
1: Reset USART4
- Bit 19 **USART4RST:** USART4 reset  
Set and cleared by software.  
0: No effect  
1: Reset USART4
- Bit 18 **USART3RST:** USART3 reset  
Set and cleared by software.  
0: No effect  
1: Reset USART3
- Bit 17 **USART2RST:** USART2 reset  
Set and cleared by software.  
0: No effect  
1: Reset USART2
- Bits 16:15 Reserved, must be kept at reset value.
- Bit 14 **SPI2RST:** SPI2 reset  
Set and cleared by software.  
0: No effect  
1: Reset SPI2
- Bits 13:12 Reserved, must be kept at reset value.
- Bit 11 **WWDGRST:** Window watchdog reset  
Set and cleared by software.  
0: No effect  
1: Reset window watchdog
- Bits 10:9 Reserved, must be kept at reset value.
- Bit 8 **TIM14RST:** TIM14 timer reset  
Set and cleared by software.  
0: No effect  
1: Reset TIM14
- Bits 7:6 Reserved, must be kept at reset value.

Bit 5 **TIM7RST:** TIM7 timer reset  
Set and cleared by software.  
0: No effect  
1: Reset TIM7

Bit 4 **TIM6RST:** TIM6 timer reset  
Set and cleared by software.  
0: No effect  
1: Reset TIM6

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

Bit 1 **TIM3RST:** TIM3 timer reset  
Set and cleared by software.  
0: No effect  
1: Reset TIM3

Bit 0 **TIM2RST:** TIM2 timer reset  
Set and cleared by software.  
0: No effect  
1: Reset TIM2

#### 6.4.6 AHB peripheral clock enable register (RCC\_AHBENR)

Address offset: 0x14

Reset value: 0x0000 0014

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

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	TSCEN	Res.	IOPF EN	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.						
							rw		rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	CRC EN	Res.	FLITF EN	Res.	SRAM EN	DMA2 EN	DMA EN							
									rw		rw		rw	rw	rw

Bits 31:25 Reserved, must be kept at reset value.

Bit 24 **TSCEN:** Touch sensing controller clock enable  
Set and cleared by software.  
0: TSC clock disabled  
1: TSC clock enabled

Bit 23 Reserved, must be kept at reset value.

Bit 22 **IOPFEN:** I/O port F clock enable  
Set and cleared by software.  
0: I/O port F clock disabled  
1: I/O port F clock enabled

- Bit 21 **IOPEEN:** I/O port E clock enable  
Set and cleared by software.  
0: I/O port E clock disabled  
1: I/O port E clock enabled
- Bit 20 **IOPDEN:** I/O port D clock enable  
Set and cleared by software.  
0: I/O port D clock disabled  
1: I/O port D clock enabled
- Bit 19 **IOPCEN:** I/O port C clock enable  
Set and cleared by software.  
0: I/O port C clock disabled  
1: I/O port C clock enabled
- Bit 18 **IOPBEN:** I/O port B clock enable  
Set and cleared by software.  
0: I/O port B clock disabled  
1: I/O port B clock enabled
- Bit 17 **IOPAEN:** I/O port A clock enable  
Set and cleared by software.  
0: I/O port A clock disabled  
1: I/O port A clock enabled
- Bits 16:7 Reserved, must be kept at reset value.
- Bit 6 **CRCEN:** CRC clock enable  
Set and cleared by software.  
0: CRC clock disabled  
1: CRC clock enabled
- Bit 5 Reserved, must be kept at reset value.
- Bit 4 **FLITFEN:** FLITF clock enable  
Set and cleared by software to disable/enable FLITF clock during Sleep mode.  
0: FLITF clock disabled during Sleep mode  
1: FLITF clock enabled during Sleep mode
- Bit 3 Reserved, must be kept at reset value.
- Bit 2 **SRAMEN:** SRAM interface clock enable  
Set and cleared by software to disable/enable SRAM interface clock during Sleep mode.  
0: SRAM interface clock disabled during Sleep mode.  
1: SRAM interface clock enabled during Sleep mode
- Bit 1 **DMA2EN:** DMA2 clock enable  
Set and cleared by software.  
0: DMA2 clock disabled  
1: DMA2 clock enabled
- Bit 0 **DMAEN:** DMA clock enable  
Set and cleared by software.  
0: DMA clock disabled  
1: DMA clock enabled

### 6.4.7 APB peripheral clock enable register 2 (RCC\_APB2ENR)

Address: 0x18

Reset value: 0x0000 0000

Access: word, half-word and byte access

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

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DBG MCUEN	Res.	Res.	Res.	TIM17 EN	TIM16 EN	TIM15 EN
									rw				rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USART1 EN	Res.	SPI1 EN	TIM1 EN	Res.	ADC EN	Res.	USART8 EN	USART7 EN	USART6 EN	Res.	Res.	Res.	SYSCFG COMPEN	
	rw		rw	rw		rw		rw	rw	rw				rw	

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

Bit 22 **DBGMCUEN** MCU debug module clock enable

Set and reset by software.

- 0: MCU debug module clock disabled
- 1: MCU debug module enabled

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

Bit 18 **TIM17EN**: TIM17 timer clock enable

Set and cleared by software.

- 0: TIM17 timer clock disabled
- 1: TIM17 timer clock enabled

Bit 17 **TIM16EN**: TIM16 timer clock enable

Set and cleared by software.

- 0: TIM16 timer clock disabled
- 1: TIM16 timer clock enabled

Bit 16 **TIM15EN**: TIM15 timer clock enable

Set and cleared by software.

- 0: TIM15 timer clock disabled
- 1: TIM15 timer clock enabled

Bit 15 Reserved, must be kept at reset value.

Bit 14 **USART1EN**: USART1 clock enable

Set and cleared by software.

- 0: USART1clock disabled
- 1: USART1clock enabled

Bit 13 Reserved, must be kept at reset value.

- Bit 12 **SPI1EN:** SPI1 clock enable  
Set and cleared by software.  
0: SPI1 clock disabled  
1: SPI1 clock enabled
- Bit 11 **TIM1EN:** TIM1 timer clock enable  
Set and cleared by software.  
0: TIM1 timer clock disabled  
1: TIM1P timer clock enabled
- Bit 10 Reserved, must be kept at reset value.
- Bit 9 **ADCEN:** ADC interface clock enable  
Set and cleared by software.  
0: ADC interface disabled  
1: ADC interface clock enabled
- Bit 8 Reserved, must be kept at reset value.
- Bit 7 **USART8EN:** USART8 clock enable  
Set and cleared by software.  
0: USART8clock disabled  
1: USART8clock enabled
- Bit 6 **USART7EN:** USART7 clock enable  
Set and cleared by software.  
0: USART7clock disabled  
1: USART7clock enabled
- Bit 5 **USART6EN:** USART6 clock enable  
Set and cleared by software.  
0: USART6clock disabled  
1: USART6clock enabled
- Bits 4:1 Reserved, must be kept at reset value.
- Bit 0 **SYSCFGCOMPEN:** SYSCFG & COMP clock enable  
Set and cleared by software.  
0: SYSCFG & COMP clock disabled  
1: SYSCFG & COMP clock enabled

#### 6.4.8 APB peripheral clock enable register 1 (RCC\_APB1ENR)

Address: 0x1C

Reset value: 0x0000 0000

Access: word, half-word and byte access

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

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	CEC EN	DAC EN	PWR EN	CRS EN	Res.	CAN EN	Res.	USB EN	I2C2 EN	I2C1 EN	USART5 EN	USART4 EN	USART3 EN	USART2 EN	Res.
	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SPI2 EN	Res.	Res.	WWDG EN	Res.	Res.	TIM14 EN	Res.	Res.	TIM7 EN	TIM6 EN	Res.	Res.	TIM3 EN	TIM2 EN
	rw			rw			rw			rw	rw			rw	rw

Bit 31 Reserved, must be kept at reset value.

Bit 30 **CECEN:** HDMI CEC clock enable

Set and cleared by software.

- 0: HDMI CEC clock disabled
- 1: HDMI CEC clock enabled

Bit 29 **DACEN:** DAC interface clock enable

Set and cleared by software.

- 0: DAC interface clock disabled
- 1: DAC interface clock enabled

Bit 28 **PWREN:** Power interface clock enable

Set and cleared by software.

- 0: Power interface clock disabled
- 1: Power interface clock enabled

Bit 27 **CRSEN:** Clock recovery system interface clock enable

Set and cleared by software.

- 0: CRS interface clock disabled
- 1: CRS interface clock enabled

Bit 26 Reserved, must be kept at reset value.

Bit 25 **CANEN:** CAN interface clock enable

Set and cleared by software.

- 0: CAN interface clock disabled
- 1: CAN interface clock enabled

Bit 24 Reserved, must be kept at reset value.

Bit 23 **USBEN:** USB interface clock enable

Set and cleared by software.

- 0: USB interface clock disabled
- 1: USB interface clock enabled

Bit 22 **I2C2EN:** I2C2 clock enable

Set and cleared by software.

- 0: I2C2 clock disabled
- 1: I2C2 clock enabled

Bit 21 **I2C1EN:** I2C1 clock enable

Set and cleared by software.

- 0: I2C1 clock disabled
- 1: I2C1 clock enabled

- Bit 20 **USART5EN:** USART5 clock enable  
Set and cleared by software.  
0: USART5 clock disabled  
1: USART5 clock enabled
- Bit 19 **USART4EN:** USART4 clock enable  
Set and cleared by software.  
0: USART4 clock disabled  
1: USART4 clock enabled
- Bit 18 **USART3EN:** USART3 clock enable  
Set and cleared by software.  
0: USART3 clock disabled  
1: USART3 clock enabled
- Bit 17 **USART2EN:** USART2 clock enable  
Set and cleared by software.  
0: USART2 clock disabled  
1: USART2 clock enabled
- Bits 16:15 Reserved, must be kept at reset value.
- Bit 14 **SPI2EN:** SPI2 clock enable  
Set and cleared by software.  
0: SPI2 clock disabled  
1: SPI2 clock enabled
- Bits 13:12 Reserved, must be kept at reset value.
- Bit 11 **WWDGEN:** Window watchdog clock enable  
Set and cleared by software.  
0: Window watchdog clock disabled  
1: Window watchdog clock enabled
- Bits 10:9 Reserved, must be kept at reset value.
- Bit 8 **TIM14EN:** TIM14 timer clock enable  
Set and cleared by software.  
0: TIM14 clock disabled  
1: TIM14 clock enabled
- Bits 7:6 Reserved, must be kept at reset value.
- Bit 5 **TIM7EN:** TIM7 timer clock enable  
Set and cleared by software.  
0: TIM7 clock disabled  
1: TIM7 clock enabled
- Bit 4 **TIM6EN:** TIM6 timer clock enable  
Set and cleared by software.  
0: TIM6 clock disabled  
1: TIM6 clock enabled
- Bits 3:2 Reserved, must be kept at reset value.

Bit 1 **TIM3EN:** TIM3 timer clock enable

Set and cleared by software.

0: TIM3 clock disabled

1: TIM3 clock enabled

Bit 0 **TIM2EN:** TIM2 timer clock enable

Set and cleared by software.

0: TIM2 clock disabled

1: TIM2 clock enabled

#### 6.4.9 RTC domain control register (RCC\_BDCR)

Address offset: 0x20

Reset value: 0x0000 0018, reset by RTC domain reset.

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

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

Note:

The LSEON, LSEBYP, RTCSEL and RTCEN bits of the *RTC domain control register (RCC\_BDCR)* are in the RTC domain. As a result, after reset, these bits are write-protected and the DBP bit in the *Power control register (PWR\_CR)* has to be set before they can be modified. Refer to [Section 5.1.3: Battery backup domain](#) for further information. These bits are only reset after a RTC domain reset (see [Section 6.1.3: RTC domain reset](#)). Any internal or external reset does not have any effect on these bits.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	BDRST
															rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTC EN	Res.	Res.	Res.	Res.	Res.	Res.	RTCSEL[1:0]	Res.	Res.	Res.	LSEDRV[1:0]	LSE BYP	LSE RDY	LSEON	
rw						rw	rw				rw	rw	rw	r	rw

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

Bit 16 **BDRST:** RTC domain software reset

Set and cleared by software.

0: Reset not activated

1: Resets the entire RTC domain

Bit 15 **RTCEN:** RTC clock enable

Set and cleared by software.

0: RTC clock disabled

1: RTC clock enabled

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

Bits 9:8 **RTCSEL[1:0]:** RTC clock source selection

Set by software to select the clock source for the RTC. Once the RTC clock source has been selected, it cannot be changed anymore unless the RTC domain is reset. The BDRST bit can be used to reset them.

- 00: No clock
- 01: LSE oscillator clock used as RTC clock
- 10: LSI oscillator clock used as RTC clock
- 11: HSE oscillator clock divided by 32 used as RTC clock

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

Bits 4:3 **LSEDRV:** LSE oscillator drive capability

Set and reset by software to modulate the LSE oscillator's drive capability. A reset of the RTC domain restores the default value.

- 00: 'Xtal mode' low drive capability
- 01: 'Xtal mode' medium-high drive capability
- 10: 'Xtal mode' medium-low drive capability
- 11: 'Xtal mode' high drive capability (reset value)

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

Bit 2 **LSEBYP:** LSE oscillator bypass

Set and cleared by software to bypass oscillator in debug mode. This bit can be written only when the external 32 kHz oscillator is disabled.

- 0: LSE oscillator not bypassed
- 1: LSE oscillator bypassed

Bit 1 **LSERDY:** LSE oscillator ready

Set and cleared by hardware to indicate when the external 32 kHz oscillator is stable. After the LSEON bit is cleared, LSERDY goes low after 6 external low-speed oscillator clock cycles.

- 0: LSE oscillator not ready
- 1: LSE oscillator ready

Bit 0 **LSEON:** LSE oscillator enable

Set and cleared by software.

- 0: LSE oscillator OFF
- 1: LSE oscillator ON

**6.4.10 Control/status register (RCC\_CSR)**

Address: 0x24

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

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

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LPWR RSTF	WWDG RSTF	IWDG RSTF	SFT RSTF	POR RSTF	PIN RSTF	OB LRSTF	RMVF	V18PWR RSTF	Res.	Res.	Res.	Res.	Res.	Res.	Res.
r	r	r	r	r	r	r	rt_w	r							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	LSI RDY	LSION
														r	rw

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

Set by hardware when a Low-power management reset occurs.  
Cleared by writing to the RMVF bit.

- 0: No Low-power management reset occurred
- 1: Low-power management reset occurred

For further information refer to [Low-power management reset](#).

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

Set by hardware when a window watchdog reset occurs.  
Cleared by writing to the RMVF bit.

- 0: No window watchdog reset occurred
- 1: Window watchdog reset occurred

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

Set by hardware when an independent watchdog reset from V<sub>DD</sub> domain occurs.  
Cleared by writing to the RMVF bit.

- 0: No watchdog reset occurred
- 1: Watchdog reset occurred

Bit 28 **SFRSTF:** Software reset flag

Set by hardware when a software reset occurs.  
Cleared by writing to the RMVF bit.

- 0: No software reset occurred
- 1: Software reset occurred

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

Set by hardware when a POR/PDR reset occurs.  
Cleared by writing to the RMVF bit.

- 0: No POR/PDR reset occurred
- 1: POR/PDR reset occurred

Bit 26 **PINRSTF:** PIN reset flag

Set by hardware when a reset from the NRST pin occurs.  
Cleared by writing to the RMVF bit.

- 0: No reset from NRST pin occurred
- 1: Reset from NRST pin occurred

Bit 25 **OBLRSTF:** Option byte loader reset flag

Set by hardware when a reset from the OBL occurs.  
Cleared by writing to the RMVF bit.

- 0: No reset from OBL occurred
- 1: Reset from OBL occurred

Bit 24 **RMVF:** Remove reset flag

Set by software to clear the reset flags including RMVF.

- 0: No effect
- 1: Clear the reset flags

Bit 23 **V18PWRRSTF:** Reset flag of the 1.8 V domain.

Set by hardware when a POR/PDR of the 1.8 V domain occurred.  
Cleared by writing to the RMVF bit.

- 0: No POR/PDR reset of the 1.8 V domain occurred
- 1: POR/PDR reset of the 1.8 V domain occurred

**Caution:** On the STM32F0x8 family, this flag must be read as reserved.

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

**Bit 1 LSIRDY:** LSI oscillator ready

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

0: LSI oscillator not ready

1: LSI oscillator ready

**Bit 0 LSION:** LSI oscillator enable

Set and cleared by software.

0: LSI oscillator OFF

1: LSI oscillator ON

#### 6.4.11 AHB peripheral reset register (RCC\_AHBRSTR)

Address: 0x28

Reset value: 0x0000 0000

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	TSC RST	Res.	IOPF RST	IOPE RST	IOPD RST	IOPC RST	IOPB RST	IOPA RST	Res.						
							rw		rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.							

Bits 31:25 Reserved, must be kept at reset value.

**Bit 24 TSCRST:** Touch sensing controller reset

Set and cleared by software.

0: No effect

1: Reset TSC

Bit 23 Reserved, must be kept at reset value.

**Bit 22 IOPFRST:** I/O port F reset

Set and cleared by software.

0: No effect

1: Reset I/O port F

**Bit 21 IOPERST:** I/O port E reset

Set and cleared by software.

0: No effect

1: Reset I/O port E

**Bit 20 IOPDRST:** I/O port D reset

Set and cleared by software.

0: No effect

1: Reset I/O port D

Bit 19 **IOPCRST:** I/O port C reset

Set and cleared by software.

0: No effect

1: Reset I/O port C

Bit 18 **IOPBRST:** I/O port B reset

Set and cleared by software.

0: No effect

1: Reset I/O port B

Bit 17 **IOPARST:** I/O port A reset

Set and cleared by software.

0: No effect

1: Reset I/O port A

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

#### 6.4.12 Clock configuration register 2 (RCC\_CFGR2)

Address: 0x2C

Reset value: 0x0000 0000

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

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

Bits 31:4 Reserved, must be kept at reset value.

### Bits 3:0 **PREDIV[3:0]** PREDIV division factor

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

*Note: Bit 0 is the same bit as bit 17 in Clock configuration register (RCC\_CFGR), so modifying bit 17 in Clock configuration register (RCC\_CFGR) also modifies bit 0 in Clock configuration register 2 (RCC\_CFGR2) (for compatibility with other STM32 products)*

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

### 6.4.13 Clock configuration register 3 (RCC\_CFGR3)

Address: 0x30

Reset value: 0x0000 0000

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	USART3SW[1:0]	USART2SW[1:0]									
												rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	ADC SW	USB SW	CEC SW	Res.	I2C1 SW	Res.	Res.	USART1SW[1:0]							
							rw	rw	rw		rw			rw	rw

Bits 31:20 Reserved, must be kept at reset value.

Bits 19:18 **USART3SW[1:0]**: USART3 clock source selection (available only on STM32F09x devices)

This bit is set and cleared by software to select the USART3 clock source.

- 00: PCLK selected as USART3 clock source (default)
- 01: System clock (SYSCLK) selected as USART3 clock
- 10: LSE clock selected as USART3 clock
- 11: HSI clock selected as USART3 clock

Bits 17:16 **USART2SW[1:0]**: USART2 clock source selection (available only on STM32F07x and STM32F09x devices)

This bit is set and cleared by software to select the USART2 clock source.

00: PCLK selected as USART2 clock source (default)

01: System clock (SYSCLK) selected as USART2 clock

10: LSE clock selected as USART2 clock

11: HSI clock selected as USART2 clock

Bits 15:9 Reserved, must be kept at reset value.

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

Obsolete setting. To be kept at reset value, connecting the HSI14 clock to the ADC asynchronous clock input. Proper ADC clock selection is done inside the ADC\_CFGR2 (refer to [Section 13.11.5: ADC configuration register 2 \(ADC\\_CFGR2\) on page 271](#)).

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

This bit is set and cleared by software to select the USB clock source.

0: HSI48 clock selected as USB clock source (default)

1: PLL clock (PLLCLK) selected as USB clock

Bit 6 **CECSW**: HDMI CEC clock source selection

This bit is set and cleared by software to select the CEC clock source.

0: HSI clock, divided by 244, selected as CEC clock (default)

1: LSE clock selected as CEC clock

Bit 5 Reserved, must be kept at reset value.

Bit 4 **I2C1SW**: I2C1 clock source selection

This bit is set and cleared by software to select the I2C1 clock source.

0: HSI clock selected as I2C1 clock source (default)

1: System clock (SYSCLK) selected as I2C1 clock

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

Bits 1:0 **USART1SW[1:0]**: USART1 clock source selection

This bit is set and cleared by software to select the USART1 clock source.

00: PCLK selected as USART1 clock source (default)

01: System clock (SYSCLK) selected as USART1 clock

10: LSE clock selected as USART1 clock

11: HSI clock selected as USART1 clock

#### 6.4.14 Clock control register 2 (RCC\_CR2)

Address: 0x34

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

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
HSI48CAL[7:0]								Res.	Res.	Res.	Res.	Res.	Res.	HSI48 RDY	HSI48 ON	
r	r	r	r	r	r	r	r							r	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
HSI14CAL[7:0]								HSI14TRIM[4:0]						HSI14 DIS	HSI14 RDY	HSI14 ON
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	r	rw	

Bits 31:24 **HSI48CAL[7:0]:** HSI48 factory clock calibration

These bits are initialized automatically at startup and are read-only.

Bits 23:18 Reserved, must be kept at reset value.

Bit 17 **HSI48RDY:** HSI48 clock ready flag

Set by hardware to indicate that HSI48 oscillator is stable. After the HSI48ON bit is cleared, HSI48RDY goes low after 6 HSI48 oscillator clock cycles.

- 0: HSI48 oscillator not ready
- 1: HSI48 oscillator ready

Bit 16 **HSI48ON:** HSI48 clock enable

Set and cleared either by software or by hardware. Set by hardware when the USB peripheral is enabled and switched on this source; reset by hardware to stop the oscillator when entering in Stop or Standby mode. This bit cannot be reset if the HSI48 is used directly or indirectly as system clock or is selected to become the system clock.

- 0: HSI48 oscillator OFF
- 1: HSI48 oscillator ON

Bits 15:8 **HSI14CAL[7:0]:** HSI14 clock calibration

These bits are initialized automatically at startup.

Bits 7:3 **HSI14TRIM[4:0]:** HSI14 clock trimming

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

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

Bit 2 **HSI14DIS:** HSI14 clock request from ADC disable

Set and cleared by software.

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

- 0: ADC interface can turn on the HSI14 oscillator
- 1: ADC interface can not turn on the HSI14 oscillator

Bit 1 **HSI14RDY:** HSI14 clock ready flag

Set by hardware to indicate that HSI14 oscillator is stable. After the HSI14ON bit is cleared, HSI14RDY goes low after 6 HSI14 oscillator clock cycles. When HSI14ON is not set but the HSI14 oscillator is enabled by the peripheral through a clock request, this bit is not set.

- 0: HSI14 oscillator not ready
- 1: HSI14 oscillator ready

**Bit 0 HSI14ON:** HSI14 clock enable

Set and cleared by software. When the HSI14 oscillator is enabled by the peripheral through a clock request, this bit is not set and resetting it does not stop the HSI14 oscillator.

- 0: HSI14 oscillator OFF
- 1: HSI14 oscillator ON

#### 6.4.15 RCC register map

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

**Table 19. RCC register map and reset values**

Offset	Register	Reset value	HSICAL[7:0]	HSITRIM[4:0]
0x00	RCC_CR	Res.	PLL NODIV	31
	Reset value	0	PLL NODIV	Res.
0x04	RCC_CFGR	Res.	MCOPRE [2:0]	30
	Reset value	0	MCOPRE [2:0]	Res.
0x08	RCC_CIR	Res.	MCO [3:0]	29
	Reset value	0	MCO [3:0]	Res.
0x0C	RCC_APB2RSTR	Res.	PLL RDY	28
	Reset value	0	PLL RDY	Res.
0x10	RCC_APB1RSTR	Res.	PLL ON	27
	Reset value	0	PLL ON	Res.
0x14	RCC_AHBENR	Res.	CSSC	26
	Reset value	0	CSSC	Res.
0x18	RCC_APB2ENR	Res.	HSI48RDYC	25
	Reset value	0	HSI48RDYC	Res.
0x1C	RCC_APB1ENR	Res.	PLL MUL[3:0]	24
	Reset value	0	PLL MUL[3:0]	Res.
0x20	RCC_BDCR	Res.	PLL RDYC	23
	Reset value	0	PLL RDYC	Res.
0x24	RCC_CSR	Res.	HSEBYP	22
	Reset value	0	HSEBYP	Res.

**Table 19. RCC register map and reset values (continued)**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x28	<b>RCC_AHBRSTR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TSC RST	Res.	IOPF RST	0	.IOPERST	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		Reset value																															
0x2C	<b>RCC_CFGR2</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
		Reset value																															
0x30	<b>RCC_CFGR3</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
		Reset value																															
0x34	<b>RCC_CR2</b>	HSI48CAL[7:0]					Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
		Reset value	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 7 Clock recovery system (CRS)

This section applies to STM32F04x, STM32F07x and STM32F09x devices only.

### 7.1 Introduction

The clock recovery system (CRS) is an advanced digital controller acting on the internal fine-granularity trimmable RC oscillator HSI48. The CRS provides powerful means for oscillator output frequency evaluation, based on comparison with a selectable synchronization signal. It is capable of doing automatic adjustment of oscillator trimming based on the measured frequency error value, while keeping the possibility of a manual trimming.

The CRS is ideally suited to provide a precise clock to the USB peripheral. In such case, the synchronization signal can be derived from the start-of-frame (SOF) packet signalization on the USB bus, which is sent by a USB host at 1 ms intervals.

The synchronization signal can also be derived from the LSE oscillator output, from an external pin, or it can be generated by user software.

### 7.2 CRS main features

- Selectable synchronization source with programmable prescaler and polarity:
  - External pin
  - LSE oscillator output
  - USB SOF packet reception
- Possibility to generate synchronization pulses by software
- Automatic oscillator trimming capability with no need of CPU action
- Manual control option for faster start-up convergence
- 16-bit frequency error counter with automatic error value capture and reload
- Programmable limit for automatic frequency error value evaluation and status reporting
- Maskable interrupts/events:
  - Expected synchronization (ESYNC)
  - Synchronization OK (SYNCOK)
  - Synchronization warning (SYNCWARN)
  - Synchronization or trimming error (ERR)

### 7.3 CRS implementation

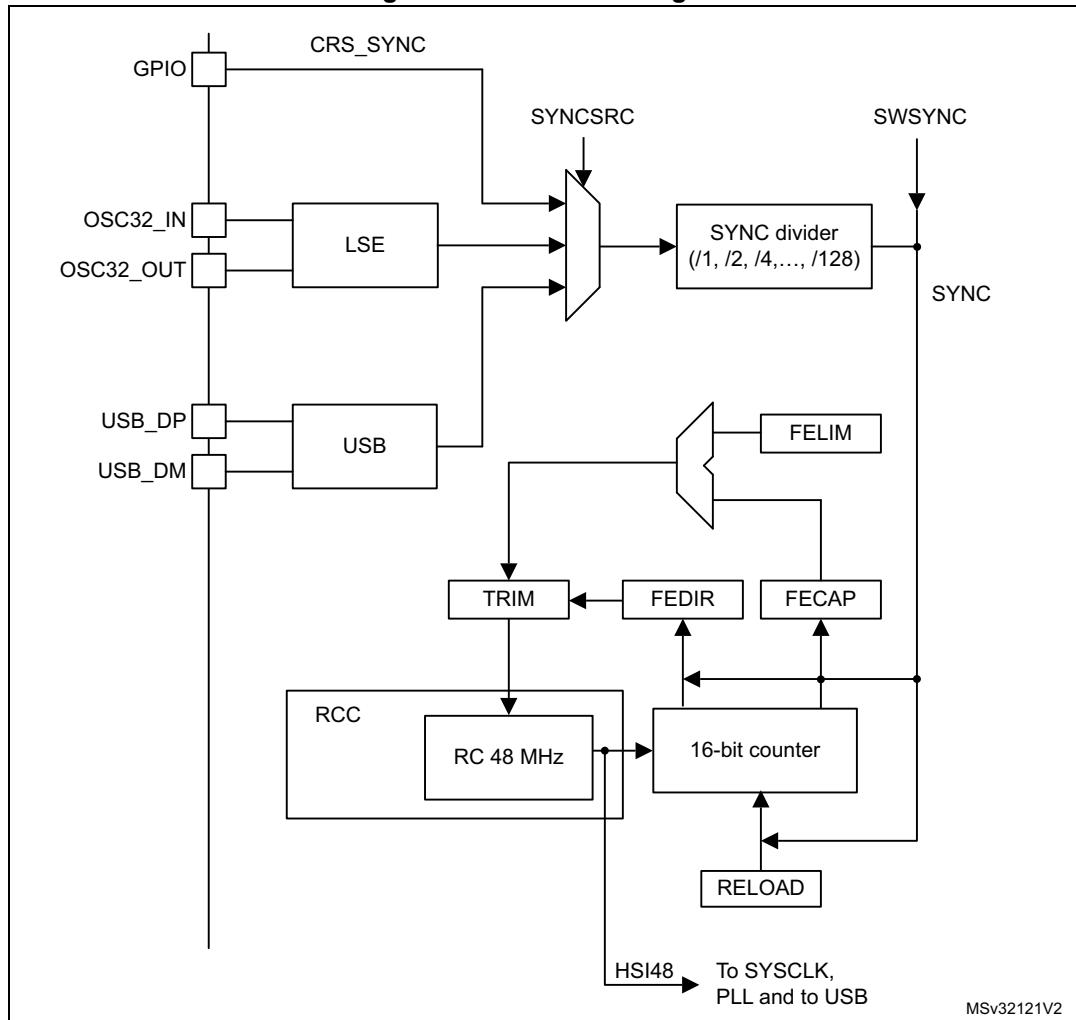
Table 20. CRS features

Feature	CRS1
TRIM width	6 bits

## 7.4 CRS functional description

### 7.4.1 CRS block diagram

Figure 14. CRS block diagram



### 7.4.2 Synchronization input

The CRS synchronization (SYNC) source, selectable through the CRS\_CFGR register, can be the signal from the external CRS\_SYNC pin, the LSE clock or the USB SOF signal. For a better robustness of the SYNC input, a simple digital filter (2 out of 3 majority votes, sampled by the RC48 clock) is implemented to filter out any glitches. This source signal also has a configurable polarity and can then be divided by a programmable binary prescaler to obtain a synchronization signal in a suitable frequency range (usually around 1 kHz).

For more information on the CRS synchronization source configuration, refer to [Section 7.7.2: CRS configuration register \(CRS\\_CFGR\)](#).

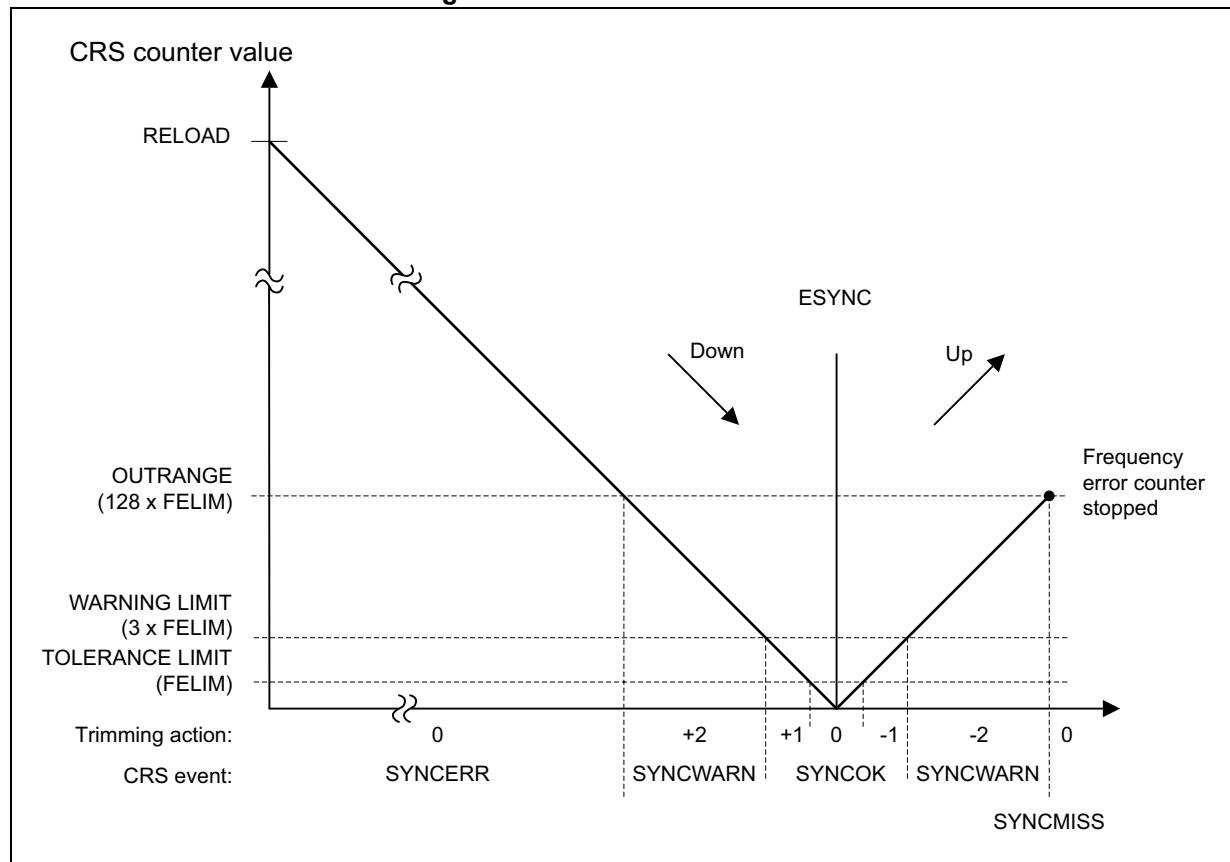
It is also possible to generate a synchronization event by software, by setting the SWSYNC bit in the CRS\_CR register.

### 7.4.3 Frequency error measurement

The frequency error counter is a 16-bit down/up counter which is reloaded with the RELOAD value on each SYNC event. It starts counting down till it reaches the zero value, where the ESYNC (expected synchronization) event is generated. Then it starts counting up to the OUTRANGE limit where it eventually stops (if no SYNC event is received) and generates a SYNCMISS event. The OUTRANGE limit is defined as the frequency error limit (FELIM field of the CRS\_CFG register) multiplied by 128.

When the SYNC event is detected, the actual value of the frequency error counter and its counting direction are stored in the FECAP (frequency error capture) field and in the FEDIR (frequency error direction) bit of the CRS\_ISR register. When the SYNC event is detected during the downcounting phase (before reaching the zero value), it means that the actual frequency is lower than the target (and so, that the TRIM value must be incremented), while when it is detected during the upcounting phase it means that the actual frequency is higher (and that the TRIM value must be decremented).

**Figure 15. CRS counter behavior**



### 7.4.4 Frequency error evaluation and automatic trimming

The measured frequency error is evaluated by comparing its value with a set of limits:

- TOLERANCE LIMIT, given directly in the FELIM field of the CRS\_CFG register
- WARNING LIMIT, defined as  $3 \times$  FELIM value
- OUTRANGE (error limit), defined as  $128 \times$  FELIM value

The result of this comparison is used to generate the status indication and also to control the automatic trimming which is enabled by setting the AUTOTRIMEN bit in the CRS\_CR register:

- When the frequency error is below the tolerance limit, it means that the actual trimming value in the TRIM field is the optimal one, hence no trimming action is needed.
  - SYNCOK status indicated
  - TRIM value not changed in AUTOTRIM mode
- When the frequency error is below the warning limit but above or equal to the tolerance limit, it means that some trimming action is necessary but that adjustment by one trimming step is enough to reach the optimal TRIM value.
  - SYNCOK status indicated
  - TRIM value adjusted by one trimming step in AUTOTRIM mode
- When the frequency error is above or equal to the warning limit but below the error limit, it means that a stronger trimming action is necessary, and there is a risk that the optimal TRIM value is not reached for the next period.
  - SYNCWARN status indicated
  - TRIM value adjusted by two trimming steps in AUTOTRIM mode
- When the frequency error is above or equal to the error limit, it means that the frequency is out of the trimming range. This can also happen when the SYNC input is not clean or when some SYNC pulse is missing (for example when one USB SOF is corrupted).
  - SYNCERR or SYNCMISS status indicated
  - TRIM value not changed in AUTOTRIM mode

*Note:* If the actual value of the TRIM field is so close to its limits that the automatic trimming would force it to overflow or underflow, then the TRIM value is set just to the limit and the TRIMOVF status is indicated.

In AUTOTRIM mode (AUTOTRIMEN bit set in the CRS\_CR register), the TRIM field of CRS\_CR is adjusted by hardware and is read-only.

## 7.4.5 CRS initialization and configuration

### RELOAD value

The RELOAD value must be selected according to the ratio between the target frequency and the frequency of the synchronization source after prescaling. It is then decreased by one to reach the expected synchronization on the zero value. The formula is the following:

$$\text{RELOAD} = (f_{\text{TARGET}} / f_{\text{SYNC}}) - 1$$

The reset value of the RELOAD field corresponds to a target frequency of 48 MHz and a synchronization signal frequency of 1 kHz (SOF signal from USB).

### FELIM value

The selection of the FELIM value is closely coupled with the HSI48 oscillator characteristics and its typical trimming step size. The optimal value corresponds to half of the trimming step size, expressed as a number of HSI48 oscillator clock ticks. The following formula can be used:

$$\text{FELIM} = (f_{\text{TARGET}} / f_{\text{SYNC}}) * \text{STEP}[\%] / 100\% / 2$$

The result must be always rounded up to the nearest integer value to obtain the best trimming response. If frequent trimming actions are not needed in the application, the hysteresis can be increased by slightly increasing the FELIM value.

The reset value of the FELIM field corresponds to  $(f_{\text{TARGET}} / f_{\text{SYNC}}) = 48000$  and to a typical trimming step size of 0.14%.

**Note:** *The trimming step size depends upon the product, check the datasheet for accurate setting.*

**Caution:** There is no hardware protection from a wrong configuration of the RELOAD and FELIM fields which can lead to an erratic trimming response. The expected operational mode requires proper setup of the RELOAD value (according to the synchronization source frequency), which is also greater than  $128 * \text{FELIM}$  value (OUTRANGE limit).

## 7.5 CRS low-power modes

**Table 21. Effect of low-power modes on CRS**

Mode	Description
Sleep	No effect. CRS interrupts cause the device to exit the Sleep mode.
Stop	CRS registers are frozen. The CRS stops operating until the Stop mode is exited and the HSI48 oscillator restarted.
Standby	The CRS peripheral is powered down and must be reinitialized after exiting Standby mode.

## 7.6 CRS interrupts

**Table 22. Interrupt control bits**

Interrupt event	Event flag	Enable control bit	Clear flag bit
Expected synchronization	ESYNCF	ESYNCIE	ESYNCC
Synchronization OK	SYNCOKF	SYNCOKIE	SYNCOKC
Synchronization warning	SYNCWARNF	SYNCWARNIE	SYNCWARNC
Synchronization or trimming error (TRIMOVF, SYNCMISS, SYNCERR)	ERRF	ERRIE	ERRC

## 7.7 CRS registers

Refer to [Section 1.2 on page 42](#) for a list of abbreviations used in register descriptions.

The peripheral registers can be accessed only by words (32-bit).

### 7.7.1 CRS control register (CRS\_CR)

Address offset: 0x000

Reset value: 0x0000 2000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	TRIM[5:0]						SW SYNC	AUTO TRIMEN	CEN	Res.	ESYNCIE	ERRIE	SYNC WARNIE	SYNC OKIE
		RW	RW	RW	RW	RW	rt_w1	RW	RW		RW	RW	RW	RW	RW

Bits 31:14 Reserved, must be kept at reset value.

Bits 13:8 **TRIM[5:0]**: HSI48 oscillator smooth trimming

These bits provide a user-programmable trimming value to the HSI48 oscillator. They can be programmed to adjust to variations in voltage and temperature that influence the frequency of the HSI48 oscillator.

The default value is 32, which corresponds to the middle of the trimming interval. The trimming step is specified in the product datasheet. A higher TRIM value corresponds to a higher output frequency.

When the AUTOTRIMEN bit is set, this field is controlled by hardware and is read-only.

Bit 7 **SWSYNC**: Generate software SYNC event

This bit is set by software in order to generate a software SYNC event. It is automatically cleared by hardware.

0: No action

1: A software SYNC event is generated.

Bit 6 **AUTOTRIMEN**: Automatic trimming enable

This bit enables the automatic hardware adjustment of TRIM bits according to the measured frequency error between two SYNC events. If this bit is set, the TRIM bits are read-only. The TRIM value can be adjusted by hardware by one or two steps at a time, depending on the measured frequency error value. Refer to [Section 7.4.4](#) for more details.

0: Automatic trimming disabled, TRIM bits can be adjusted by the user.

1: Automatic trimming enabled, TRIM bits are read-only and under hardware control.

Bit 5 **CEN**: Frequency error counter enable

This bit enables the oscillator clock for the frequency error counter.

0: Frequency error counter disabled

1: Frequency error counter enabled

When this bit is set, the CRS\_CFG register is write-protected and cannot be modified.

Bit 4 Reserved, must be kept at reset value.

Bit 3 **ESYNCIE**: Expected SYNC interrupt enable

0: Expected SYNC (ESYNCF) interrupt disabled

1: Expected SYNC (ESYNCF) interrupt enabled

- Bit 2 **ERRIE**: Synchronization or trimming error interrupt enable  
 0: Synchronization or trimming error (ERRF) interrupt disabled  
 1: Synchronization or trimming error (ERRF) interrupt enabled
- Bit 1 **SYNCWARNIE**: SYNC warning interrupt enable  
 0: SYNC warning (SYNCWARNF) interrupt disabled  
 1: SYNC warning (SYNCWARNF) interrupt enabled
- Bit 0 **SYNCOKIE**: SYNC event OK interrupt enable  
 0: SYNC event OK (SYNCOKF) interrupt disabled  
 1: SYNC event OK (SYNCOKF) interrupt enabled

## 7.7.2 CRS configuration register (CRS\_CFGR)

This register can be written only when the frequency error counter is disabled (CEN bit is cleared in CRS\_CR). When the counter is enabled, this register is write-protected.

Address offset: 0x04

Reset value: 0x2022 BB7F

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SYNCPOL	Res.	SYNCSRC[1:0]		Res.	SYNCDIV[2:0]			FELIM[7:0]							
rw		rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RELOAD[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **SYNCPOL**: SYNC polarity selection

This bit is set and cleared by software to select the input polarity for the SYNC signal source.  
 0: SYNC active on rising edge (default)  
 1: SYNC active on falling edge

Bit 30 Reserved, must be kept at reset value.

Bits 29:28 **SYNCSRC[1:0]**: SYNC signal source selection

These bits are set and cleared by software to select the SYNC signal source.  
 00: GPIO selected as SYNC signal source  
 01: LSE selected as SYNC signal source  
 10: USB SOF selected as SYNC signal source (default).  
 11: Reserved

*Note: When using USB LPM (Link Power Management) and the device is in Sleep mode, the periodic USB SOF is not generated by the host. No SYNC signal is therefore provided to the CRS to calibrate the HSI48 oscillator on the run. To guarantee the required clock precision after waking up from Sleep mode, the LSE or reference clock on the GPIOs should be used as SYNC signal.*

Bit 27 Reserved, must be kept at reset value.

Bits 26:24 **SYNCDIV[2:0]**: SYNC divider

These bits are set and cleared by software to control the division factor of the SYNC signal.

- 000: SYNC not divided (default)
- 001: SYNC divided by 2
- 010: SYNC divided by 4
- 011: SYNC divided by 8
- 100: SYNC divided by 16
- 101: SYNC divided by 32
- 110: SYNC divided by 64
- 111: SYNC divided by 128

Bits 23:16 **FELIM[7:0]**: Frequency error limit

FELIM contains the value to be used to evaluate the captured frequency error value latched in the FECAP[15:0] bits of the CRS\_ISR register. Refer to [Section 7.4.4](#) for more details about FECAP evaluation.

Bits 15:0 **RELOAD[15:0]**: Counter reload value

RELOAD is the value to be loaded in the frequency error counter with each SYNC event. Refer to [Section 7.4.3](#) for more details about counter behavior.

### 7.7.3 CRS interrupt and status register (CRS\_ISR)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FECAP[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FEDIR	Res.	Res.	Res.	Res.	TRIM OVF	SYNC MISS	SYNC ERR	Res.	Res.	Res.	Res.	ESYNCF	ERRF	SYNC WARNF	SYNC OKF
r					r	r	r					r	r	r	r

Bits 31:16 **FECAP[15:0]**: Frequency error capture

FECAP is the frequency error counter value latched in the time of the last SYNC event.

Refer to [Section 7.4.4](#) for more details about FECAP usage.

Bit 15 **FEDIR**: Frequency error direction

FEDIR is the counting direction of the frequency error counter latched in the time of the last SYNC event. It shows whether the actual frequency is below or above the target.

0: Upcounting direction, the actual frequency is above the target.

1: Downcounting direction, the actual frequency is below the target.

Bits 14:11 Reserved, must be kept at reset value.

Bit 10 **TRIMOVF**: Trimming overflow or underflow

This flag is set by hardware when the automatic trimming tries to over- or under-flow the TRIM value. An interrupt is generated if the ERRIE bit is set in the CRS\_CR register. It is cleared by software by setting the ERRC bit in the CRS\_ICR register.

0: No trimming error signalized

1: Trimming error signalized

Bit 9 **SYNCMISS**: SYNC missed

This flag is set by hardware when the frequency error counter reached value FELIM \* 128 and no SYNC was detected, meaning either that a SYNC pulse was missed or that the frequency error is too big (internal frequency too high) to be compensated by adjusting the TRIM value, and that some other action has to be taken. At this point, the frequency error counter is stopped (waiting for a next SYNC) and an interrupt is generated if the ERRIE bit is set in the CRS\_CR register. It is cleared by software by setting the ERRC bit in the CRS\_ICR register.

- 0: No SYNC missed error signalized
- 1: SYNC missed error signalized

Bit 8 **SYNCERR**: SYNC error

This flag is set by hardware when the SYNC pulse arrives before the ESYNC event and the measured frequency error is greater than or equal to FELIM \* 128. This means that the frequency error is too big (internal frequency too low) to be compensated by adjusting the TRIM value, and that some other action has to be taken. An interrupt is generated if the ERRIE bit is set in the CRS\_CR register. It is cleared by software by setting the ERRC bit in the CRS\_ICR register.

- 0: No SYNC error signalized
- 1: SYNC error signalized

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

Bit 3 **ESYNCF**: Expected SYNC flag

This flag is set by hardware when the frequency error counter reached a zero value. An interrupt is generated if the ESYNCIE bit is set in the CRS\_CR register. It is cleared by software by setting the ESYNCC bit in the CRS\_ICR register.

- 0: No expected SYNC signalized
- 1: Expected SYNC signalized

Bit 2 **ERRF**: Error flag

This flag is set by hardware in case of any synchronization or trimming error. It is the logical OR of the TRIMOVF, SYNCMISS and SYNCERR bits. An interrupt is generated if the ERRIE bit is set in the CRS\_CR register. It is cleared by software in reaction to setting the ERRC bit in the CRS\_ICR register, which clears the TRIMOVF, SYNCMISS and SYNCERR bits.

- 0: No synchronization or trimming error signalized
- 1: Synchronization or trimming error signalized

Bit 1 **SYNCWARNF**: SYNC warning flag

This flag is set by hardware when the measured frequency error is greater than or equal to FELIM \* 3, but smaller than FELIM \* 128. This means that to compensate the frequency error, the TRIM value must be adjusted by two steps or more. An interrupt is generated if the SYNCWARNIE bit is set in the CRS\_CR register. It is cleared by software by setting the SYNCWARNC bit in the CRS\_ICR register.

- 0: No SYNC warning signalized
- 1: SYNC warning signalized

Bit 0 **SYNCOKF**: SYNC event OK flag

This flag is set by hardware when the measured frequency error is smaller than FELIM \* 3. This means that either no adjustment of the TRIM value is needed or that an adjustment by one trimming step is enough to compensate the frequency error. An interrupt is generated if the SYNCOKIE bit is set in the CRS\_CR register. It is cleared by software by setting the SYNCOKC bit in the CRS\_ICR register.

- 0: No SYNC event OK signalized
- 1: SYNC event OK signalized

### 7.7.4 CRS interrupt flag clear register (CRS\_ICR)

Address offset: 0x0C

Reset value: 0x0000 0000

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

Bits 31:4 Reserved, must be kept at reset value.

Bit 3 **ESYNCC**: Expected SYNC clear flag

Writing 1 to this bit clears the ESYNCF flag in the CRS\_ISR register.

Bit 2 **ERRC**: Error clear flag

Writing 1 to this bit clears TRIMOVF, SYNCMISS and SYNCERR bits and consequently also the ERRF flag in the CRS\_ISR register.

Bit 1 **SYNCWARNC**: SYNC warning clear flag

Writing 1 to this bit clears the SYNCWARNF flag in the CRS\_ISR register.

Bit 0 **SYNCOKC**: SYNC event OK clear flag

Writing 1 to this bit clears the SYNCOKF flag in the CRS\_ISR register.

### 7.7.5 CRS register map

Table 23. CRS register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x00	<b>CRS_CR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
0x04	<b>CRS_CFGR</b>	SyncPol	Res.	Sync SRC [1:0]	Res.	Sync DIV [2:0]	FELIM[7:0]							RELOAD[15:0]																ERRF	SyncWarnf	SyncOkf		
		0	1	0	0	0	0	0	0	1	0	0	0	1	0	1	0	1	0	1	1	1	1	0	1	1	1	1	1	1	1	1		
0x08	<b>CRS_ISR</b>	FECAP[15:0]															FEDIR	Res.	Res.	Res.	Res.													
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

**Table 23. CRS register map and reset values (continued)**

<b>Offset</b>	<b>Register</b>	<b>31</b>	<b>30</b>	<b>29</b>	<b>28</b>	<b>27</b>	<b>26</b>	<b>25</b>	<b>24</b>	<b>23</b>	<b>22</b>	<b>21</b>	<b>20</b>	<b>19</b>	<b>18</b>	<b>17</b>	<b>16</b>	<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>10</b>	<b>9</b>	<b>8</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>
0x0C	<b>CRS_ICR</b>	Res.	Res.	Res.	ESYNCC	ERRC	SYNCWARNC	SYNCOKC	0																								
	Reset value																								0	0	0	0	0	0			

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 8 General-purpose I/Os (GPIO)

### 8.1 Introduction

Each general-purpose I/O port has four 32-bit configuration registers (GPIO<sub>x</sub>\_MODER, GPIO<sub>x</sub>\_OTYPER, GPIO<sub>x</sub>\_OSPEEDR and GPIO<sub>x</sub>\_PUPDR), two 32-bit data registers (GPIO<sub>x</sub>\_IDR and GPIO<sub>x</sub>\_ODR) and a 32-bit set/reset register (GPIO<sub>x</sub>\_BSRR). Ports A and B also have a 32-bit locking register (GPIO<sub>x</sub>\_LCKR) and two 32-bit alternate function selection registers (GPIO<sub>x</sub>\_AFRH and GPIO<sub>x</sub>\_AFRL).

On STM32F07x and STM32F09x devices, also ports C, D, E, and F have two 32-bit alternate function selection registers (GPIO<sub>x</sub>\_AFRH and GPIO<sub>x</sub>\_AFRL).

Port E is available on STM32F07x and STM32F09x devices only.

### 8.2 GPIO main features

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

### 8.3 GPIO functional description

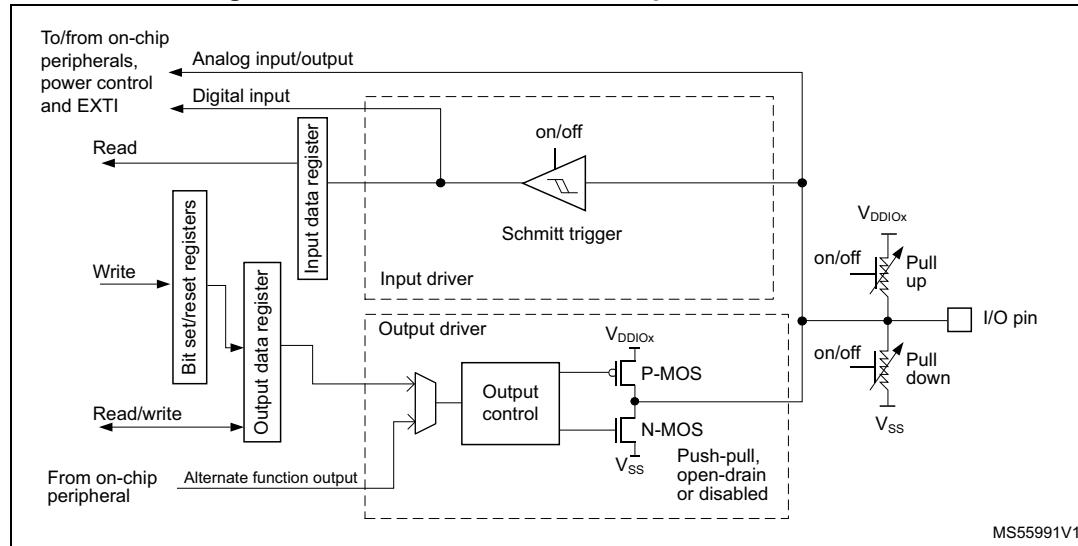
Subject to the specific hardware characteristics of each I/O port listed in the datasheet, each port bit of the general-purpose I/O (GPIO) ports can be individually configured by software in several modes:

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

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

*Figure 16* shows the basic structures of a standard I/O port bit. *Table 24* gives the possible port bit configurations.

**Figure 16. Basic structure of an I/O port bit**



**Table 24. Port bit configuration table<sup>(1)</sup>**

MODER(i) [1:0]	OTYPER(i)	OSPEEDR(i) [1:0]	PUPDR(i) [1:0]		I/O configuration	
01	0	SPEED [1:0]	0	0	GP output	PP
	0		0	1	GP output	PP + PU
	0		1	0	GP output	PP + PD
	0		1	1	Reserved	
	1		0	0	GP output	OD
	1		0	1	GP output	OD + PU
	1		1	0	GP output	OD + PD
	1		1	1	Reserved (GP output OD)	
10	0	SPEED [1:0]	0	0	AF	PP
	0		0	1	AF	PP + PU
	0		1	0	AF	PP + PD
	0		1	1	Reserved	
	1		0	0	AF	OD
	1		0	1	AF	OD + PU
	1		1	0	AF	OD + PD
	1		1	1	Reserved	

**Table 24. Port bit configuration table<sup>(1)</sup> (continued)**

<b>MODER(i) [1:0]</b>	<b>OTYPER(i)</b>	<b>OSPEEDR(i) [1:0]</b>		<b>PUPDR(i) [1:0]</b>		<b>I/O configuration</b>	
00	x	x	x	0	0	Input	Floating
	x	x	x	0	1	Input	PU
	x	x	x	1	0	Input	PD
	x	x	x	1	1	Reserved (input floating)	
11	x	x	x	0	0	Input/output	Analog
	x	x	x	0	1	Reserved	
	x	x	x	1	0		
	x	x	x	1	1	Reserved	

1. GP = general-purpose, PP = push-pull, PU = pull-up, PD = pull-down, OD = open-drain, AF = alternate function.

### 8.3.1 General-purpose I/O (GPIO)

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

The debug pins are in AF pull-up/pull-down after reset:

- PA14: SWCLK in pull-down
- PA13: SWDIO in pull-up

When the pin is configured as output, the value written to the output data register (GPIOx\_ODR) is output on the I/O pin. It is possible to use the output driver in push-pull mode or open-drain mode (only the low level is driven, high level is HI-Z).

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

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

### 8.3.2 I/O pin alternate function multiplexer and mapping

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

Each I/O pin has a multiplexer with up to eight alternate function inputs (AF0 to AF7) that can be configured through the GPIOx\_AFRL (for pin 0 to 7) and GPIOx\_AFRH (for pin 8 to 15) registers:

- After reset the multiplexer selection is alternate function 0 (AF0). The I/Os are configured in alternate function mode through GPIOx\_MODER register.
- The specific alternate function assignments for each pin are detailed in the device datasheet.

In addition to this flexible I/O multiplexing architecture, each peripheral has alternate functions mapped onto different I/O pins to optimize the number of peripherals available in smaller packages.

To use an I/O in a given configuration, the user has to proceed as follows:

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

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

### 8.3.3 I/O port control registers

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

### 8.3.4 I/O port data registers

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

See [Section 8.4.5: GPIO port input data register \(GPIOx\\_IDR\) \(x = A to F\)](#) and [Section 8.4.6: GPIO port output data register \(GPIOx\\_ODR\) \(x = A to F\)](#) for the register descriptions.

### 8.3.5 I/O data bitwise handling

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

To each bit in GPIOx\_ODR, correspond two control bits in GPIOx\_BSRR: BS(i) and BR(i). When written to 1, bit BS(i) **sets** the corresponding ODR(i) bit. When written to 1, bit BR(i) **resets** the ODR(i) corresponding bit.

Writing any bit to 0 in GPIOx\_BSRR does not have any effect on the corresponding bit in GPIOx\_ODR. If there is an attempt to both set and reset a bit in GPIOx\_BSRR, the set action takes priority.

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

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

### 8.3.6 GPIO locking mechanism

It is possible to freeze the port A and B GPIO control registers by applying a specific write sequence to the GPIOx\_LCKR register. The frozen registers are GPIOx\_MODER, GPIOx\_OTYPER, GPIOx\_OSPEEDR, GPIOx\_PUPDR, GPIOx\_AFRL and GPIOx\_AFRH.

To write the GPIOx\_LCKR register, a specific write / read sequence has to be applied. When the right LOCK sequence is applied to bit 16 in this register, the value of LCKR[15:0] is used to lock the configuration of the I/Os (during the write sequence the LCKR[15:0] value must be the same). When the LOCK sequence has been applied to a port bit, the value of the port bit can no longer be modified until the next MCU reset or peripheral reset. Each GPIOx\_LCKR bit freezes the corresponding bit in the control registers (GPIOx\_MODER, GPIOx\_OTYPER, GPIOx\_OSPEEDR, GPIOx\_PUPDR, GPIOx\_AFRL and GPIOx\_AFRH).

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

For more details refer to LCKR register description in [Section 8.4.8: GPIO port configuration lock register \(GPIOx\\_LCKR\) \(x = A to B\)](#).

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

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

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

For code example refer to [Section A.4.2: Alternate function selection sequence code example on page 945](#).

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

### 8.3.8 External interrupt/wake-up lines

All ports have external interrupt capability. To use external interrupt lines, the given pin must not be configured in analog mode or being used as oscillator pin, so the input trigger is kept enabled.

Refer to [Section 11.2: Extended interrupts and events controller \(EXTI\)](#) and to [Section 11.2.3: Event management](#).

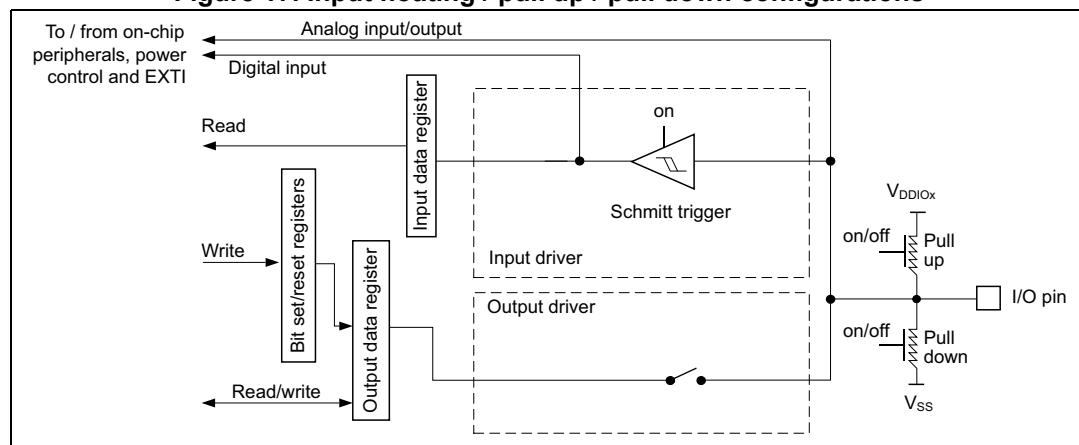
### 8.3.9 Input configuration

When the I/O port is programmed as input:

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

[Figure 17](#) shows the input configuration of the I/O port bit.

**Figure 17. Input floating / pull up / pull down configurations**



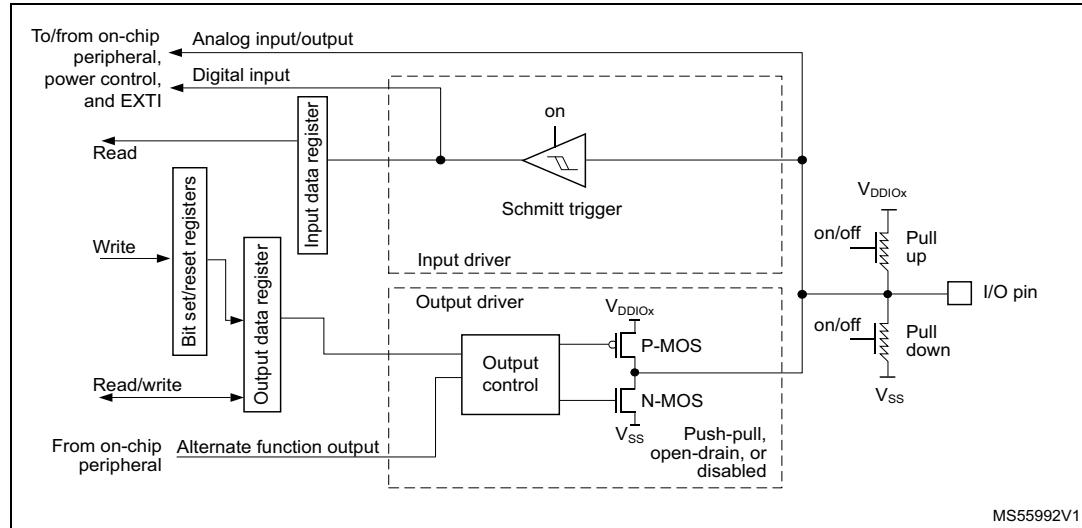
### 8.3.10 Output configuration

When the I/O port is programmed as output:

- The output buffer is enabled:
  - Open drain mode: a “0” in the output register activates the N-MOS whereas a “1” in the output register leaves the port in Hi-Z (the P-MOS is never activated)
  - Push-pull mode: a “0” in the output register activates the N-MOS whereas a “1” in the output register activates the P-MOS
- The Schmitt trigger input is activated
- The pull-up and pull-down resistors are activated depending on the value in the GPIOx\_PUPDR register
- The data present on the I/O pin are sampled into the input data register every AHB clock cycle
- A read access to the input data register gets the I/O state
- A read access to the output data register gets the last written value

*Figure 18* shows the output configuration of the I/O port bit.

**Figure 18. Output configuration**



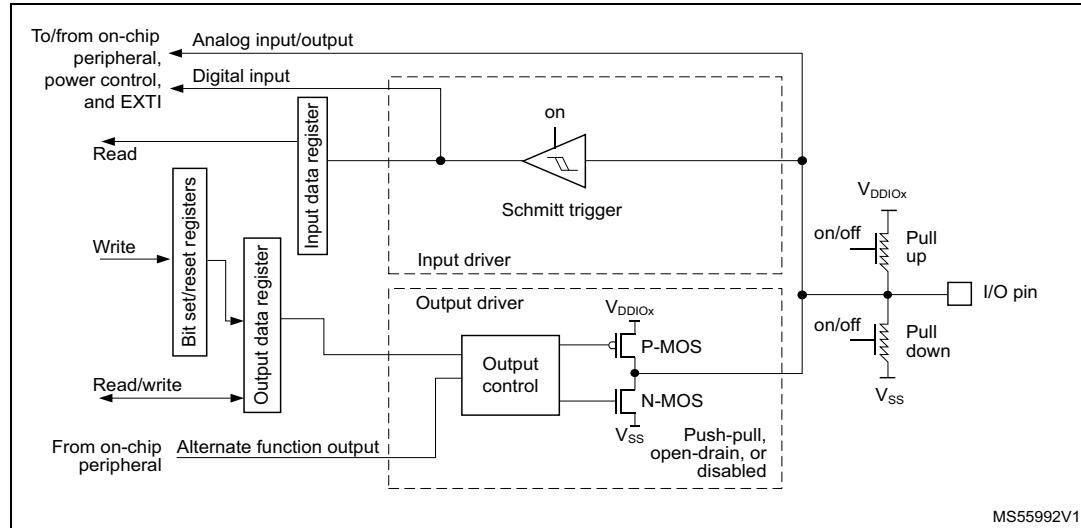
### 8.3.11 Alternate function configuration

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

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

*Figure 19* shows the alternate function configuration of the I/O port bit.

**Figure 19. Alternate function configuration**



### 8.3.12 Analog configuration

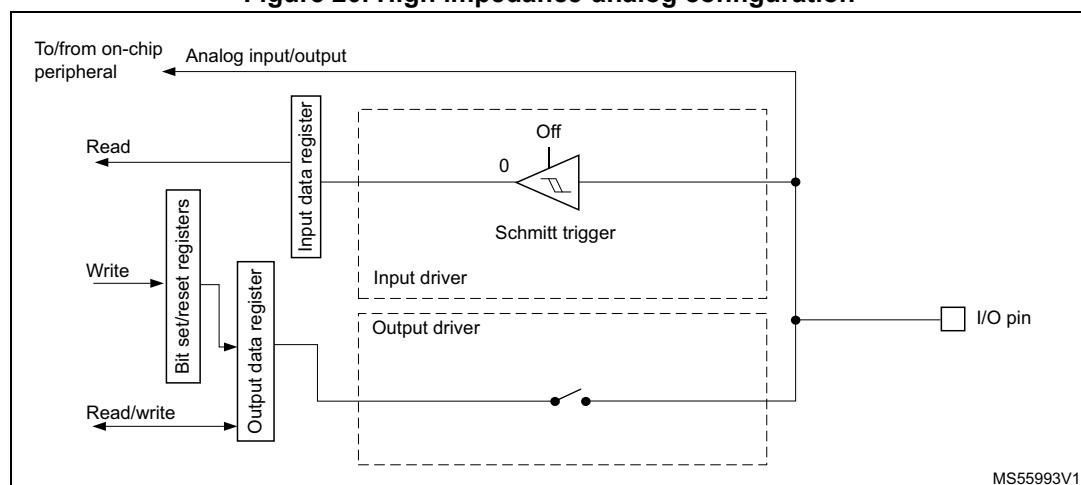
When the I/O port is programmed as analog configuration:

- The output buffer is disabled
- The Schmitt trigger input is deactivated, providing zero consumption for every analog value of the I/O pin. The output of the Schmitt trigger is forced to a constant value (0).
- The weak pull-up and pull-down resistors are disabled by hardware
- Read access to the input data register gets the value “0”

For code example refer to [Section A.4.3: Analog GPIO configuration code example on page 946](#).

*Figure 20* shows the high-impedance, analog-input configuration of the I/O port bits.

**Figure 20. High impedance-analog configuration**



### 8.3.13 Using the HSE or LSE oscillator pins as GPIOs

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

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

When the oscillator is configured in a user external clock mode, only the `OSC_IN` pin is reserved for clock input and the OSC\_OUT or OSC32\_OUT pin can still be used as normal GPIO.

### 8.3.14 Using the GPIO pins in the RTC supply domain

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

For details about I/O control by the RTC, refer to [Section 25.4: RTC functional description](#).

## 8.4 GPIO registers

For a summary of register bits, register address offsets and reset values, refer to [Table 25](#).

The peripheral registers can be written in word, half word or byte mode.

### 8.4.1 GPIO port mode register (GPIOx\_MODER) (x = A to F)

Address offset: 0x00

Reset value: 0x2800 0000 for port A

Reset value: 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]	MODER14[1:0]	MODER13[1:0]	MODER12[1:0]	MODER11[1:0]	MODER10[1:0]	MODER9[1:0]	MODER8[1:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]	MODER6[1:0]	MODER5[1:0]	MODER4[1:0]	MODER3[1:0]	MODER2[1:0]	MODER1[1:0]	MODERO[1:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **MODER[15:0][1:0]**: Port x configuration I/O pin y (y = 15 to 0)

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

00: Input mode (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

### 8.4.2 GPIO port output type register (GPIOx\_OTYPER) (x = A to F)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw															

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

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

These bits are written by software to configure the I/O output type.

0: Output push-pull (reset state)

1: Output open-drain

### 8.4.3 GPIO port output speed register (GPIOx\_OSPEEDR) (x = A to F)

Address offset: 0x08

Reset value: 0x0C00 0000 (for port A)

Reset value: 0x0000 0000 (for other ports)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [1:0]	
rw	rw	rw	rw	rw	rw										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSPEEDR7 [1:0]		OSPEEDR6 [1:0]		OSPEEDR5 [1:0]		OSPEEDR4 [1:0]		OSPEEDR3 [1:0]		OSPEEDR2 [1:0]		OSPEEDR1 [1:0]		OSPEEDR0 [1:0]	
rw	rw	rw	rw	rw	rw										

Bits 31:0 **OSPEEDR[15:0][1:0]**: Port x configuration I/O pin y (y = 15 to 0)

These bits are written by software to configure the I/O output speed.

x0: Low speed

01: Medium speed

11: High speed

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

### 8.4.4 GPIO port pull-up/pull-down register (GPIOx\_PUPDR) (x = A to F)

Address offset: 0x0C

Reset value: 0x2400 0000 (for port A)

Reset value: 0x0000 0000 (for other ports)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw										

Bits 31:0 **PUPDR[15:0][1:0]**: Port x configuration I/O pin y (y = 15 to 0)

These bits are written by software to configure the I/O pull-up or pull-down

00: No pull-up, pull-down

01: Pull-up

10: Pull-down

11: Reserved

### 8.4.5 GPIO port input data register (GPIO<sub>x</sub>\_IDR) (x = A to F)

Address offset: 0x10

Reset value: 0x0000 XXXX

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

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

Bits 15:0 **IDR[15:0]**: Port x input data I/O pin y (y = 15 to 0)

These bits are read-only. They contain the input value of the corresponding I/O port.

### 8.4.6 GPIO port output data register (GPIO<sub>x</sub>\_ODR) (x = A to F)

Address offset: 0x14

Reset value: 0x0000 0000

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

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

Bits 15:0 **ODR[15:0]**: Port output data I/O pin y (y = 15 to 0)

These bits can be read and written by software.

*Note: For atomic bit set/reset, the ODR bits can be individually set and/or reset by writing to the GPIO<sub>x</sub>\_BSRR register (x = A..F).*

### 8.4.7 GPIO port bit set/reset register (GPIOx\_BSRR) (x = A to F)

Address offset: 0x18

Reset value: 0x0000 0000

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

Bits 31:16 **BR[15:0]**: Port x reset I/O pin y (y = 15 to 0)

These bits are write-only. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Resets the corresponding ODRx bit

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

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

These bits are write-only. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Sets the corresponding ODRx bit

### 8.4.8 GPIO port configuration lock register (GPIOx\_LCKR) (x = A to B)

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

*Note: A specific write sequence is used to write to the GPIOx\_LCKR register. Only word access (32-bit long) is allowed during this locking sequence.*

Each lock bit freezes a specific configuration register (control and alternate function registers).

Address offset: 0x1C

Reset value: 0x0000 0000

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

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

#### Bit 16 **LCKK:** Lock key

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

0: Port configuration lock key not active

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

LOCK key write sequence:

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

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

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

RD LCKR

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

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

*Any error in the lock sequence aborts the lock.*

*After the first lock sequence on any bit of the port, any read access on the LCKK bit returns 1 until the next MCU reset or peripheral reset. For code example refer to Section A.4.1: Lock sequence code example on page 945.*

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

These bits are read/write but can only be written when the LCKK bit is 0.

0: Port configuration not locked

1: Port configuration locked

### 8.4.9 GPIO alternate function low register (GPIOx\_AFRL) (x = A to F)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFSEL7[3:0]				AFSEL6[3:0]				AFSEL5[3:0]				AFSEL4[3:0]			
rw	rw	rw	rw												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFSEL3[3:0]				AFSEL2[3:0]				AFSEL1[3:0]				AFSEL0[3:0]			
rw	rw	rw	rw												

#### Bits 31:0 **AFSELy[3:0]:** Alternate function selection for port x pin y (y = 0..7)

These bits are written by software to configure alternate function I/Os

AFSELy selection:

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

### 8.4.10 GPIO alternate function high register (GPIOx\_AFRH) (x = A to F)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFSEL15[3:0]				AFSEL14[3:0]				AFSEL13[3:0]				AFSEL12[3:0]			
rw	rw	rw	rw												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFSEL11[3:0]				AFSEL10[3:0]				AFSEL9[3:0]				AFSEL8[3:0]			
rw	rw	rw	rw												

Bits 31:0 **AFSELy[3:0]**: Alternate function selection for port x pin y (y = 8..15)

These bits are written by software to configure alternate function I/Os

AFSELy selection:

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

### 8.4.11 GPIO port bit reset register (GPIOx\_BRR) (x = A to F)

Address offset: 0x28

Reset value: 0x0000 0000

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

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

Bits 15:0 **BR[15:0]**: Port x reset IO pin y (y = 15 to 0)

These bits are write-only. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODx bit

1: Reset the corresponding ODx bit

## 8.4.12 GPIO register map

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

**Table 25. GPIO register map and reset values**

Offset	Register name	Reset value	0x00	GPIOA_MODER	31
0x00	GPIOx_MODER (where x = B..F)	0 OSPEEDR15[1:0] 0 OSPEEDR15[1:0]	0x00	0 MODER15[1:0] 0 MODER15[1:0]	30
0x00	GPIOx_MODER (where x = B..F)	0 OSPEEDR14[1:0] 0 OSPEEDR14[1:0]	0x00	0 MODER14[1:0] 0 MODER14[1:0]	29
0x04	GPIOx_OTYPER (where x = A..F)	0 PUPDR13[1:0] 0 PUPDR13[1:0]	0x00	0 MODER13[1:0] 0 MODER13[1:0]	28
0x04	GPIOx_OTYPER (where x = A..F)	0 PUPDR12[1:0] 0 PUPDR12[1:0]	0x00	0 MODER12[1:0] 0 MODER12[1:0]	27
0x08	GPIOA_OSPEEDR	0 PUPDR11[1:0] 0 PUPDR11[1:0]	0x00	0 MODER11[1:0] 0 MODER11[1:0]	26
0x08	GPIOx_OSPEEDR (where x = B..F)	0 PUPDR10[1:0] 0 PUPDR10[1:0]	0x00	0 MODER10[1:0] 0 MODER10[1:0]	25
0x08	GPIOx_OSPEEDR (where x = B..F)	0 PUPDR9[1:0] 0 PUPDR9[1:0]	0x00	0 MODER9[1:0] 0 MODER9[1:0]	24
0x0C	GPIOA_PUPDR	0 PUPDR8[1:0] 0 PUPDR8[1:0]	0x00	0 MODER8[1:0] 0 MODER8[1:0]	20
0x0C	GPIOx_PUPDR (where x = B..F)	0 PUPDR7[1:0] 0 PUPDR7[1:0]	0x00	0 MODER7[1:0] 0 MODER7[1:0]	19
0x10	GPIOx_PUPDR (where x = B..F)	0 PUPDR6[1:0] 0 PUPDR6[1:0]	0x00	0 MODER6[1:0] 0 MODER6[1:0]	18
0x10	GPIOx_IDR (where x = A..F)	0 PUPDR5[1:0] 0 PUPDR5[1:0]	0x00	0 MODER5[1:0] 0 MODER5[1:0]	17
0x10	GPIOx_IDR (where x = A..F)	0 PUPDR4[1:0] 0 PUPDR4[1:0]	0x00	0 MODER4[1:0] 0 MODER4[1:0]	16
0x14	GPIOx_ODR (where x = A..F)	0 PUPDR3[1:0] 0 PUPDR3[1:0]	0x00	0 MODER3[1:0] 0 MODER3[1:0]	15
0x14	GPIOx_ODR (where x = A..F)	0 PUPDR2[1:0] 0 PUPDR2[1:0]	0x00	0 MODER2[1:0] 0 MODER2[1:0]	14
0x18	GPIOx_BSRR (where x = A..F)	0 PUPDR1[1:0] 0 PUPDR1[1:0]	0x00	0 MODER1[1:0] 0 MODER1[1:0]	13
0x18	GPIOx_BSRR (where x = A..F)	0 PUPDR0[1:0] 0 PUPDR0[1:0]	0x00	0 MODER0[1:0] 0 MODER0[1:0]	12

**Table 25. GPIO register map and reset values (continued)**

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0											
0x1C	<b>GPIOx_LCKR</b> (where x = A..B)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	LCKK	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	Reset value																																											
0x20	<b>GPIOx_AFRL</b> (where x = A..F)	AFSEL7 [3:0]	AFSEL6 [3:0]	AFSEL5 [3:0]	AFSEL4 [3:0]	AFSEL3 [3:0]	AFSEL2 [3:0]	AFSEL1 [3:0]	AFSEL0 [3:0]								BR15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
	Reset value	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0								BR14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x24	<b>GPIOx_AFRH</b> (where x = A..F)	AFSEL15 [3:0]	AFSEL14 [3:0]	AFSEL13 [3:0]	AFSEL12 [3:0]	AFSEL11 [3:0]	AFSEL10 [3:0]	AFSEL9 [3:0]	AFSEL8 [3:0]								BR13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	Reset value	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0								BR12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x28	<b>GPIOx_BRR</b> (where x = A..F)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	BR9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Reset value																BR8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 9 System configuration controller (SYSCFG)

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

- Enabling/disabling I<sup>2</sup>C Fast Mode Plus on some IO ports
- Remapping some DMA trigger sources to different DMA channels
- Remapping the memory located at the beginning of the code area
- Pending interrupt status registers for each interrupt line on STM32F09x devices
- Managing the external interrupt line connection to the GPIOs
- Managing robustness feature

### 9.1 SYSCFG registers

#### 9.1.1 SYSCFG configuration register 1 (SYSCFG\_CFGR1)

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

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

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

Address offset: 0x00

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

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	TIM3_DMA_RMP	TIM2_DMA_RMP	TIM1_DMA_RMP	I2C1_DMA_RMP	USART3_DMA_RMP	USART2_DMA_RMP	SPI2_DMA_RMP	I2C_PA10_FMP	I2C_PA9_FMP	I2C2_FMP	I2C1_FMP	I2C_PB9_FMP	I2C_PB8_FMP	I2C_PB7_FMP	I2C_PB6_FMP	
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TIM17_DMA_RMP_2	TIM16_DMA_RMP_2	TIM17_DMA_RMP	TIM16_DMA_RMP	USART1_RX_DMA_RMP	USART1_TX_DMA_RMP	ADC_DMA_RMP	IR_MOD [1:0]	Res.	PA11_PA12_RMP	Res.	Res.	MEM_MODE [1:0]			
	rw	rw	rw	rw	rw	rw	rw	rw		rw			rw	rw	rw	rw

Bit 31 Reserved, must be kept at reset value.

Bit 30 **TIM3\_DMA\_RMP**: TIM3 DMA request remapping bit. Available on STM32F07x devices only.

This bit is set and cleared by software. It controls the remapping of TIM3 DMA requests.

0: No remap (TIM3\_CH1 and TIM3\_TRIG DMA requests mapped on DMA channel 4)

1: Remap (TIM3\_CH1 and TIM3\_TRIG DMA requests mapped on DMA channel 6)

- Bit 29 **TIM2\_DMA\_RMP:** TIM2 DMA request remapping bit. Available on STM32F07x devices only.  
 This bit is set and cleared by software. It controls the remapping of TIM2 DMA requests.  
 0: No remap (TIM2\_CH2 and TIM2\_CH4 DMA requests mapped on DMA channel 3 and 4 respectively)  
 1: Remap (TIM2\_CH2 and TIM2\_CH4 DMA requests mapped on DMA channel 7)
- Bit 28 **TIM1\_DMA\_RMP:** TIM1 DMA request remapping bit. Available on STM32F07x devices only.  
 This bit is set and cleared by software. It controls the remapping of TIM1 DMA requests.  
 0: No remap (TIM1\_CH1, TIM1\_CH2 and TIM1\_CH3 DMA requests mapped on DMA channel 2, 3 and 4 respectively)  
 1: Remap (TIM1\_CH1, TIM1\_CH2 and TIM1\_CH3 DMA requests mapped on DMA channel 6)
- Bit 27 **I2C1\_DMA\_RMP:** I2C1 DMA request remapping bit. Available on STM32F07x devices only.  
 This bit is set and cleared by software. It controls the remapping of I2C1 DMA requests.  
 0: No remap (I2C1\_RX and I2C1\_TX DMA requests mapped on DMA channel 3 and 2 respectively)  
 1: Remap (I2C1\_RX and I2C1\_TX DMA requests mapped on DMA channel 7 and 6 respectively)
- Bit 26 **USART3\_DMA\_RMP:** USART3 DMA request remapping bit. Available on STM32F07x devices only.  
 This bit is set and cleared by software. It controls the remapping of USART3 DMA requests.  
 0: (USART3\_RX and USART3\_TX DMA requests mapped on DMA channel 6 and 7 respectively)  
 1: Remap (USART3\_RX and USART3\_TX DMA requests mapped on DMA channel 3 and 2 respectively)
- Bit 25 **USART2\_DMA\_RMP:** USART2 DMA request remapping bit. Available on STM32F07x devices only.  
 This bit is set and cleared by software. It controls the remapping of USART2 DMA requests.  
 0: No remap (USART2\_RX and USART2\_TX DMA requests mapped on DMA channel 5 and 4 respectively)  
 1: Remap (USART2\_RX and USART2\_TX DMA requests mapped on DMA channel 6 and 7 respectively)
- Bit 24 **SPI2\_DMA\_RMP:** SPI2 DMA request remapping bit. Available on STM32F07x devices only.  
 This bit is set and cleared by software. It controls the remapping of SPI2 DMA requests.  
 0: No remap (SPI2\_RX and SPI2\_TX DMA requests mapped on DMA channel 4 and 5 respectively)  
 1: Remap (SPI2\_RX and SPI2\_TX DMA requests mapped on DMA channel 6 and 7 respectively)
- Bits 23:22 **I2C\_PAx\_FMP:** Fast Mode Plus (FM+) driving capability activation bits. Available on STM32F03x, STM32F04x and STM32F09x devices only.  
 These bits are set and cleared by software. Each bit enables I<sup>2</sup>C FM+ mode for PA10 and PA9 I/Os.  
 0: PAx pin operates in standard mode.  
 1: I<sup>2</sup>C FM+ mode enabled on PAx pin and the Speed control is bypassed.

- Bit 21 **I2C2\_FMP**: FM+ driving capability activation for I2C2. Available on STM32F07x and STM32F09x devices only.  
 This bit is set and cleared by software. This bit is OR-ed with I2C\_Pxx\_FM+ bits.  
 0: FM+ mode is controlled by I2C\_Pxx\_FM+ bits only.  
 1: FM+ mode is enabled on all I2C2 pins selected through selection bits in GPIOx\_AFR registers. This is the only way to enable the FM+ mode for pads without a dedicated I2C\_Pxx\_FM+ control bit.
- Bit 20 **I2C1\_FMP**: FM+ driving capability activation for I2C1. Not available on STM32F05x devices.  
 This bit is set and cleared by software. This bit is OR-ed with I2C\_Pxx\_FM+ bits.  
 0: FM+ mode is controlled by I2C\_Pxx\_FM+ bits only.  
 1: FM+ mode is enabled on all I2C1 pins selected through selection bits in GPIOx\_AFR registers. This is the only way to enable the FM+ mode for pads without a dedicated I2C\_Pxx\_FM+ control bit.
- Bits 19:16 **I2C\_PBx\_FMP**: Fast Mode Plus (FM+) driving capability activation bits.  
 These bits are set and cleared by software. Each bit enables I<sup>2</sup>C FM+ mode for PB6, PB7, PB8, and PB9 I/Os.  
 0: PBx pin operates in standard mode.  
 1: I<sup>2</sup>C FM+ mode enabled on PBx pin and the Speed control is bypassed.
- Bit 15 Reserved, must be kept at reset value.
- Bit 14 **TIM17\_DMA\_RMP2**: TIM17 alternate DMA request remapping bit. Available on STM32F07x devices only.  
 This bit is set and cleared by software. It controls the alternate remapping of TIM17 DMA requests.  
 0: No alternate remap (TIM17 DMA requests mapped according to TIM17\_DMA\_RMP bit)  
 1: Alternate remap (TIM17\_CH1 and TIM17\_UP DMA requests mapped on DMA channel 7)
- Bit 13 **TIM16\_DMA\_RMP2**: TIM16 alternate DMA request remapping bit. Available on STM32F07x devices only.  
 This bit is set and cleared by software. It controls the alternate remapping of TIM16 DMA requests.  
 0: No alternate remap (TIM16 DMA requests mapped according to TIM16\_DMA\_RMP bit)  
 1: Alternate remap (TIM16\_CH1 and TIM16\_UP DMA requests mapped on DMA channel 6)
- Bit 12 **TIM17\_DMA\_RMP**: TIM17 DMA request remapping bit. Available on STM32F03x, STM32F04x, STM32F05x and STM32F07x devices only.  
 This bit is set and cleared by software. It controls the remapping of TIM17 DMA requests.  
 0: No remap (TIM17\_CH1 and TIM17\_UP DMA requests mapped on DMA channel 1)  
 1: Remap (TIM17\_CH1 and TIM17\_UP DMA requests mapped on DMA channel 2)
- Bit 11 **TIM16\_DMA\_RMP**: TIM16 DMA request remapping bit. Available on STM32F03x, STM32F04x, STM32F05x and STM32F07x devices only.  
 This bit is set and cleared by software. It controls the remapping of TIM16 DMA requests.  
 0: No remap (TIM16\_CH1 and TIM16\_UP DMA requests mapped on DMA channel 3)  
 1: Remap (TIM16\_CH1 and TIM16\_UP DMA requests mapped on DMA channel 4)
- Bit 10 **USART1\_RX\_DMA\_RMP**: USART1\_RX DMA request remapping bit. Available on STM32F03x, STM32F04x, STM32F05x and STM32F07x devices only.  
 This bit is set and cleared by software. It controls the remapping of USART1\_RX DMA requests.  
 0: No remap (USART1\_RX DMA request mapped on DMA channel 3)  
 1: Remap (USART1\_RX DMA request mapped on DMA channel 5)

- Bit 9 **USART1\_TX\_DMA\_RMP**: USART1\_TX DMA request remapping bit. Available on STM32F03x, STM32F04x, STM32F05x and STM32F07x devices only.  
This bit is set and cleared by software. It bit controls the remapping of USART1\_TX DMA requests.  
0: No remap (USART1\_TX DMA request mapped on DMA channel 2)  
1: Remap (USART1\_TX DMA request mapped on DMA channel 4)
- Bit 8 **ADC\_DMA\_RMP**: ADC DMA request remapping bit. Available on STM32F03x, STM32F04x, STM32F05x and STM32F07x devices only.  
This bit is set and cleared by software. It controls the remapping of ADC DMA requests.  
0: No remap (ADC DMA request mapped on DMA channel 1)  
1: Remap (ADC DMA request mapped on DMA channel 2)
- Bits 7:6 **IR\_MOD[1:0]**: IR Modulation Envelope signal selection. Available on STM32F09x devices only.  
Those bits allow to select the modulation envelope signal between TIM16, USART1 and USART4:  
00: TIM16 selected  
01: USART1 selected  
10: USART4 selected  
11: Reserved
- Bit 5 Reserved, must be kept at reset value.
- Bit 4 **PA11\_PA12\_RMP**: PA11 and PA12 remapping bit for small packages (28 and 20 pins). Available on STM32F04x devices only.  
This bit is set and cleared by software. It controls the mapping of either PA9/10 or PA11/12 pin pair on small pin-count packages.  
0: No remap (pin pair PA9/10 mapped on the pins)  
1: Remap (pin pair PA11/12 mapped instead of PA9/10)
- Bits 3:2 Reserved, must be kept at reset value.
- Bits 1:0 **MEM\_MODE[1:0]**: Memory mapping selection bits  
These bits are set and cleared by software. They control the memory internal mapping at address 0x0000 0000. After reset these bits take on the value selected by the actual boot mode configuration. Refer to [Section 2.5: Boot configuration](#) for more details.  
x0: Main Flash memory mapped at 0x0000 0000  
01: System Flash memory mapped at 0x0000 0000  
11: Embedded SRAM mapped at 0x0000 0000

### 9.1.2 SYSCFG external interrupt configuration register 1 (SYSCFG\_EXTICR1)

Address offset: 0x08

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
rw	rw	rw	rw												

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

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

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

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

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

### 9.1.3 SYSCFG external interrupt configuration register 2 (SYSCFG\_EXTICR2)

Address offset: 0x0C

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI7[3:0]				EXTI6[3:0]				EXTI5[3:0]				EXTI4[3:0]			
rw	rw	rw	rw												

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

Bits 15:0 **EXTIx[3:0]**: EXTI x configuration bits (x = 4 to 7)

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

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

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

### 9.1.4 SYSCFG external interrupt configuration register 3 (SYSCFG\_EXTICR3)

Address offset: 0x10

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI11[3:0]				EXTI10[3:0]				EXTI9[3:0]				EXTI8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

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

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

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

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

### 9.1.5 SYSCFG external interrupt configuration register 4 (SYSCFG\_EXTICR4)

Address offset: 0x14

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI15[3:0]				EXTI14[3:0]				EXTI13[3:0]				EXTI12[3:0]			
rw	rw	rw	rw												

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

Bits 15:0 **EXTIx[3:0]**: EXTI x configuration bits (x = 12 to 15)

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

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

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

### 9.1.6 SYSCFG configuration register 2 (SYSCFG\_CFGR2)

Address offset: 0x18

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SRAM_PEF	Res.	Res.	Res.	Res.	Res.	PVD_LOCK	SRAM_PARITY_LOCK	LOCKUP_LOCK						
							rc_w1						rw	rw	rw

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

**Bit 8 SRAM\_PEF:** SRAM parity error flag

This bit is set by hardware when an SRAM parity error is detected. It is cleared by software by writing '1'.

0: No SRAM parity error detected

1: SRAM parity error detected

Bits 7:3 Reserved, must be kept at reset value

**Bit 2 PVD\_LOCK:** PVD lock enable bit

This bit is set by software and cleared by a system reset. It can be used to enable and lock the PVD connection to TIM1/15/16/17 Break input, as well as the PVDE and PLS[2:0] in the PWR\_CR register.

0: PVD interrupt disconnected from TIM1/15/16/17 Break input. PVDE and PLS[2:0] bits can be programmed by the application.

1: PVD interrupt connected to TIM1/15/16/17 Break input, PVDE and PLS[2:0] bits are read only.

**Bit 1 SRAM\_PARITY\_LOCK:** SRAM parity lock bit

This bit is set by software and cleared by a system reset. It can be used to enable and lock the SRAM parity error signal connection to TIM1/15/16/17 Break input.

0: SRAM parity error disconnected from TIM1/15/16/17 Break input

1: SRAM parity error connected to TIM1/15/16/17 Break input

**Bit 0 LOCKUP\_LOCK:** Cortex-M0 LOCKUP bit enable bit

This bit is set by software and cleared by a system reset. It can be used to enable and lock the connection of Cortex-M0 LOCKUP (Hardfault) output to TIM1/15/16/17 Break input.

0: Cortex-M0 LOCKUP output disconnected from TIM1/15/16/17 Break input

1: Cortex-M0 LOCKUP output connected to TIM1/15/16/17 Break input

### 9.1.7 SYSCFG interrupt line 0 status register (SYSCFG\_ITLINE0)

A dedicated set of registers is implemented on STM32F09x to collect all pending interrupt sources associated with each interrupt line into a single register. This allows users to check by single read which peripheral requires service in case more than one source is associated to the interrupt line.

All bits in those registers are read only, set by hardware when there is corresponding interrupt request pending and cleared by resetting the interrupt source flags in the peripheral registers.

Address offset: 80h

System reset value: 0x0000

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

Bits 31:1 Reserved (read as '0')

Bit 0 **WWDG**: Window watchdog interrupt pending flag

### 9.1.8 SYSCFG interrupt line 1 status register (SYSCFG\_ITLINE1)

Address offset: 84h

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	VDDIO2	PVDOUT													
														r	r

Bits 31:2 Reserved (read as '0')

Bit 1 **VDDIO2**: VDDIO2 supply monitoring interrupt request pending (EXTI line 31)

Bit 0 **PVDOUT**: PVD supply monitoring interrupt request pending (EXTI line 16). This bit is not available on STM32F0x8 devices.

### 9.1.9 SYSCFG interrupt line 2 status register (SYSCFG\_ITLINE2)

Address offset: 88h

System reset value: 0x0000

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

Bits 31:3 Reserved (read as '0')

Bit 2 **RTC\_ALRA**: RTC Alarm interrupt request pending (EXTI line 17)

Bit 1 **RTC\_TSTAMP**: RTC Tamper and TimeStamp interrupt request pending (EXTI line 19)

Bit 0 **RTC\_WAKEUP**: RTC Wake Up interrupt request pending (EXTI line 20)

### 9.1.10 SYSCFG interrupt line 3 status register (SYSCFG\_ITLINE3)

Address offset: 8Ch

System reset value: 0x0000

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

Bits 31:1 Reserved (read as '0')

Bit 0 **FLASH\_ITF**: Flash interface interrupt request pending

### 9.1.11 SYSCFG interrupt line 4 status register (SYSCFG\_ITLINE4)

Address offset: 90h

System reset value: 0x0000

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

Bits 31:2 Reserved (read as '0')

Bit 1 **CRS**: Clock recovery system interrupt request pending

Bit 0 **RCC**: Reset and clock control interrupt request pending

### 9.1.12 SYSCFG interrupt line 5 status register (SYSCFG\_ITLINE5)

Address offset: 94h

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	EXTI1 EXTI0														
															r r

Bits 31:2 Reserved (read as '0')

Bit 1 **EXTI1**: EXTI line 1 interrupt request pending

Bit 0 **EXTI0**: EXTI line 0 interrupt request pending

### 9.1.13 SYSCFG interrupt line 6 status register (SYSCFG\_ITLINE6)

Address offset: 98h

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	EXTI3	EXTI2													
														r	r

Bits 31:2 Reserved (read as '0')

Bit 1 **EXTI3**: EXTI line 3 interrupt request pending

Bit 0 **EXTI2**: EXTI line 2 interrupt request pending

### 9.1.14 SYSCFG interrupt line 7 status register (SYSCFG\_ITLINE7)

Address offset: 9Ch

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	EXTI15	EXTI14	EXTI13	EXTI12	EXTI11	EXTI10	EXTI9	EXTI8	EXTI7	EXTI6	EXTI5	EXTI4
				r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:10 Reserved (read as '0')

Bit 11 **EXTI15**: EXTI line 15 interrupt request pending

Bit 10 **EXTI14**: EXTI line 14 interrupt request pending

Bit 9 **EXTI13**: EXTI line 13 interrupt request pending

Bit 8 **EXTI12**: EXTI line 12 interrupt request pending

Bit 7 **EXTI11**: EXTI line 11 interrupt request pending

Bit 6 **EXTI10**: EXTI line 10 interrupt request pending

Bit 5 **EXTI9**: EXTI line 9 interrupt request pending

Bit 4 **EXTI8**: EXTI line 8 interrupt request pending

Bit 3 **EXTI7**: EXTI line 7 interrupt request pending

Bit 2 **EXTI6**: EXTI line 6 interrupt request pending

Bit 1 **EXTI5**: EXTI line 5 interrupt request pending

Bit 0 **EXTI4**: EXTI line 4 interrupt request pending

### 9.1.15 SYSCFG interrupt line 8 status register (SYSCFG\_ITLINE8)

Address offset: A0h

System reset value: 0x0000

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

Bits 31:2 Reserved (read as '0')

Bit 1 **TCS\_EOA**: Touch sensing controller end of acquisition interrupt request pending

Bit 0 **TCS\_MCE**: Touch sensing controller max count error interrupt request pending

### 9.1.16 SYSCFG interrupt line 9 status register (SYSCFG\_ITLINE9)

Address offset: A4h

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	DMA1_CH1														
															r

Bits 31:1 Reserved (read as '0')

Bit 0 **DMA1\_CH1**: DMA1 channel 1 interrupt request pending

### 9.1.17 SYSCFG interrupt line 10 status register (SYSCFG\_ITLINE10)

Address offset: A8h

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	DMA2_CH2	DMA2_CH1	DMA1_CH3	DMA1_CH2											
												r	r	r	r

Bits 31:4 Reserved (read as '0')

Bit 3 **DMA2\_CH2**: DMA2 channel 2 interrupt request pending

Bit 2 **DMA2\_CH1**: DMA2 channel 1 interrupt request pending

Bit 1 **DMA1\_CH3**: DMA1 channel 3 interrupt request pending

Bit 0 **DMA1\_CH2**: DMA1 channel 2 interrupt request pending

### 9.1.18 SYSCFG interrupt line 11 status register (SYSCFG\_ITLINE11)

Address offset: ACh

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	DMA2_CH5	DMA2_CH4	DMA2_CH3	DMA1_CH7	DMA1_CH6	DMA1_CH5	DMA1_CH4								
									r	r	r	r	r	r	r

Bits 31:7 Reserved (read as '0')

Bit 6 **DMA2\_CH5**: DMA2 channel 5 interrupt request pending

Bit 5 **DMA2\_CH4**: DMA2 channel 4 interrupt request pending

Bit 4 **DMA2\_CH3**: DMA2 channel 3 interrupt request pending

Bit 3 **DMA1\_CH7**: DMA1 channel 7 interrupt request pending

Bit 2 **DMA1\_CH6**: DMA1 channel 6 interrupt request pending

Bit 1 **DMA1\_CH5**: DMA1 channel 5 interrupt request pending

Bit 0 **DMA1\_CH4**: DMA1 channel 4 interrupt request pending

### 9.1.19 SYSCFG interrupt line 12 status register (SYSCFG\_ITLINE12)

Address offset: B0h

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.													
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	COMP2	COMP1	ADC												
													r	r	r

Bits 31:3 Reserved (read as '0')

Bit 2 **COMP2**: Comparator 2 interrupt request pending (EXTI line 22)

Bit 1 **COMP1**: Comparator 1 interrupt request pending (EXTI line 21)

Bit 0 **ADC**: ADC interrupt request pending

### 9.1.20 SYSCFG interrupt line 13 status register (SYSCFG\_ITLINE13)

Address offset: B4h

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TIM1_BRK	TIM1_UPD	TIM1_TRG	TIM1_CCU											
												r	r	r	r

Bits 31:4 Reserved (read as '0')

Bit 3 **TIM1\_BRK**: Timer 1 break interrupt request pending

Bit 2 **TIM1\_UPD**: Timer 1 update interrupt request pending

Bit 1 **TIM1\_TRG**: Timer 1 trigger interrupt request pending

Bit 0 **TIM1\_CCU**: Timer 1 commutation interrupt request pending

### 9.1.21 SYSCFG interrupt line 14 status register (SYSCFG\_ITLINE14)

Address offset: B8h

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TIM1_CC														
															r

Bits 31:1 Reserved (read as '0')

Bit 0 **TIM1\_CC**: Timer 1 capture compare interrupt request pending

### 9.1.22 SYSCFG interrupt line 15 status register (SYSCFG\_ITLINE15)

Address offset: BC<sub>h</sub>

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TIM2														
															r

Bits 31:1 Reserved (read as '0')

Bit 0 **TIM2**: Timer 2 interrupt request pending

### 9.1.23 SYSCFG interrupt line 16 status register (SYSCFG\_ITLINE16)

Address offset: C0h

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TIM3														
															r

Bits 31:1 Reserved (read as '0')

Bit 0 **TIM3**: Timer 3 interrupt request pending

### 9.1.24 SYSCFG interrupt line 17 status register (SYSCFG\_ITLINE17)

Address offset: C4h

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	DAC	TIM6													
														r	r

Bits 31:1 Reserved (read as '0')

Bit 1 **DAC**: DAC underrun interrupt request pending

Bit 0 **TIM6**: Timer 6 interrupt request pending

### 9.1.25 SYSCFG interrupt line 18 status register (SYSCFG\_ITLINE18)

Address offset: C8h

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TIM7														
															r

Bits 31:1 Reserved (read as '0')

Bit 0 **TIM7**: Timer 7 interrupt request pending

### 9.1.26 SYSCFG interrupt line 19 status register (SYSCFG\_ITLINE19)

Address offset: CCh

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TIM14														
															r

Bits 31:1 Reserved (read as '0')

Bit 0 **TIM14**: Timer 14 interrupt request pending

### 9.1.27 SYSCFG interrupt line 20 status register (SYSCFG\_ITLINE20)

Address offset: D0h

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TIM15														
															r

Bits 31:1 Reserved (read as '0')

Bit 0 **TIM15**: Timer 15 interrupt request pending

### 9.1.28 SYSCFG interrupt line 21 status register (SYSCFG\_ITLINE21)

Address offset: D4h

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TIM16														
															r

Bits 31:1 Reserved (read as '0')

Bit 0 **TIM16**: Timer 16 interrupt request pending

### 9.1.29 SYSCFG interrupt line 22 status register (SYSCFG\_ITLINE22)

Address offset: D8h

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TIM17														
															r

Bits 31:1 Reserved (read as '0')

Bit 0 **TIM17**: Timer 17 interrupt request pending

### 9.1.30 SYSCFG interrupt line 23 status register (SYSCFG\_ITLINE23)

Address offset: DCh

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	I2C1														
															r

Bits 31:1 Reserved (read as '0')

Bit 0 **I2C1**: I2C1 interrupt request pending, combined with EXTI line 23

### 9.1.31 SYSCFG interrupt line 24 status register (SYSCFG\_ITLINE24)

Address offset: E0h

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	I2C2														
															r

Bits 31:1 Reserved (read as '0')

Bit 0 **I2C2**: I2C2 interrupt request pending

### 9.1.32 SYSCFG interrupt line 25 status register (SYSCFG\_ITLINE25)

Address offset: E4h

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SPI1														
															r

Bits 31:1 Reserved (read as '0')

Bit 0 **SPI1**: SPI1 interrupt request pending

### 9.1.33 SYSCFG interrupt line 26 status register (SYSCFG\_ITLINE26)

Address offset: E8h

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SPI2														
															r

Bits 31:1 Reserved (read as '0')

Bit 0 **SPI2**: SPI2 interrupt request pending

### 9.1.34 SYSCFG interrupt line 27 status register (SYSCFG\_ITLINE27)

Address offset: EC<sub>h</sub>

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USART1														
															r

Bits 31:1 Reserved (read as '0')

Bit 0 **USART1**: USART1 interrupt request pending, combined with EXTI line 25

### 9.1.35 SYSCFG interrupt line 28 status register (SYSCFG\_ITLINE28)

Address offset: F0h

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USART2														
															r

Bits 31:1 Reserved (read as '0')

Bit 0 **USART2**: USART2 interrupt request pending, combined with EXTI line 26

### 9.1.36 SYSCFG interrupt line 29 status register (SYSCFG\_ITLINE29)

Address offset: F4h

System reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USART8	USART7	USART6	USART5	USART4										
										r	r	r	r	r	r

Bits 31:6 Reserved (read as '0')

Bit 5 **USART8**: USART8 interrupt request pending

Bit 4 **USART7**: USART7 interrupt request pending

Bit 3 **USART6**: USART6 interrupt request pending

Bit 2 **USART5**: USART5 interrupt request pending

Bit 1 **USART4**: USART4 interrupt request pending

Bit 0 **USART3**: USART3 interrupt request pending, combined with EXTI line 28.

### 9.1.37 SYSCFG interrupt line 30 status register (SYSCFG\_ITLINE30)

Address offset: F8h

System reset value: 0x0000

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

Bits 31:2 Reserved (read as '0')

Bit 1 **CAN**: CAN interrupt request pending

Bit 0 **CEC**: CEC interrupt request pending, combined with EXTI line 27

### **9.1.38 SYSCFG register maps**

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

**Table 26. SYSCFG register map and reset values**

**Table 27. SYSCFG register map and reset values for STM32F09x devices**

**Table 27. SYSCFG register map and reset values for STM32F09x devices (continued)**

**Table 27. SYSCFG register map and reset values for STM32F09x devices (continued)**

Offset	Register	31	30	29	28
0xC4	SYSCFG_ITLINE17	Res.	Res.	Res.	Res.
	Reset value	Res.	Res.	Res.	Res.
0xC8	SYSCFG_ITLINE18	Res.	Res.	Res.	Res.
	Reset value	Res.	Res.	Res.	Res.
0xCC	SYSCFG_ITLINE19	Res.	Res.	Res.	Res.
	Reset value	Res.	Res.	Res.	Res.
0xD0	SYSCFG_ITLINE20	Res.	Res.	Res.	Res.
	Reset value	Res.	Res.	Res.	Res.
0xD4	SYSCFG_ITLINE21	Res.	Res.	Res.	Res.
	Reset value	Res.	Res.	Res.	Res.
0xD8	SYSCFG_ITLINE22	Res.	Res.	Res.	Res.
	Reset value	Res.	Res.	Res.	Res.
0xDC	SYSCFG_ITLINE23	Res.	Res.	Res.	Res.
	Reset value	Res.	Res.	Res.	Res.
0xE0	SYSCFG_ITLINE24	Res.	Res.	Res.	Res.
	Reset value	Res.	Res.	Res.	Res.
0xE4	SYSCFG_ITLINE25	Res.	Res.	Res.	Res.
	Reset value	Res.	Res.	Res.	Res.
0xE8	SYSCFG_ITLINE26	Res.	Res.	Res.	Res.
	Reset value	Res.	Res.	Res.	Res.
0xEC	SYSCFG_ITLINE27	Res.	Res.	Res.	Res.
	Reset value	Res.	Res.	Res.	Res.
0xF0	SYSCFG_ITLINE28	Res.	Res.	Res.	Res.
	Reset value	Res.	Res.	Res.	Res.
0xF4	SYSCFG_ITLINE29	Res.	Res.	Res.	Res.
	Reset value	Res.	Res.	Res.	Res.
0xF8	SYSCFG_ITLINE30	Res.	Res.	Res.	Res.
	Reset value	Res.	Res.	Res.	Res.
CAN	Res.	Res.	Res.	Res.	DAC
CEC	USART3	USART2	USART1	SPI2	SPI1
				I2C1	I2C2
				TIM17	TIM16
				TIM15	TIM14
				TIM7	TIM6
				0	0

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 10 Direct memory access controller (DMA)

### 10.1 Introduction

The direct memory access (DMA) controller is a bus master and system peripheral.

The DMA is used to perform programmable data transfers between memory-mapped peripherals and/or memories, upon the control of an off-loaded CPU.

The DMA controller features a single AHB master architecture.

There is one instance of DMA with up to 7 channels, except for the STM32F09x devices that also include a DMA2 with 5 channels.

Each channel is dedicated to managing memory access requests from one or more peripherals. Each DMA includes an arbiter for handling the priority between DMA requests.

### 10.2 DMA main features

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

## 10.3 DMA implementation

### 10.3.1 DMA1 and DMA2

DMA1 and DMA2 are implemented with the hardware configuration parameters shown in the table below.

**Table 28. DMA1 and DMA2 implementation**

Feature	DMA1	DMA2 (only for STM32F09x devices)
Number of channels	Up to 7 <sup>(1)</sup>	5

1. 5 channels for STM32F03x/04x/05x and 7 channels for STM32F07x.

### 10.3.2 DMA request mapping

#### DMA controller on STM32F03x/04x/05x/07x

The hardware requests from the peripherals (TIMx, ADC, DAC, SPI, I2C, and USARTx) are simply logically ORed before entering the DMA. This means that on one channel, only one request must be enabled at a time.

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

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

*Table 29* and *Table 30* list the DMA requests for each channel.

**Table 29. DMA requests for each channel on STM32F03x/04x/05x devices**

Peripheral	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5
ADC	ADC <sup>(1)</sup>	ADC <sup>(2)</sup>	-	-	-
SPI	-	SPI1_RX	SPI1_TX	SPI2_RX	SPI2_TX
USART	-	USART1_TX <sup>(1)</sup>	USART1_RX <sup>(1)</sup>	USART1_TX <sup>(2)</sup> USART2_TX	USART1_RX <sup>(2)</sup> USART2_RX
I2C	-	I2C1_TX	I2C1_RX	I2C2_TX	I2C2_RX
TIM1	-	TIM1_CH1	TIM1_CH2	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_CH3 TIM1_UP
TIM2	TIM2_CH3	TIM2_UP	TIM2_CH2	TIM2_CH4	TIM2_CH1
TIM3	-	TIM3_CH3	TIM3_CH4 TIM3_UP	TIM3_CH1 TIM3_TRIG	-
TIM6 / DAC	-	-	TIM6_UP DAC_Channel1	-	-
TIM15	-	-	-	-	TIM15_CH1 TIM15_UP TIM15_TRIG TIM15_COM

**Table 29. DMA requests for each channel on STM32F03x/04x/05x devices (continued)**

Peripheral	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5
<b>TIM16</b>	-	-	TIM16_CH1 <sup>(1)</sup> TIM16_UP <sup>(1)</sup>	TIM16_CH1 <sup>(2)</sup> TIM16_UP <sup>(2)</sup>	-
<b>TIM17</b>	TIM17_CH1 <sup>(1)</sup> TIM17_UP <sup>(1)</sup>	TIM17_CH1 <sup>(2)</sup> TIM17_UP <sup>(2)</sup>	-	-	-

1. DMA request mapped on this DMA channel only if the corresponding remapping bit is cleared in [SYSCFG configuration register 1 \(SYSCFG\\_CFGR1\)](#).
2. DMA request mapped on this DMA channel only if the corresponding remapping bit is set in [SYSCFG configuration register 1 \(SYSCFG\\_CFGR1\)](#).

**Table 30. DMA requests for each channel on STM32F07x devices**

Peripheral	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
<b>ADC</b>	ADC <sup>(1)</sup>	ADC <sup>(2)</sup>	-	-	-	-	-
<b>SPI</b>	-	SPI1_RX	SPI1_TX	SPI2_RX <sup>(1)</sup>	SPI2_TX <sup>(1)</sup>	SPI2_RX <sup>(2)</sup>	SPI2_TX <sup>(2)</sup>
<b>USART</b>	-	USART1_TX <sup>(1)</sup> USART3_TX <sup>(2)</sup>	USART1_RX <sup>(1)</sup> USART3_RX <sup>(2)</sup>	USART1_TX <sup>(2)</sup> USART2_TX <sup>(1)</sup>	USART1_X <sup>(2)</sup> USART2_RX <sup>(1)</sup>	USART2_RX <sup>(2)</sup> USART3_RX <sup>(1)</sup> USART4_RX	USART2_TX <sup>(2)</sup> USART3_TX <sup>(1)</sup> USART4_TX
<b>I2C</b>	-	I2C1_TX <sup>(1)</sup>	I2C1_RX <sup>(1)</sup>	I2C2_TX	I2C2_RX	I2C1_TX <sup>(2)</sup>	I2C1_RX <sup>(2)</sup>
<b>TIM1</b>	-	TIM1_CH1 <sup>(1)</sup>	TIM1_CH2 <sup>(1)</sup>	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_CH3 <sup>(1)</sup> TIM1_UP	TIM1_CH1 <sup>(2)</sup> TIM1_CH2 <sup>(2)</sup> TIM1_CH3 <sup>(2)</sup>	-
<b>TIM2</b>	TIM2_CH3	TIM2_UP	TIM2_CH2 <sup>(1)</sup>	TIM2_CH4 <sup>(1)</sup>	TIM2_CH1	-	TIM2_CH2 <sup>(2)</sup> TIM2_CH4 <sup>(2)</sup>
<b>TIM3</b>	-	TIM3_CH3	TIM3_CH4 TIM3_UP	TIM3_CH1 <sup>(1)</sup> TIM3_TRIG <sup>(1)</sup>	-	TIM3_CH1 <sup>(2)</sup> TIM3_TRIG <sup>(2)</sup>	-
<b>TIM6 / DAC</b>	-	-	TIM6_UP DAC_Channel1	-	-	-	-
<b>TIM7 / DAC</b>	-	-	-	TIM7_UP DAC_Channel2	-	-	-
<b>TIM15</b>	-	-	-	-	TIM15_CH1 TIM15_UP TIM15_TRIG TIM15_COM	-	-
<b>TIM16</b>	-	-	TIM16_CH1 <sup>(1)</sup> TIM16_UP <sup>(1)</sup>	TIM16_CH1 <sup>(2)</sup> TIM16_UP <sup>(2)</sup>	-	TIM16_CH1 <sup>(3)</sup> TIM16_UP <sup>(3)</sup>	-
<b>TIM17</b>	TIM17_CH1 <sup>(1)</sup> TIM17_UP <sup>(1)</sup>	TIM17_CH1 <sup>(2)</sup> TIM17_UP <sup>(2)</sup>	-	-	-	-	TIM17_CH1 <sup>(3)</sup> TIM17_UP <sup>(3)</sup>

1. DMA request mapped on this DMA channel only if the corresponding remapping bit is cleared in [SYSCFG configuration register 1 \(SYSCFG\\_CFGR1\)](#).
2. DMA request mapped on this DMA channel only if the corresponding remapping bit is set in [SYSCFG configuration register 1 \(SYSCFG\\_CFGR1\)](#).
3. DMA request mapped on this DMA channel only if the additional RMP2 remapping bit is set in [SYSCFG configuration register 1 \(SYSCFG\\_CFGR1\)](#).

## DMA1 and DMA2 controllers on STM32F09x devices

The hardware requests from the peripherals are mapped to the DMA channels through the DMA\_CSELR channel selection register. On one channel, only one request must be enabled at a time. Refer to [Figure 21](#).

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

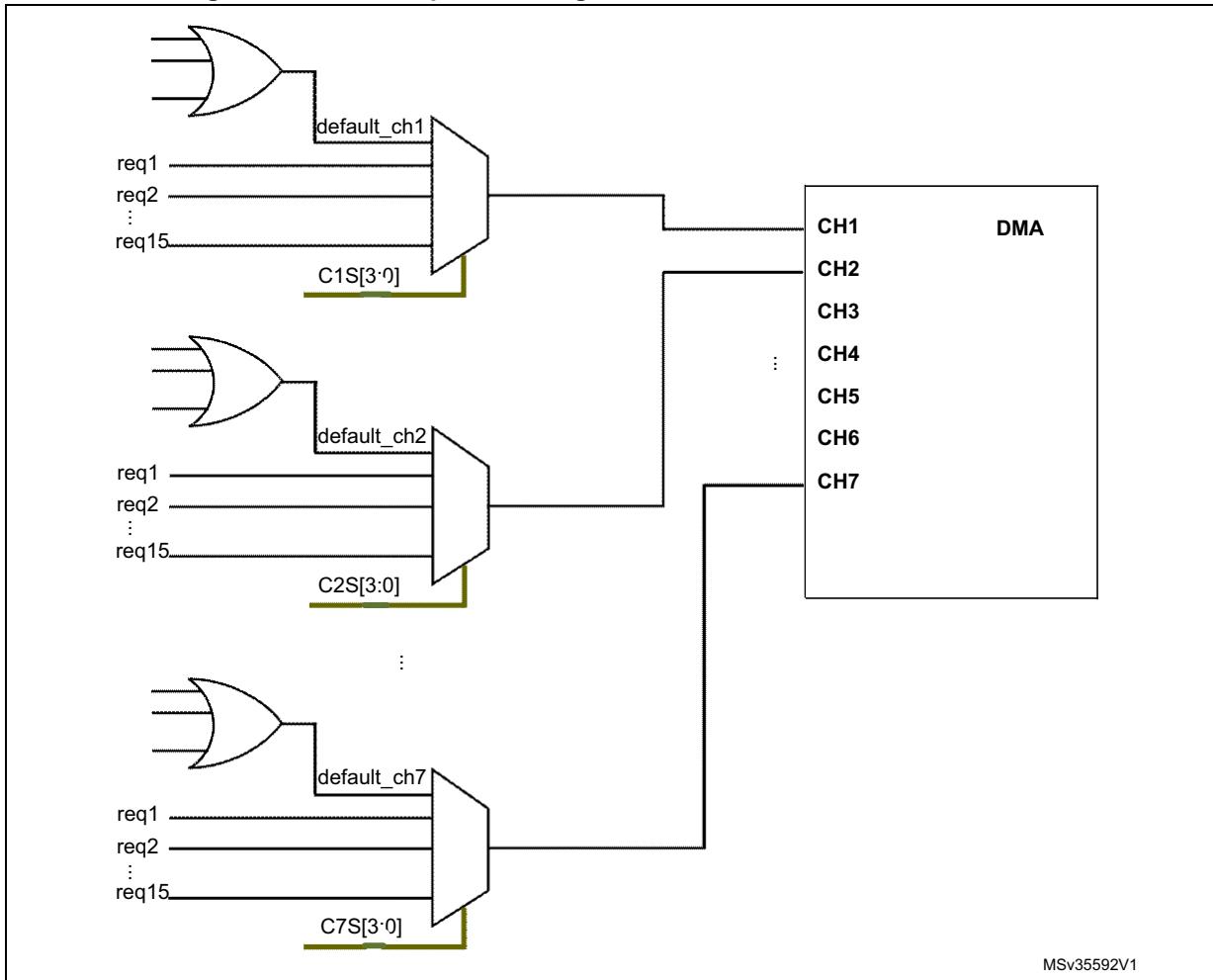
**Caution:** A same peripheral request can be assigned to two different channels only if the application ensures that these channels are not requested to be served at the same time. In other words, if two different channels receive a same asserted peripheral request at the same time, an unpredictable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

The default mapping position 0 ensures the compatibility with the DMA mapping used on other STM32F0xx products. The hardware requests from the peripherals (TIMx, ADC, DAC, SPI, I2C, and USARTx) are simply logically ORed before entering the DMA. This means that on one channel, only one request must be enabled at a time.

Alternate mapping positions 1 to 15 brings higher flexibility to map hardware requests on DMA channels. When alternate mapping position is used for some peripheral, the same request is removed from the default mapping position to avoid conflicts.

[Table 31](#) and [Table 32](#) list the DMA requests for each channel and alternate position.

Figure 21. DMAx request routing architecture on STM32F09x devices



1. Channels 6 and 7 are not available on DMA2.
2. Once a DMA request is selected on position 1 to 15, it disappears from the default location on position 0.

Table 31. DMA1 requests for each channel on STM32F09x devices

CxS [3:0]	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
0000	TIM2_CH3	TIM2_UP TIM3_CH3	TIM3_CH4 TIM3_UP	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	-	-
	-	-	-		TIM2_CH1	-	-
	-	-	-		TIM15_CH1 TIM15_UP TIM15_TRIG TIM15_COM	-	-
	ADC	-	TIM6_UP DAC_ Channel1	TIM7_UP DAC_ Channel2		-	-
	-	USART1_TX	USART1_RX	USART2_TX	USART2_RX	USART3_RX	USART3_TX
	-	-	-	-	-	USART4_RX	USART4_TX
	-	SPI1_RX	SPI1_TX	SPI2_RX	SPI2_TX	-	-
	-	I2C1_TX	I2C1_RX	I2C2_TX	I2C2_RX	-	-
	-	TIM1_CH1	TIM1_CH2	-	TIM1_CH3	-	-
	-	-	TIM2_CH2	TIM2_CH4	-	-	-
0001	TIM17_CH1 TIM17_UP	-	TIM16_CH1 TIM16_UP	TIM3_CH1 TIM3_TRIG	-	-	-
	ADC	ADC	TIM6_UP DAC_ Channel1	TIM7_UP DAC_ Channel2	-	-	-
0010	-	I2C1_TX	I2C1_RX	I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
0011	-	SPI1_RX	SPI1_TX	SPI2_RX	SPI2_TX	SPI2_RX	SPI2_TX
0100	-	TIM1_CH1	TIM1_CH2	-	TIM1_CH3	TIM1_CH1 TIM1_CH2 TIM1_CH3	-
0101	-	-	TIM2_CH2	TIM2_CH4	-	-	TIM2_CH2 TIM2_CH4
0110	-	-	-	TIM3_CH1 TIM3_TRIG	-	TIM3_CH1 TIM3_TRIG	-
0111	TIM17_CH1 TIM17_UP	TIM17_CH1 TIM17_UP	TIM16_CH1 TIM16_UP	TIM16_CH1 TIM16_UP	-	TIM16_CH1 TIM16_UP	TIM17_CH1 TIM17_UP
1000	USART1_RX	USART1_TX	USART1_RX	USART1_TX	USART1_RX	USART1_RX	USART1_TX
1001	USART2_RX	USART2_TX	USART2_RX	USART2_TX	USART2_RX	USART2_RX	USART2_TX
1010	USART3_RX	USART3_TX	USART3_RX	USART3_TX	USART3_RX	USART3_RX	USART3_TX
1011	USART4_RX	USART4_TX	USART4_RX	USART4_TX	USART4_RX	USART4_RX	USART4_TX
1100	USART5_RX	USART5_TX	USART5_RX	USART5_TX	USART5_RX	USART5_RX	USART5_TX
1101	USART6_RX	USART6_TX	USART6_RX	USART6_TX	USART6_RX	USART6_RX	USART6_TX

**Table 31. DMA1 requests for each channel on STM32F09x devices (continued)**

CxS [3:0]	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
<b>1110</b>	USART7_RX	USART7_TX	USART7_RX	USART7_TX	USART7_RX	USART7_RX	USART7_TX
<b>1111</b>	USART8_RX	USART8_TX	USART8_RX	USART8_TX	USART8_RX	USART8_RX	USART8_TX

**Table 32. DMA2 requests for each channel on STM32F09x devices**

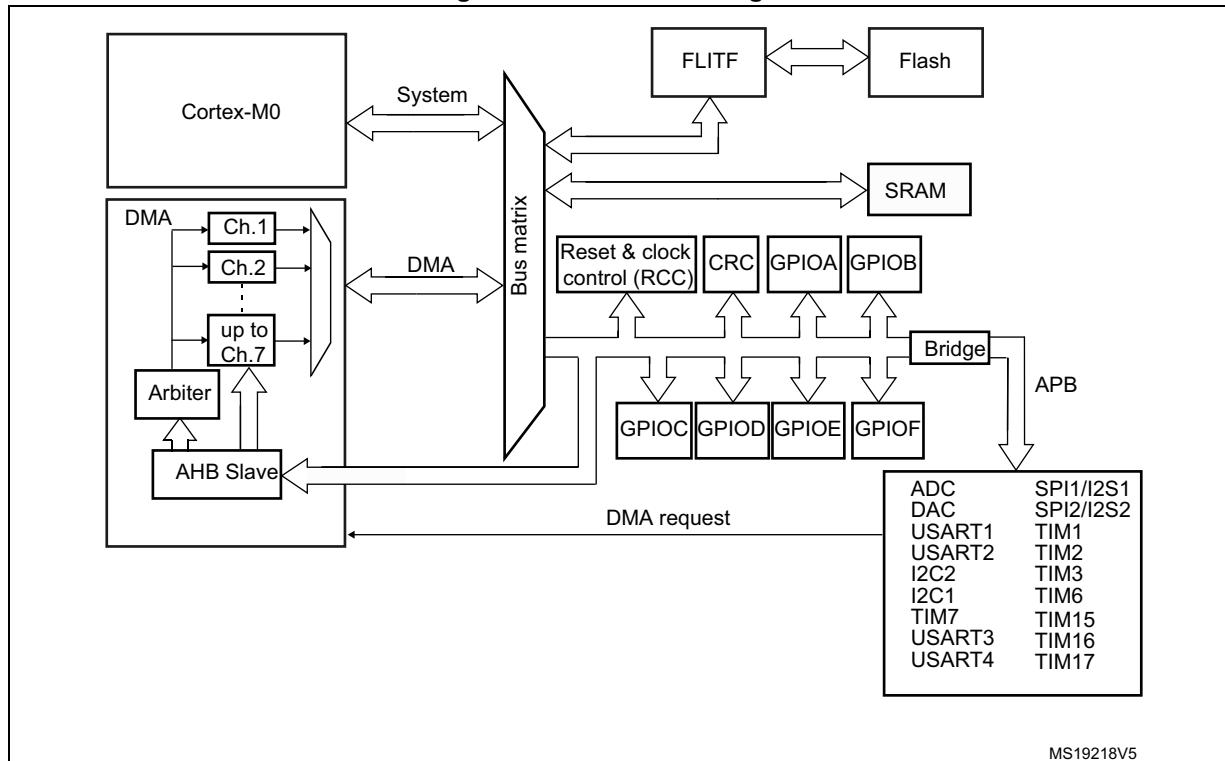
CxS[3:0]	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5
<b>0000</b>	-	-	-	-	-
<b>0001</b>	-	-	TIM6_UP DAC_Channel1	TIM7_UP DAC_Channel2	ADC
<b>0010</b>	I2C2_TX	I2C2_RX	-	-	-
<b>0011</b>	-	-	SPI1_RX	SPI1_TX	-
<b>0100</b>	-	-	-	-	-
<b>0101</b>	-	-	-	-	-
<b>0110</b>	-	-	-	-	-
<b>0111</b>	-	-	-	-	-
<b>1000</b>	USART1_TX	USART1_RX	USART1_RX	USART1_TX	USART1_TX
<b>1001</b>	USART2_TX	USART2_RX	USART2_RX	USART2_TX	USART2_TX
<b>1010</b>	USART3_TX	USART3_RX	USART3_RX	USART3_TX	USART3_TX
<b>1011</b>	USART4_TX	USART4_RX	USART4_RX	USART4_TX	USART4_TX
<b>1100</b>	USART5_TX	USART5_RX	USART5_RX	USART5_TX	USART5_TX
<b>1101</b>	USART6_TX	USART6_RX	USART6_RX	USART6_TX	USART6_TX
<b>1110</b>	USART7_TX	USART7_RX	USART7_RX	USART7_TX	USART7_TX
<b>1111</b>	USART8_TX	USART8_RX	USART8_RX	USART8_TX	USART8_TX

## 10.4 DMA functional description

### 10.4.1 DMA block diagram

The DMA block diagram is shown in the figure below.

**Figure 22. DMA block diagram**



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

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

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

#### 10.4.2 DMA transfers

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

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

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

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

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

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

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

- a single DMA transfer, encapsulating two AHB transfers of a single data, over the DMA AHB bus master:
  - a single data read (byte, half-word or word) from the peripheral data register or a location in the memory, addressed through an internal current peripheral/memory address register.  
The start address used for the first single transfer is the base address of the peripheral or memory, and is programmed in the DMA\_CPARx or DMA\_CMARx register.
  - a single data write (byte, half-word or word) to the peripheral data register or a location in the memory, addressed through an internal current peripheral/memory address register.  
The start address used for the first transfer is the base address of the peripheral or memory, and is programmed in the DMA\_CPARx or DMA\_CMARx register.
- post-decrementing of the programmed DMA\_CNDTRx register  
This register contains the remaining number of data items to transfer (number of AHB ‘read followed by write’ transfers).

This sequence is repeated until DMA\_CNDTRx is null.

**Note:** *The AHB master bus source/destination address must be aligned with the programmed size of the transferred single data to the source/destination.*

#### 10.4.3 DMA arbitration

The DMA arbiter manages the priority between the different channels.

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

The priorities are managed in two stages:

- software: priority of each channel is configured in the DMA\_CCRx register, to one of the four different levels:
  - very high
  - high
  - medium
  - low
- hardware: if two requests have the same software priority level, the channel with the lowest index gets priority. For example, channel 2 gets priority over channel 4.

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

#### 10.4.4 DMA channels

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

A DMA channel is programmed at block transfer level.

##### Programmable data sizes

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

##### Pointer incrementation

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

If the **incremented mode** is enabled (PINC or MINC set to 1), the address of the next transfer is the address of the previous one incremented by 1, 2 or 4, depending on the data size defined in PSIZE[1:0] or MSIZE[1:0]. The first transfer address is the one programmed in the DMA\_CPARx or DMA\_CMARx register. During transfers, these registers keep the initially programmed value. The current transfer addresses (in the current internal peripheral/memory address register) are not accessible by software.

If the channel x is configured in **non-circular mode**, no DMA request is served after the last data transfer (once the number of single data to transfer reaches zero). The DMA channel must be disabled in order to reload a new number of data items into the DMA\_CNDTRx register.

**Note:** *If the channel x is disabled, the DMA registers are not reset. The DMA channel registers (DMA\_CCRx, DMA\_CPARx and DMA\_CMARx) retain the initial values programmed during the channel configuration phase.*

In **circular mode**, after the last data transfer, the DMA\_CNDTRx register is automatically reloaded with the initially programmed value. The current internal address registers are reloaded with the base address values from the DMA\_CPARx and DMA\_CMARx registers.

## Channel configuration procedure

The following sequence is needed to configure a DMA channel x:

1. Set the peripheral register address in the DMA\_CPARx register.  
The data is moved from/to this address to/from the memory after the peripheral event, or after the channel is enabled in memory-to-memory mode.
2. Set the memory address in the DMA\_CMARx register.  
The data is written to/read from the memory after the peripheral event or after the channel is enabled in memory-to-memory mode.
3. Configure the total number of data to transfer in the DMA\_CNDTRx register.  
After each data transfer, this value is decremented.
4. Configure the parameters listed below in the DMA\_CCRx register:
  - the channel priority
  - the data transfer direction
  - the circular mode
  - the peripheral and memory incremented mode
  - the peripheral and memory data size
  - the interrupt enable at half and/or full transfer and/or transfer error
5. Activate the channel by setting the EN bit in the DMA\_CCRx register.

For code example refer to the Appendix [Section A.5.1: DMA channel configuration sequence code example](#).

A channel, as soon as enabled, may serve any DMA request from the peripheral connected to this channel, or may start a memory-to-memory block transfer.

*Note:*

*The two last steps of the channel configuration procedure may be merged into a single access to the DMA\_CCRx register, to configure and enable the channel.*

## Channel state and disabling a channel

A channel x in active state is an enabled channel (read DMA\_CCRx.EN = 1). An active channel x is a channel that must have been enabled by the software (DMA\_CCRx.EN set to 1) and afterwards with no occurred transfer error (DMA\_ISR.TEIFx = 0). In case there is a transfer error, the channel is automatically disabled by hardware (DMA\_CCRx.EN = 0).

The three following use cases may happen:

- Suspend and resume a channel

This corresponds to the two following actions:

- An active channel is disabled by software (writing DMA\_CCRx.EN = 0 whereas DMA\_CCRx.EN = 1).
- The software enables the channel again (DMA\_CCRx.EN set to 1) without reconfiguring the other channel registers (such as DMA\_CNDTRx, DMA\_CPARx and DMA\_CMARx).

This case is not supported by the DMA hardware, that does not guarantee that the remaining data transfers are performed correctly.

- Stop and abort a channel

If the application does not need any more the channel, this active channel can be disabled by software. The channel is stopped and aborted but the DMA\_CNDTRx

register content may not correctly reflect the remaining data transfers versus the aborted source and destination buffer/register.

- Abort and restart a channel

This corresponds to the software sequence: disable an active channel, then reconfigure the channel and enable it again.

This is supported by the hardware if the following conditions are met:

- The application guarantees that, when the software is disabling the channel, a DMA data transfer is not occurring at the same time over its master port. For example, the application can first disable the peripheral in DMA mode, in order to ensure that there is no pending hardware DMA request from this peripheral.
- The software must operate separated write accesses to the same DMA\_CCRx register: First disable the channel. Second reconfigure the channel for a next block transfer including the DMA\_CCRx if a configuration change is needed. There are read-only DMA\_CCRx register fields when DMA\_CCRx.EN=1. Finally enable again the channel.

When a channel transfer error occurs, the EN bit of the DMA\_CCRx register is cleared by hardware. This EN bit can not be set again by software to re-activate the channel x, until the TEIFx bit of the DMA\_ISR register is set.

### Circular mode (in memory-to-peripheral/peripheral-to-memory transfers)

The circular mode is available to handle circular buffers and continuous data flows (such as ADC scan mode). This feature is enabled using the CIRC bit in the DMA\_CCRx register.

Note:

*The circular mode must not be used in memory-to-memory mode. Before enabling a channel in circular mode (CIRC = 1), the software must clear the MEM2MEM bit of the DMA\_CCRx register. When the circular mode is activated, the amount of data to transfer is automatically reloaded with the initial value programmed during the channel configuration phase, and the DMA requests continue to be served.*

*In order to stop a circular transfer, the software needs to stop the peripheral from generating DMA requests (such as quit the ADC scan mode), before disabling the DMA channel.*

*The software must explicitly program the DMA\_CNDTRx value before starting/enabling a transfer, and after having stopped a circular transfer.*

### Memory-to-memory mode

The DMA channels may operate without being triggered by a request from a peripheral. This mode is called memory-to-memory mode, and is initiated by software.

If the MEM2MEM bit in the DMA\_CCRx register is set, the channel, if enabled, initiates transfers. The transfer stops once the DMA\_CNDTRx register reaches zero.

Note:

*The memory-to-memory mode must not be used in circular mode. Before enabling a channel in memory-to-memory mode (MEM2MEM = 1), the software must clear the CIRC bit of the DMA\_CCRx register.*

### Peripheral-to-peripheral mode

Any DMA channel can operate in peripheral-to-peripheral mode:

- when the hardware request from a peripheral is selected to trigger the DMA channel  
This peripheral is the DMA initiator and paces the data transfer from/to this peripheral to/from a register belonging to another memory-mapped peripheral (this one being not configured in DMA mode).
- when no peripheral request is selected and connected to the DMA channel  
The software configures a register-to-register transfer by setting the MEM2MEM bit of the DMA\_CCRx register.

### Programming transfer direction, assigning source/destination

The value of the DIR bit of the DMA\_CCRx register sets the direction of the transfer, and consequently, it identifies the source and the destination, regardless the source/destination type (peripheral or memory):

- **DIR = 1** defines typically a memory-to-peripheral transfer. More generally, if DIR = 1:
  - The **source** attributes are defined by the DMA\_MARx register, the MSIZE[1:0] field and MINC bit of the DMA\_CCRx register.  
Regardless of their usual naming, these ‘memory’ register, field and bit are used to define the source peripheral in peripheral-to-peripheral mode.
  - The **destination** attributes are defined by the DMA\_PARx register, the PSIZE[1:0] field and PINC bit of the DMA\_CCRx register.  
Regardless of their usual naming, these ‘peripheral’ register, field and bit are used to define the destination memory in memory-to-memory mode.
- **DIR = 0** defines typically a peripheral-to-memory transfer. More generally, if DIR = 0:
  - The **source** attributes are defined by the DMA\_PARx register, the PSIZE[1:0] field and PINC bit of the DMA\_CCRx register.  
Regardless of their usual naming, these ‘peripheral’ register, field and bit are used to define the source memory in memory-to-memory mode
  - The **destination** attributes are defined by the DMA\_MARx register, the MSIZE[1:0] field and MINC bit of the DMA\_CCRx register.  
Regardless of their usual naming, these ‘memory’ register, field and bit are used to define the destination peripheral in peripheral-to-peripheral mode.

#### 10.4.5 DMA data width, alignment and endianness

When PSIZE[1:0] and MSIZE[1:0] are not equal, the DMA controller performs some data alignments as described in the table below.

**Table 33. Programmable data width and endian behavior (when PINC = MINC = 1)**

Source port width (MSIZE if DIR = 1, else PSIZE)	Destination port width (PSIZE if DIR = 1, else MSIZE)	Number of data items to transfer (NDT)	Source content: address / data (DMA_CMARx if DIR = 1, else DMA_CPARx)	DMA transfers	Destination content: address / data (DMA_CPARx if DIR = 1, else DMA_CMARx)
8	8	4	@0x0 / B0 @0x1 / B1 @0x2 / B2 @0x3 / B3	1: read B0[7:0] @0x0 then write B0[7:0] @0x0 2: read B1[7:0] @0x1 then write B1[7:0] @0x1 3: read B2[7:0] @0x2 then write B2[7:0] @0x2 4: read B3[7:0] @0x3 then write B3[7:0] @0x3	@0x0 / B0 @0x1 / B1 @0x2 / B2 @0x3 / B3
8	16	4	@0x0 / B0 @0x1 / B1 @0x2 / B2 @0x3 / B3	1: read B0[7:0] @0x0 then write 00B0[15:0] @0x0 2: read B1[7:0] @0x1 then write 00B1[15:0] @0x2 3: read B2[7:0] @0x2 then write 00B2[15:0] @0x4 4: read B3[7:0] @0x3 then write 00B3[15:0] @0x6	@0x0 / 00B0 @0x2 / 00B1 @0x4 / 00B2 @0x6 / 00B3
8	32	4	@0x0 / B0 @0x1 / B1 @0x2 / B2 @0x3 / B3	1: read B0[7:0] @0x0 then write 000000B0[31:0] @0x0 2: read B1[7:0] @0x1 then write 000000B1[31:0] @0x4 3: read B2[7:0] @0x2 then write 000000B2[31:0] @0x8 4: read B3[7:0] @0x3 then write 000000B3[31:0] @0xC	@0x0 / 000000B0 @0x4 / 000000B1 @0x8 / 000000B2 @0xC / 000000B3
16	8	4	@0x0 / B1B0 @0x2 / B3B2 @0x4 / B5B4 @0x6 / B7B6	1: read B1B0[15:0] @0x0 then write B0[7:0] @0x0 2: read B3B2[15:0] @0x2 then write B2[7:0] @0x1 3: read B5B4[15:0] @0x4 then write B4[7:0] @0x2 4: read B7B6[15:0] @0x6 then write B6[7:0] @0x3	@0x0 / B0 @0x1 / B2 @0x2 / B4 @0x3 / B6
16	16	4	@0x0 / B1B0 @0x2 / B3B2 @0x4 / B5B4 @0x6 / B7B6	1: read B1B0[15:0] @0x0 then write B1B0[15:0] @0x0 2: read B3B2[15:0] @0x2 then write B3B2[15:0] @0x2 3: read B5B4[15:0] @0x4 then write B5B4[15:0] @0x4 4: read B7B6[15:0] @0x6 then write B7B6[15:0] @0x6	@0x0 / B1B0 @0x2 / B3B2 @0x4 / B5B4 @0x6 / B7B6
16	32	4	@0x0 / B1B0 @0x2 / B3B2 @0x4 / B5B4 @0x6 / B7B6	1: read B1B0[15:0] @0x0 then write 0000B1B0[31:0] @0x0 2: read B3B2[15:0] @0x2 then write 0000B3B2[31:0] @0x4 3: read B5B4[15:0] @0x4 then write 0000B5B4[31:0] @0x8 4: read B7B6[15:0] @0x6 then write 0000B7B6[31:0] @0xC	@0x0 / 0000B1B0 @0x4 / 0000B3B2 @0x8 / 0000B5B4 @0xC / 0000B7B6
32	8	4	@0x0 / B3B2B1B0 @0x4 / B7B6B5B4 @0x8 / BBBAB9B8 @0xC / BFBEBDDBC	1: read B3B2B1B0[31:0] @0x0 then write B0[7:0] @0x0 2: read B7B6B5B4[31:0] @0x4 then write B4[7:0] @0x1 3: read BBBAB9B8[31:0] @0x8 then write B8[7:0] @0x2 4: read BFBEBDDBC[31:0] @0xC then write BC[7:0] @0x3	@0x0 / B0 @0x1 / B4 @0x2 / B8 @0x3 / BC
32	16	4	@0x0 / B3B2B1B0 @0x4 / B7B6B5B4 @0x8 / BBBAB9B8 @0xC / BFBEBDDBC	1: read B3B2B1B0[31:0] @0x0 then write B1B0[15:0] @0x0 2: read B7B6B5B4[31:0] @0x4 then write B5B4[15:0] @0x2 3: read BBBAB9B8[31:0] @0x8 then write B9B8[15:0] @0x4 4: read BFBEBDDBC[31:0] @0xC then write BDBC[15:0] @0x6	@0x0 / B1B0 @0x2 / B5B4 @0x4 / B9B8 @0x6 / BDBC
32	32	4	@0x0 / B3B2B1B0 @0x4 / B7B6B5B4 @0x8 / BBBAB9B8 @0xC / BFBEBDDBC	1: read B3B2B1B0[31:0] @0x0 then write B3B2B1B0[31:0] @0x0 2: read B7B6B5B4[31:0] @0x4 then write B7B6B5B4[31:0] @0x4 3: read BBBAB9B8[31:0] @0x8 then write BBBAB9B8[31:0] @0x8 4: read BFBEBDDBC[31:0] @0xC then write BFBEBDDBC[31:0] @0xC	@0x0 / B3B2B1B0 @0x4 / B7B6B5B4 @0x8 / BBBAB9B8 @0xC / BFBEBDDBC

### Addressing AHB peripherals not supporting byte/half-word write transfers

When the DMA controller initiates an AHB byte or half-word write transfer, the data are duplicated on the unused lanes of the AHB master 32-bit data bus (HWDATA[31:0]).

When the AHB slave peripheral does not support byte or half-word write transfers and does not generate any error, the DMA controller writes the 32 HWDATA bits as shown in the two examples below:

- To write the half-word 0xABCD, the DMA controller sets the HWDATA bus to 0xABCDABCD with a half-word data size (HSIZE = HalfWord in AHB master bus).
- To write the byte 0xAB, the DMA controller sets the HWDATA bus to 0xABABABAB with a byte data size (HSIZE = Byte in the AHB master bus).

Assuming the AHB/APB bridge is an AHB 32-bit slave peripheral that does not take into account the HSIZE data, any AHB byte or half-word transfer is changed into a 32-bit APB transfer as described below:

- An AHB byte write transfer of 0xB0 to one of the 0x0, 0x1, 0x2 or 0x3 addresses, is converted to an APB word write transfer of 0xB0B0B0B0 to the 0x0 address.
- An AHB half-word write transfer of 0xB1B0 to the 0x0 or 0x2 addresses, is converted to an APB word write transfer of 0xB1B0B1B0 to the 0x0 address.

#### 10.4.6 DMA error management

A DMA transfer error is generated when reading from or writing to a reserved address space. When a DMA transfer error occurs during a DMA read or write access, the faulty channel x is automatically disabled through a hardware clear of its EN bit in the corresponding DMA\_CCRx register.

The TEIFx bit of the DMA\_ISR register is set. An interrupt is then generated if the TEIE bit of the DMA\_CCRx register is set.

The EN bit of the DMA\_CCRx register can not be set again by software (channel x re-activated) until the TEIFx bit of the DMA\_ISR register is cleared (by setting the CTEIFx bit of the DMA\_IFCR register).

When the software is notified with a transfer error over a channel which involves a peripheral, the software has first to stop this peripheral in DMA mode, in order to disable any pending or future DMA request. Then software may normally reconfigure both DMA and the peripheral in DMA mode for a new transfer.

### 10.5 DMA interrupts

An interrupt can be generated on a half transfer, transfer complete or transfer error for each DMA channel x. Separate interrupt enable bits are available for flexibility.

**Table 34. DMA interrupt requests**

Interrupt request	Interrupt event	Event flag	Interrupt enable bit
Channel x interrupt	Half transfer on channel x	HTIFx	HTIEx
	Transfer complete on channel x	TCIFx	TCIEx
	Transfer error on channel x	TEIFx	TEIEx
	Half transfer or transfer complete or transfer error on channel x	GIFx	-

### 10.6 DMA registers

Refer to [Section 1.2](#) for a list of abbreviations used in register descriptions.

The DMA registers have to be accessed by words (32-bit).

### 10.6.1 DMA interrupt status register (DMA\_ISR)

Address offset: 0x000

Reset value: 0x0000 0000

The content of this register is linked to the DMA channels availability. See [Section 10.3: DMA implementation](#) for more details.

Every status bit is cleared by hardware when the software sets the corresponding clear bit or the corresponding global clear bit CGIFx, in the DMA\_IFCR register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	Res	Res	Res	TEIF7	HTIF7	TCIF7	GIF7	TEIF6	HTIF6	TCIF6	GIF6	TEIF5	HTIF5	TCIF5	GIF5
				r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TEIF4	HTIF4	TCIF4	GIF4	TEIF3	HTIF3	TCIF3	GIF3	TEIF2	HTIF2	TCIF2	GIF2	TEIF1	HTIF1	TCIF1	GIF1
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:28 Reserved, must be kept at reset value.

Bit 27 **TEIF7**: transfer error (TE) flag for channel 7

- 0: no TE event
- 1: a TE event occurred

Bit 26 **HTIF7**: half transfer (HT) flag for channel 7

- 0: no HT event
- 1: a HT event occurred

Bit 25 **TCIF7**: transfer complete (TC) flag for channel 7

- 0: no TC event
- 1: a TC event occurred

Bit 24 **GIF7**: global interrupt flag for channel 7

- 0: no TE, HT or TC event
- 1: a TE, HT or TC event occurred

Bit 23 **TEIF6**: transfer error (TE) flag for channel 6

- 0: no TE event
- 1: a TE event occurred

Bit 22 **HTIF6**: half transfer (HT) flag for channel 6

- 0: no HT event
- 1: a HT event occurred

Bit 21 **TCIF6**: transfer complete (TC) flag for channel 6

- 0: no TC event
- 1: a TC event occurred

Bit 20 **GIF6**: global interrupt flag for channel 6

- 0: no TE, HT or TC event
- 1: a TE, HT or TC event occurred

Bit 19 **TEIF5**: transfer error (TE) flag for channel 5

- 0: no TE event
- 1: a TE event occurred

- Bit 18 **HTIF5**: half transfer (HT) flag for channel 5  
0: no HT event  
1: a HT event occurred
- Bit 17 **TCIF5**: transfer complete (TC) flag for channel 5  
0: no TC event  
1: a TC event occurred
- Bit 16 **GIF5**: global interrupt flag for channel 5  
0: no TE, HT or TC event  
1: a TE, HT or TC event occurred
- Bit 15 **TEIF4**: transfer error (TE) flag for channel 4  
0: no TE event  
1: a TE event occurred
- Bit 14 **HTIF4**: half transfer (HT) flag for channel 4  
0: no HT event  
1: a HT event occurred
- Bit 13 **TCIF4**: transfer complete (TC) flag for channel 4  
0: no TC event  
1: a TC event occurred
- Bit 12 **GIF4**: global interrupt flag for channel 4  
0: no TE, HT or TC event  
1: a TE, HT or TC event occurred
- Bit 11 **TEIF3**: transfer error (TE) flag for channel 3  
0: no TE event  
1: a TE event occurred
- Bit 10 **HTIF3**: half transfer (HT) flag for channel 3  
0: no HT event  
1: a HT event occurred
- Bit 9 **TCIF3**: transfer complete (TC) flag for channel 3  
0: no TC event  
1: a TC event occurred
- Bit 8 **GIF3**: global interrupt flag for channel 3  
0: no TE, HT or TC event  
1: a TE, HT or TC event occurred
- Bit 7 **TEIF2**: transfer error (TE) flag for channel 2  
0: no TE event  
1: a TE event occurred
- Bit 6 **HTIF2**: half transfer (HT) flag for channel 2  
0: no HT event  
1: a HT event occurred
- Bit 5 **TCIF2**: transfer complete (TC) flag for channel 2  
0: no TC event  
1: a TC event occurred
- Bit 4 **GIF2**: global interrupt flag for channel 2  
0: no TE, HT or TC event  
1: a TE, HT or TC event occurred

- Bit 3 **TEIF1**: transfer error (TE) flag for channel 1  
 0: no TE event  
 1: a TE event occurred
- Bit 2 **HTIF1**: half transfer (HT) flag for channel 1  
 0: no HT event  
 1: a HT event occurred
- Bit 1 **TCIF1**: transfer complete (TC) flag for channel 1  
 0: no TC event  
 1: a TC event occurred
- Bit 0 **GIF1**: global interrupt flag for channel 1  
 0: no TE, HT or TC event  
 1: a TE, HT or TC event occurred

## 10.6.2 DMA interrupt flag clear register (DMA\_IFCR)

Address offset: 0x04

Reset value: 0x0000 0000

The content of this register is linked to the DMA channels availability. See [Section 10.3: DMA implementation](#) for more details.

Setting the global clear bit CGIFx of the channel x in this DMA\_IFCR register, causes the DMA hardware to clear the corresponding GIFx bit and any individual flag among TEIFx, HTIFx, TCIFx, in the DMA\_ISR register.

Setting any individual clear bit among CTEIFx, CHTIFx, CTCIFx in this DMA\_IFCR register, causes the DMA hardware to clear the corresponding individual flag and the global flag GIFx in the DMA\_ISR register, provided that none of the two other individual flags is set.

Writing 0 into any flag clear bit has no effect.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	CTEIF7	CHTIF7	CTCIF7	CGIF7	CTEIF6	CHTIF6	CTCIF6	CGIF6	CTEIF5	CHTIF5	CTCIF5	CGIF5
				w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTEIF4	CHTIF4	CTCIF4	CGIF4	CTEIF3	CHTIF3	CTCIF3	CGIF3	CTEIF2	CHTIF2	CTCIF2	CGIF2	CTEIF1	CHTIF1	CTCIF1	CGIF1
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:28 Reserved, must be kept at reset value.

Bit 27 **CTEIF7**: transfer error flag clear for channel 7

Bit 26 **CHTIF7**: half transfer flag clear for channel 7

Bit 25 **CTCIF7**: transfer complete flag clear for channel 7

Bit 24 **CGIF7**: global interrupt flag clear for channel 7

Bit 23 **CTEIF6**: transfer error flag clear for channel 6

Bit 22 **CHTIF6**: half transfer flag clear for channel 6

- Bit 21 **CTCIF6**: transfer complete flag clear for channel 6
- Bit 20 **CGIF6**: global interrupt flag clear for channel 6
- Bit 19 **CTEIF5**: transfer error flag clear for channel 5
- Bit 18 **CHTIF5**: half transfer flag clear for channel 5
- Bit 17 **CTCIF5**: transfer complete flag clear for channel 5
- Bit 16 **CGIF5**: global interrupt flag clear for channel 5
- Bit 15 **CTEIF4**: transfer error flag clear for channel 4
- Bit 14 **CHTIF4**: half transfer flag clear for channel 4
- Bit 13 **CTCIF4**: transfer complete flag clear for channel 4
- Bit 12 **CGIF4**: global interrupt flag clear for channel 4
- Bit 11 **CTEIF3**: transfer error flag clear for channel 3
- Bit 10 **CHTIF3**: half transfer flag clear for channel 3
- Bit 9 **CTCIF3**: transfer complete flag clear for channel 3
- Bit 8 **CGIF3**: global interrupt flag clear for channel 3
- Bit 7 **CTEIF2**: transfer error flag clear for channel 2
- Bit 6 **CHTIF2**: half transfer flag clear for channel 2
- Bit 5 **CTCIF2**: transfer complete flag clear for channel 2
- Bit 4 **CGIF2**: global interrupt flag clear for channel 2
- Bit 3 **CTEIF1**: transfer error flag clear for channel 1
- Bit 2 **CHTIF1**: half transfer flag clear for channel 1
- Bit 1 **CTCIF1**: transfer complete flag clear for channel 1
- Bit 0 **CGIF1**: global interrupt flag clear for channel 1

### 10.6.3 DMA channel x configuration register (DMA\_CCRx)

Address offset:  $0x08 + 0x14 * (x - 1)$ , ( $x = 1$  to  $7$ )

Reset value: 0x0000 0000

The address offsets of these registers are linked to the DMA channels availability. See [Section 10.3: DMA implementation](#) for more details.

The register fields/bits MEM2MEM, PL[1:0], MSIZE[1:0], PSIZE[1:0], MINC, PINC, and DIR are read-only when EN = 1.

The states of MEM2MEM and CIRC bits must not be both high at the same time.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	MEM2 MEM	PL[1:0]		MSIZE[1:0]		PSIZE[1:0]		MINC	PINC	CIRC	DIR	TEIE	HTIE	TCIE	EN
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bit 14 **MEM2MEM**: memory-to-memory mode

0: disabled

1: enabled

*Note: this bit is set and cleared by software.*

*It must not be written when the channel is enabled (EN = 1).*

*It is read-only when the channel is enabled (EN = 1).*

Bits 13:12 **PL[1:0]**: priority level

00: low

01: medium

10: high

11: very high

*Note: this field is set and cleared by software.*

*It must not be written when the channel is enabled (EN = 1).*

*It is read-only when the channel is enabled (EN = 1).*

Bits 11:10 **MSIZE[1:0]**: memory size

Defines the data size of each DMA transfer to the identified memory.

In memory-to-memory mode, this field identifies the memory source if DIR = 1 and the memory destination if DIR = 0.

In peripheral-to-peripheral mode, this field identifies the peripheral source if DIR = 1 and the peripheral destination if DIR = 0.

00: 8 bits

01: 16 bits

10: 32 bits

11: reserved

*Note: this field is set and cleared by software.*

*It must not be written when the channel is enabled (EN = 1).*

*It is read-only when the channel is enabled (EN = 1).*

Bits 9:8 **PSIZE[1:0]**: peripheral size

Defines the data size of each DMA transfer to the identified peripheral.

In memory-to-memory mode, this field identifies the memory destination if DIR = 1 and the memory source if DIR = 0.

In peripheral-to-peripheral mode, this field identifies the peripheral destination if DIR = 1 and the peripheral source if DIR = 0.

00: 8 bits

01: 16 bits

10: 32 bits

11: reserved

*Note: this field is set and cleared by software.*

*It must not be written when the channel is enabled (EN = 1).*

*It is read-only when the channel is enabled (EN = 1).*

Bit 7 **MINC**: memory increment mode

Defines the increment mode for each DMA transfer to the identified memory.

In memory-to-memory mode, this field identifies the memory source if DIR = 1 and the memory destination if DIR = 0.

In peripheral-to-peripheral mode, this field identifies the peripheral source if DIR = 1 and the peripheral destination if DIR = 0.

0: disabled

1: enabled

*Note: this bit is set and cleared by software.*

*It must not be written when the channel is enabled (EN = 1).*

*It is read-only when the channel is enabled (EN = 1).*

Bit 6 **PINC**: peripheral increment mode

Defines the increment mode for each DMA transfer to the identified peripheral.

In memory-to-memory mode, this field identifies the memory destination if DIR = 1 and the memory source if DIR = 0.

In peripheral-to-peripheral mode, this field identifies the peripheral destination if DIR = 1 and the peripheral source if DIR = 0.

0: disabled

1: enabled

*Note: this bit is set and cleared by software.*

*It must not be written when the channel is enabled (EN = 1).*

*It is read-only when the channel is enabled (EN = 1).*

Bit 5 **CIRC**: circular mode

0: disabled

1: enabled

*Note: this bit is set and cleared by software.*

*It must not be written when the channel is enabled (EN = 1).*

*It is not read-only when the channel is enabled (EN = 1).*

Bit 4 **DIR**: data transfer direction

This bit must be set only in memory-to-peripheral and peripheral-to-memory modes.

0: read from peripheral

- Source attributes are defined by PSIZE and PINC, plus the DMA\_CPARx register.  
This is still valid in a memory-to-memory mode.
- Destination attributes are defined by MSIZE and MINC, plus the DMA\_CMARx register. This is still valid in a peripheral-to-peripheral mode.

1: read from memory

- Destination attributes are defined by PSIZE and PINC, plus the DMA\_CPARx register. This is still valid in a memory-to-memory mode.
- Source attributes are defined by MSIZE and MINC, plus the DMA\_CMARx register.  
This is still valid in a peripheral-to-peripheral mode.

*Note: this bit is set and cleared by software.*

*It must not be written when the channel is enabled (EN = 1).*

*It is read-only when the channel is enabled (EN = 1).*

Bit 3 **TEIE**: transfer error interrupt enable

0: disabled

1: enabled

*Note: this bit is set and cleared by software.*

*It must not be written when the channel is enabled (EN = 1).*

*It is not read-only when the channel is enabled (EN = 1).*

Bit 2 **HTIE**: half transfer interrupt enable

- 0: disabled
- 1: enabled

*Note: this bit is set and cleared by software.*

*It must not be written when the channel is enabled (EN = 1).*

*It is not read-only when the channel is enabled (EN = 1).*

Bit 1 **TCIE**: transfer complete interrupt enable

- 0: disabled
- 1: enabled

*Note: this bit is set and cleared by software.*

*It must not be written when the channel is enabled (EN = 1).*

*It is not read-only when the channel is enabled (EN = 1).*

Bit 0 **EN**: channel enable

When a channel transfer error occurs, this bit is cleared by hardware. It can not be set again by software (channel x re-activated) until the TEIFx bit of the DMA\_ISR register is cleared (by setting the CTEIFx bit of the DMA\_IFCR register).

- 0: disabled
- 1: enabled

*Note: this bit is set and cleared by software.*

#### 10.6.4 DMA channel x number of data to transfer register (DMA\_CNDTR $x$ )

Address offset: 0x0C + 0x14 \* ( $x - 1$ ), ( $x = 1$  to 7)

Reset value: 0x0000 0000

The address offsets of these registers are linked to the DMA channels availability. See [Section 10.3: DMA implementation](#) for more details.

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

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

Bits 15:0 **NDT[15:0]**: number of data to transfer (0 to  $2^{16} - 1$ )

This field is updated by hardware when the channel is enabled:

- It is decremented after each single DMA ‘read followed by write’ transfer, indicating the remaining amount of data items to transfer.
- It is kept at zero when the programmed amount of data to transfer is reached, if the channel is not in circular mode (CIRC = 0 in the DMA\_CCR $x$  register).
- It is reloaded automatically by the previously programmed value, when the transfer is complete, if the channel is in circular mode (CIRC = 1).

If this field is zero, no transfer can be served whatever the channel status (enabled or not).

*Note: this field is set and cleared by software.*

*It must not be written when the channel is enabled (EN = 1).*

*It is read-only when the channel is enabled (EN = 1).*

### 10.6.5 DMA channel x peripheral address register (DMA\_CPARx)

Address offset:  $0x10 + 0x14 * (x - 1)$ , ( $x = 1$  to  $7$ )

Reset value: 0x0000 0000

The address offsets of these registers are linked to the DMA channels availability. See [Section 10.3: DMA implementation](#) for more details.

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

Bits 31:0 **PA[31:0]**: peripheral address

It contains the base address of the peripheral data register from/to which the data is read/written.

When PSIZE[1:0] = 01 (16 bits), bit 0 of PA[31:0] is ignored. Access is automatically aligned to a half-word address.

When PSIZE = 10 (32 bits), bits 1 and 0 of PA[31:0] are ignored. Access is automatically aligned to a word address.

In memory-to-memory mode, this register identifies the memory destination address if DIR = 1 and the memory source address if DIR = 0.

In peripheral-to-peripheral mode, this register identifies the peripheral destination address DIR = 1 and the peripheral source address if DIR = 0.

*Note: this register is set and cleared by software.*

*It must not be written when the channel is enabled (EN = 1).*

*It is not read-only when the channel is enabled (EN = 1).*

### 10.6.6 DMA channel x memory address register (DMA\_CMARx)

Address offset:  $0x14 + 0x14 * (x - 1)$ , ( $x = 1$  to  $7$ )

Reset value: 0x0000 0000

The address offsets of these registers are linked to the DMA channels availability. See [Section 10.3: DMA implementation](#) for more details.

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

Bits 31:0 **MA[31:0]**: peripheral address

It contains the base address of the memory from/to which the data is read/written.

When MSIZE[1:0] = 01 (16 bits), bit 0 of MA[31:0] is ignored. Access is automatically aligned to a half-word address.

When MSIZE = 10 (32 bits), bits 1 and 0 of MA[31:0] are ignored. Access is automatically aligned to a word address.

In memory-to-memory mode, this register identifies the memory source address if DIR = 1 and the memory destination address if DIR = 0.

In peripheral-to-peripheral mode, this register identifies the peripheral source address DIR = 1 and the peripheral destination address if DIR = 0.

*Note: this register is set and cleared by software.*

*It must not be written when the channel is enabled (EN = 1).*

*It is not read-only when the channel is enabled (EN = 1).*

### 10.6.7 DMA channel selection register (DMA\_CSELR)

Address offset: 0xA8

Reset value: 0x0000 0000

This register is present only on STM32F09xx devices.

This register is used to manage the mapping of DMA channels as detailed in [Section 10.3.2: DMA request mapping](#).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	C7S[3:0]				C6S[3:0]				C5S[3:0]			
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C4S[3:0]				C3S[3:0]				C2S[3:0]				C1S[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:28 Reserved, must be kept at reset value.

Bits 27:24 **C7S[3:0]**: DMA channel 7 selection

Details available in [Section 10.3.2: DMA request mapping](#)

Bits 23:20 **C6S[3:0]**: DMA channel 6 selection

Details available in [Section 10.3.2: DMA request mapping](#)

Bits 19:16 **C5S[3:0]**: DMA channel 5 selection

Details available in [Section 10.3.2: DMA request mapping](#)

Bits 15:12 **C4S[3:0]**: DMA channel 4 selection

Details available in [Section 10.3.2: DMA request mapping](#)

Bits 11:8 **C3S[3:0]**: DMA channel 3 selection

Details available in [Section 10.3.2: DMA request mapping](#)

Bits 7:4 **C2S[3:0]**: DMA channel 2 selection

Details available in [Section 10.3.2: DMA request mapping](#)

Bits 3:0 **C1S[3:0]**: DMA channel 1 selection

Details available in [Section 10.3.2: DMA request mapping](#)

### 10.6.8 DMA register map

The table below gives the DMA register map and reset values.

**Table 35. DMA register map and reset values**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
0x000	DMA_ISR	Res.	Res.	Res.	Res.	TEIF7	HTIF7	TCIF7	GIF7	TEIF6	HTIF6	TCIF6	TEIF5	HTIF5	TCIF5	TEIF4	HTIF4	
	Reset value					0	0	0	0	0	0	0	0	0	0	0	0	
0x004	DMA_IFCR	Res.	Res.	Res.	Res.	CTEIF7	CHTIF7	CTCIF7	CGIF7	CTEIF6	CHTIF6	CTCIF6	CTEIF5	CHTIF5	CTCIF5	CTEIF4	CHTIF4	CTCIF4
	Reset value					0	0	0	0	0	0	0	0	0	0	0	0	

**Table 35. DMA register map and reset values (continued)**

**Table 35. DMA register map and reset values (continued)**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x05C	DMA_CNDTR5	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value																																	
0x060	DMA_CPAR5																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x064	DMA_CMAR5																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x068	Reserved																																	
0x06C	DMA_CCR6	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																	
0x070	DMA_CNDTR6	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																	
0x074	DMA_CPAR6																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x078	DMA_CMAR6																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x07C	Reserved																																	
0x080	DMA_CCR7	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																	
0x084	DMA_CNDTR7	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																	
0x088	DMA_CPAR7																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x08C	DMA_CMAR7																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x090 to 0x0A4	Reserved																																	
0x0A8	DMA_CSELR	Res.	Res.	Res.	Res.	C7S[3:0]	C6S[3:0]	C5S[3:0]	C4S[3:0]	C3S[3:0]	C2S[3:0]	C1S[3:0]																						
	Reset value					0	0	0	0	0	0	0																						

Refer to [Section 2.2](#) for the register boundary addresses.

## 11 Interrupts and events

### 11.1 Nested vectored interrupt controller (NVIC)

#### 11.1.1 NVIC main features

- 32 maskable interrupt channels (not including the sixteen Cortex®-M0 interrupt lines)
- 4 programmable priority levels (2 bits of interrupt priority are used)
- Low-latency exception and interrupt handling
- Power management control
- Implementation of System control registers

The NVIC and the processor core interface are closely coupled, which enables low latency interrupt processing and efficient processing of late arriving interrupts.

All interrupts including the core exceptions are managed by the NVIC. For more information on exceptions and NVIC programming, refer to the PM0215 programming manual.

For code example refer to the Appendix section [A.6.1: NVIC initialization example](#).

#### 11.1.2 SysTick calibration value register

The SysTick calibration value is set to 6000, which gives a reference time base of 1 ms with the SysTick clock set to 6 MHz (max  $f_{HCLK} / 8$ ).

#### 11.1.3 Interrupt and exception vectors

[Table 36](#) is the vector table for STM32F0xx devices. Consider peripheral availability on your device.

**Table 36. Vector table**

Position	Priority	Type of priority	Acronym	Description	Address
-	-	-	-	Reserved	0x0000 0000
-	-3	Fixed	Reset	Reset	0x0000 0004
-	-2	Fixed	NMI	Non maskable interrupt. The RCC clock security system (CSS) and the RAM parity check are linked to the NMI vector.	0x0000 0008
-	-1	Fixed	HardFault	All classes of fault	0x0000 000C
-	3	Settable	SVCALL	System service call via SWI instruction	0x0000 002C
-	5	Settable	PendSV	Pendable request for system service	0x0000 0038
-	6	Settable	SysTick	System tick timer	0x0000 003C
0	7	Settable	WWDG	Window watchdog interrupt	0x0000 0040
1	8	Settable	PVD_VDDIO2	PVD and $V_{DDIO2}$ supply comparator interrupt (combined EXTI lines 16 and 31)	0x0000 0044
2	9	Settable	RTC	RTC interrupts (combined EXTI lines 17, 19 and 20)	0x0000 0048

**Table 36. Vector table (continued)**

<b>Position</b>	<b>Priority</b>	<b>Type of priority</b>	<b>Acronym</b>	<b>Description</b>	<b>Address</b>
3	10	Settable	FLASH	Flash global interrupt	0x0000 004C
4	11	Settable	RCC_CRS	RCC and CRS global interrupts	0x0000 0050
5	12	Settable	EXTI0_1	EXTI Line[1:0] interrupts	0x0000 0054
6	13	Settable	EXTI2_3	EXTI Line[3:2] interrupts	0x0000 0058
7	14	Settable	EXTI4_15	EXTI Line[15:4] interrupts	0x0000 005C
8	15	Settable	TSC	Touch sensing interrupt	0x0000 0060
9	16	Settable	DMA_CH1	DMA channel 1 interrupt	0x0000 0064
10	17	Settable	DMA_CH2_3 DMA2_CH1_2	DMA channel 2 and 3 interrupts DMA2 channel 1 and 2 interrupts	0x0000 0068
11	18	Settable	DMA_CH4_5_6_7 DMA2_CH3_4_5	DMA channel 4, 5, 6 and 7 interrupts DMA2 channel 3, 4 and 5 interrupts	0x0000 006C
12	19	Settable	ADC_COMP	ADC and COMP interrupts (ADC interrupt combined with EXTI lines 21 and 22)	0x0000 0070
13	20	Settable	TIM1_BRK_UP_ TRG_COM	TIM1 break, update, trigger and commutation interrupt	0x0000 0074
14	21	Settable	TIM1_CC	TIM1 capture compare interrupt	0x0000 0078
15	22	Settable	TIM2	TIM2 global interrupt	0x0000 007C
16	23	Settable	TIM3	TIM3 global interrupt	0x0000 0080
17	24	Settable	TIM6_DAC	TIM6 global interrupt and DAC underrun interrupt	0x0000 0084
18	25	Settable	TIM7	TIM7 global interrupt	0x0000 0088
19	26	Settable	TIM14	TIM14 global interrupt	0x0000 008C
20	27	Settable	TIM15	TIM15 global interrupt	0x0000 0090
21	28	Settable	TIM16	TIM16 global interrupt	0x0000 0094
22	29	Settable	TIM17	TIM17 global interrupt	0x0000 0098
23	30	Settable	I2C1	I <sup>2</sup> C1 global interrupt (combined with EXTI line 23)	0x0000 009C
24	31	Settable	I2C2	I <sup>2</sup> C2 global interrupt	0x0000 00A0
25	32	Settable	SPI1	SPI1 global interrupt	0x0000 00A4
26	33	Settable	SPI2	SPI2 global interrupt	0x0000 00A8
27	34	Settable	USART1	USART1 global interrupt (combined with EXTI line 25)	0x0000 00AC
28	35	Settable	USART2	USART2 global interrupt (combined with EXTI line 26)	0x0000 00B0
29	36	Settable	USART3_4_5_6_7_8	USART3, USART4, USART5, USART6, USART7, USART8 global interrupts (combined with EXTI line 28)	0x0000 00B4
30	37	Settable	CEC_CAN	CEC and CAN global interrupts (combined with EXTI line 27)	0x0000 00B8
31	38	Settable	USB	USB global interrupt (combined with EXTI line 18)	0x0000 00BC

## 11.2 Extended interrupts and events controller (EXTI)

The extended interrupts and events controller (EXTI) manages the external and internal asynchronous events/interrupts and generates the event request to the CPU/Interrupt controller and a wake-up request to the Power manager.

The EXTI allows the management of up to 32 external/internal event line (23 external event lines and 9 internal event lines).

The active edge of each external interrupt line can be chosen independently, whilst for internal interrupt the active edge is always the rising one. An interrupt could be left pending: in case of an external one, a status register is instantiated and indicates the source of the interrupt; an event is always a simple pulse and it's used for triggering the core Wake-up (e.g. Cortex-M0 RXEV pin). For internal interrupts, the pending status is assured by the generating IP, so no need for a specific flag. Each input line can be masked independently for interrupt or event generation, in addition the internal lines are sampled only in STOP mode. This controller allows also to emulate the (only) external events by software, multiplexed with the corresponding hardware event line, by writing to a dedicated register.

### 11.2.1 Main features

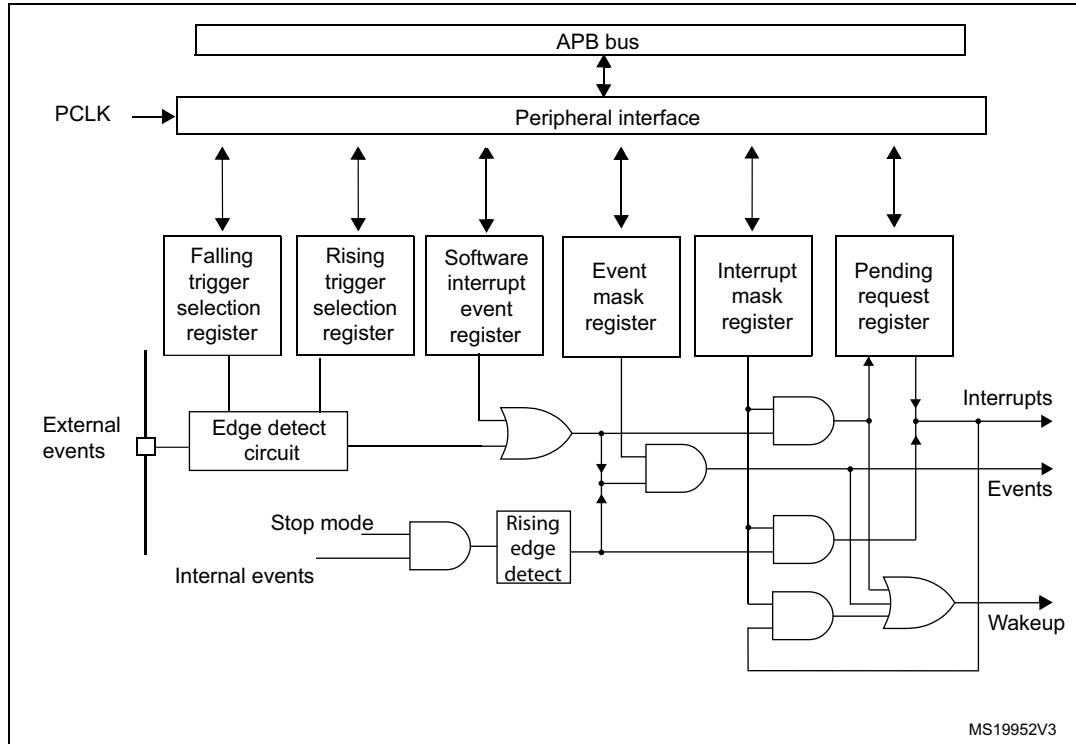
The EXTI main features are the following:

- Supports generation of up to 32 event/interrupt requests
- Independent mask on each event/interrupt line
- Automatic disable of internal lines when system is not in STOP mode
- Independent trigger for external event/interrupt line
- Dedicated status bit for external interrupt line
- Emulation for all the external event requests

## 11.2.2 Block diagram

The extended interrupt/event block diagram is shown in [Figure 23](#).

**Figure 23. Extended interrupts and events controller (EXTI) block diagram**



## 11.2.3 Event management

The STM32F0xx is able to handle external or internal events in order to wake up the core (WFE). The wake-up event can be generated either by:

- enabling an interrupt in the peripheral control register but not in the NVIC, and enabling the SEVONPEND bit in the Cortex-M0 System control register. When the MCU resumes from WFE, the EXTI peripheral interrupt pending bit and the peripheral NVIC IRQ channel pending bit (in the NVIC interrupt clear pending register) have to be cleared.
- or by configuring an external or internal EXTI line in event mode. When the CPU resumes from WFE, it is not necessary to clear the peripheral interrupt pending bit or the NVIC IRQ channel pending bit as the pending bit corresponding to the event line is not set.

## 11.2.4 Functional description

For the external interrupt lines, to generate the interrupt, the interrupt line should be configured and enabled. This is done by programming the two trigger registers with the desired edge detection and by enabling the interrupt request by writing a '1' to the corresponding bit in the interrupt mask register. When the selected edge occurs on the external interrupt line, an interrupt request is generated. The pending bit corresponding to the interrupt line is also set. This request is reset by writing a '1' in the pending register.

For the internal interrupt lines, the active edge is always the rising edge, the interrupt is enabled by default in the interrupt mask register and there is no corresponding pending bit in the pending register.

To generate the event, the event line should be configured and enabled. This is done by programming the two trigger registers with the desired edge detection and by enabling the event request by writing a '1' to the corresponding bit in the event mask register. When the selected edge occurs on the event line, an event pulse is generated. The pending bit corresponding to the event line is not set.

For the external lines, an interrupt/event request can also be generated by software by writing a '1' in the software interrupt/event register.

**Note:** *The interrupts or events associated to the internal lines can be triggered only when the system is in STOP mode. If the system is still running, no interrupt/event is generated.*

For code example refer to the Appendix section [A.6.2: External interrupt selection code example](#).

### Hardware interrupt selection

To configure a line as interrupt source, use the following procedure:

- Configure the corresponding mask bit in the EXTI\_IMR register.
- Configure the trigger selection bits of the interrupt line (EXTI\_RTSR and EXTI\_FTSR)
- Configure the enable and mask bits that control the NVIC IRQ channel mapped to the EXTI so that an interrupt coming from one of the EXTI line can be correctly acknowledged.

### Hardware event selection

To configure a line as event source, use the following procedure:

- Configure the corresponding mask bit in the EXTI\_EMR register.
- Configure the Trigger Selection bits of the Event line (EXTI\_RTSR and EXTI\_FTSR)

### Software interrupt/event selection

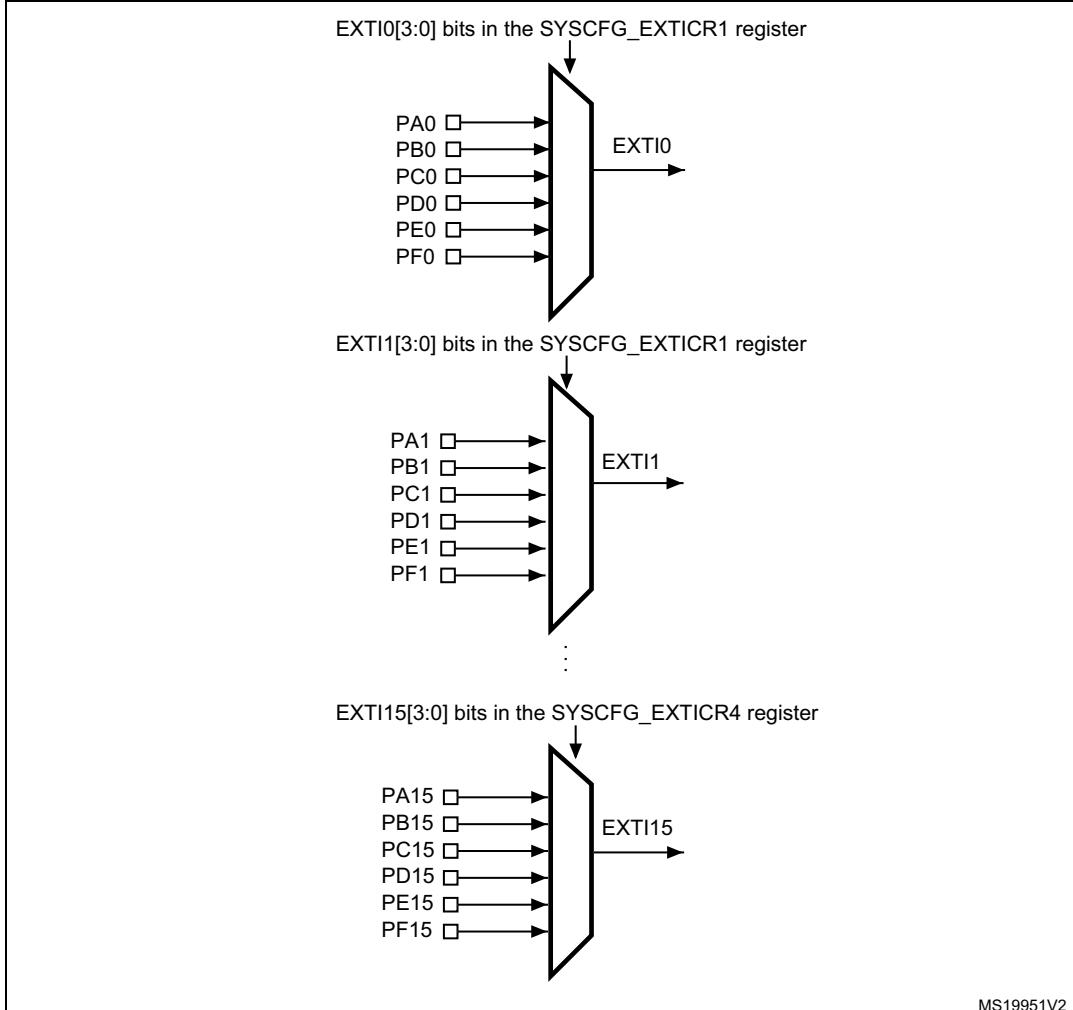
Any of the external lines can be configured as software interrupt/event lines. The following is the procedure to generate a software interrupt.

- Configure the corresponding mask bit (EXTI\_IMR, EXTI\_EMR)
- Set the required bit of the software interrupt register (EXTI\_SWIER)

### 11.2.5 External and internal interrupt/event line mapping

The GPIOs are connected to the 16 external interrupt/event lines in the following manner:

**Figure 24. External interrupt/event GPIO mapping**



MS19951V2

The remaining lines are connected as follow:

- EXTI line 16 is connected to the PVD output
- EXTI line 17 is connected to the RTC Alarm event
- EXTI line 18 is connected to the internal USB wake-up event
- EXTI line 19 is connected to the RTC Tamper andTimeStamp events
- EXTI line 20 is connected to the RTC Wake-up event (available only on STM32F07x and STM32F09x devices)
- EXTI line 21 is connected to the Comparator 1 output
- EXTI line 22 is connected to the Comparator 2 output
- EXTI line 23 is connected to the internal I2C1 wake-up event
- EXTI line 24 is reserved (internally held low)
- EXTI line 25 is connected to the internal USART1 wake-up event
- EXTI line 26 is connected to the internal USART2 wake-up event (available only on STM32F07x and STM32F09x devices)
- EXTI line 27 is connected to the internal CEC wake-up event
- EXTI line 28 is connected to the internal USART3 wake-up event (available only on STM32F09x devices)
- EXTI line 29 is reserved (internally held low)
- EXTI line 30 is reserved (internally held low)
- EXTI line 31 is connected to the  $V_{DDIO2}$  supply comparator output (available only on STM32F04x, STM32F07x and STM32F09x devices)

*Note:* *EXTI lines which are reserved or not used on some devices are considered as internal.*

## 11.3 EXTI registers

Refer to [Section 1.2 on page 42](#) for a list of abbreviations used in register descriptions.

The peripheral registers have to be accessed by words (32-bit).

### 11.3.1 Interrupt mask register (EXTI\_IMR)

Address offset: 0x00

Reset value: 0x0FF4 0000 (STM32F03x devices)  
 0x7FF4 0000 (STM32F04x devices)  
 0x0F94 0000 (STM32F05x devices)  
 0x7F84 0000 (STM32F07x and STM32F09x devices)

*Note:* *The reset value for the internal lines is set to '1' in order to enable the interrupt by default.*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IM31	IM30	IM29	IM28	IM27	IM26	IM25	IM24	IM23	IM22	IM21	IM20	IM19	IM18	IM17	IM16
rw															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IM15	IM14	IM13	IM12	IM11	IM10	IM9	IM8	IM7	IM6	IM5	IM4	IM3	IM2	IM1	IMO
rw															

Bits 31:0 **IMx**: Interrupt Mask on line x (x = 31 to 0)

0: Interrupt request from Line x is masked

1: Interrupt request from Line x is not masked

### 11.3.2 Event mask register (EXTI\_EMR)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EM31	EM30	EM29	EM28	EM27	EM26	EM25	EM24	EM23	EM22	EM21	EM20	EM19	EM18	EM17	EM16
rw															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EM15	EM14	EM13	EM12	EM11	EM10	EM9	EM8	EM7	EM6	EM5	EM4	EM3	EM2	EM1	EM0
rw															

Bits 31:0 **EMx**: Event mask on line x (x = 31 to 0)

0: Event request from Line x is masked

1: Event request from Line x is not masked

### 11.3.3 Rising trigger selection register (EXTI\_RTSR)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RT31	Res.	RT22	RT21	RT20	RT19	Res.	RT17	RT16							
rw									rw	rw	rw	rw		rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RT15	RT14	RT13	RT12	RT11	RT10	RT9	RT8	RT7	RT6	RT5	RT4	RT3	RT2	RT1	RT0
rw															

Bit 31 **RT31**: Rising trigger event configuration bit of line 31

0: Rising trigger disabled (for Event and Interrupt) for input line  
1: Rising trigger enabled (for Event and Interrupt) for input line.

Bits 30:23 Reserved, must be kept at reset value.

Bits 22:19 **RTx**: Rising trigger event configuration bit of line x (x = 22 to 19)

0: Rising trigger disabled (for Event and Interrupt) for input line  
1: Rising trigger enabled (for Event and Interrupt) for input line.

Bit 18 Reserved, must be kept at reset value.

Bits 17:0 **RTx**: Rising trigger event configuration bit of line x (x = 17 to 0)

0: Rising trigger disabled (for Event and Interrupt) for input line  
1: Rising trigger enabled (for Event and Interrupt) for input line.

**Note:** The external wake-up lines are edge triggered. No glitches must be generated on these lines. If a rising edge on an external interrupt line occurs during a write operation to the EXTI\_FTSR register, the pending bit is not set.

Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

### 11.3.4 Falling trigger selection register (EXTI\_FTSR)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FT31	Res.	FT22	FT21	FT20	FT19	Res.	FT17	FT16							
rw									rw	rw	rw	rw		rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FT15	FT14	FT13	FT12	FT11	FT10	FT9	FT8	FT7	FT6	FT5	FT4	FT3	FT2	FT1	FT0
rw															

Bit 31 **FT31**: Falling trigger event configuration bit of line 31

- 0: Falling trigger disabled (for Event and Interrupt) for input line
- 1: Falling trigger enabled (for Event and Interrupt) for input line.

Bits 30:23 Reserved, must be kept at reset value.

Bits 22:19 **FTx**: Falling trigger event configuration bit of line x (x = 22 to 19)

- 0: Falling trigger disabled (for Event and Interrupt) for input line.
- 1: Falling trigger enabled (for Event and Interrupt) for input line.

Bit 18 Reserved, must be kept at reset value.

Bits 17:0 **FTx**: Falling trigger event configuration bit of line x (x = 17 to 0)

- 0: Falling trigger disabled (for Event and Interrupt) for input line.
- 1: Falling trigger enabled (for Event and Interrupt) for input line.

**Note:** The external wake-up lines are edge triggered. No glitches must be generated on these lines. If a falling edge on an external interrupt line occurs during a write operation to the EXTI\_FTSR register, the pending bit is not set.

Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

### 11.3.5 Software interrupt event register (EXTI\_SWIER)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SWI31	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SWI22	SWI21	SWI20	SWI19	Res.	SWI17	SWI16
rw									rw	rw	rw	rw		rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWI15	SWI14	SWI13	SWI12	SWI11	SWI10	SWI9	SWI8	SWI7	SWI6	SWI5	SWI4	SWI3	SWI2	SWI1	SWI0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **SWI31:** Software interrupt on line 31

If the interrupt is enabled on this line in the EXTI\_IMR, writing a '1' to this bit when it is at '0' sets the corresponding pending bit in EXTI\_PR resulting in an interrupt request generation.

This bit is cleared by clearing the corresponding bit of EXTI\_PR (by writing a '1' to the bit)

Bits 30:23 Reserved, must be kept at reset value.

Bits 22:19 **SWIx:** Software interrupt on line x (x = 22 to 19)

If the interrupt is enabled on this line in the EXTI\_IMR, writing a '1' to this bit when it is at '0' sets the corresponding pending bit in EXTI\_PR resulting in an interrupt request generation.

This bit is cleared by clearing the corresponding bit of EXTI\_PR (by writing a '1' to the bit)

Bit 18 Reserved, must be kept at reset value.

Bits 17:0 **SWIx:** Software interrupt on line x (x = 17 to 0)

If the interrupt is enabled on this line in the EXTI\_IMR, writing a '1' to this bit when it is at '0' sets the corresponding pending bit in EXTI\_PR resulting in an interrupt request generation.

This bit is cleared by clearing the corresponding bit of EXTI\_PR (by writing a '1' to the bit).

### 11.3.6 Pending register (EXTI\_PR)

Address offset: 0x14

Reset value: undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PIF31	Res.	PIF22	PIF21	PIF20	PIF19	Res.	PIF17	PIF16							
rc_w1									rc_w1	rc_w1	rc_w1	rc_w1		rc_w1	rc_w1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PIF15	PIF14	PIF13	PIF12	PIF11	PIF10	PIF9	PIF8	PIF7	PIF6	PIF5	PIF4	PIF3	PIF2	PIF1	PIF0
rc_w1															

Bit 31 **PIF31:** Pending bit on line 31

0: no trigger request occurred

1: selected trigger request occurred

This bit is set when the selected edge event arrives on the external interrupt line. This bit is cleared by writing a 1 to the bit.

Bits 30:23 Reserved, must be kept at reset value.

Bits 22:19 **PIFx:** Pending bit on line x (x = 22 to 19)

0: no trigger request occurred

1: selected trigger request occurred

This bit is set when the selected edge event arrives on the external interrupt line. This bit is cleared by writing a 1 to the bit.

Bit 18 Reserved, must be kept at reset value.

Bits 17:0 **PIFx:** Pending bit on line x (x = 17 to 0)

0: no trigger request occurred

1: selected trigger request occurred

This bit is set when the selected edge event arrives on the external interrupt line. This bit is cleared by writing a 1 to the bit.

### 11.3.7 EXTI register map

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

**Table 37. External interrupt/event controller register map and reset values**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	<b>EXTI_IMR</b>																																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x04	<b>EXTI_EMR</b>																																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x08	<b>EXTI_RTSR</b>	RT31	0	RT31	0	RT23	0	RT23	0	RT22	0	RT22	0	RT21	0	RT21	0	RT20	0	RT20	0	RT19	0	RT19	0	RT19	0	RT19	0	RT19	0	RT19	0
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x0C	<b>EXTI_FTSR</b>	FT31	0	FT31	0	FT23	0	FT23	0	FT22	0	FT22	0	FT21	0	FT21	0	FT20	0	FT20	0	FT19	0	FT19	0	FT19	0	FT19	0	FT19	0	FT19	0
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x10	<b>EXTI_SWIER</b>	SWI31	0	SWI31	0	SWI23	0	SWI23	0	SWI22	0	SWI22	0	SWI21	0	SWI21	0	SWI20	0	SWI20	0	SWI19	0	SWI19	0	SWI19	0	SWI19	0	SWI19	0	SWI19	0
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x14	<b>EXTI_PR</b>	PIF31	0	PIF31	0	PIF23	0	PIF23	0	PIF22	0	PIF22	0	PIF21	0	PIF21	0	PIF20	0	PIF20	0	PIF19	0	PIF19	0	PIF19	0	PIF19	0	PIF19	0	PIF19	0
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 12 Cyclic redundancy check calculation unit (CRC)

### 12.1 Introduction

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

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

### 12.2 CRC main features

- Uses CRC-32 (Ethernet) polynomial: 0x4C11DB7  

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$
- Alternatively, uses fully programmable polynomial with programmable size (7, 8, 16, 32 bits)
- Handles 8-, 16-, 32-bit data size
- Programmable CRC initial value
- Single input/output 32-bit data register
- Input buffer to avoid bus stall during calculation
- CRC computation done in 4 AHB clock cycles (HCLK) for the 32-bit data size
- General-purpose 8-bit register (can be used for temporary storage)
- Reversibility option on I/O data
- Accessed through AHB slave peripheral by 32-bit words only, with the exception of CRC\_DR register that can be accessed by words, right-aligned half-words and right-aligned bytes

### 12.3 CRC implementation

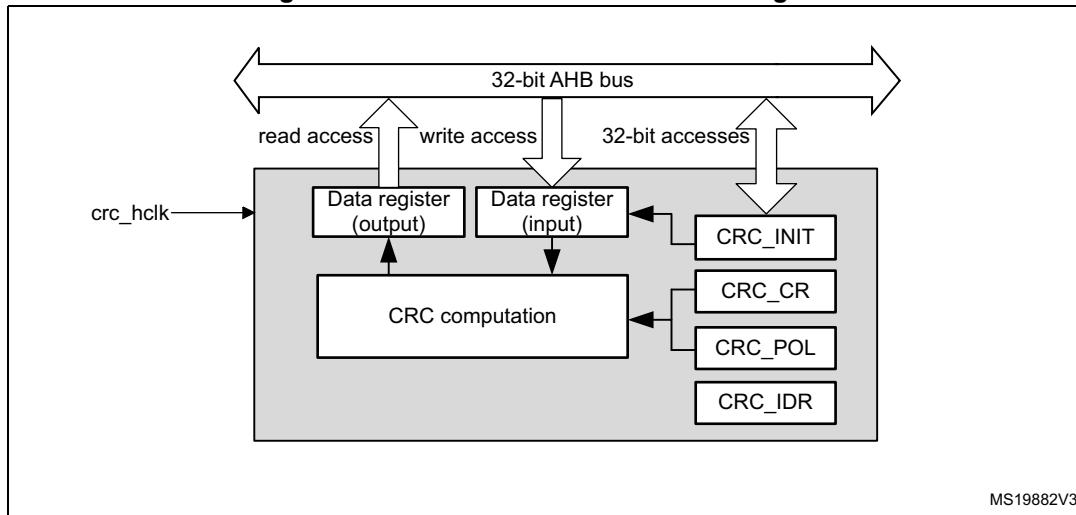
Table 38. CRC features

Modes/features	STM32F07x and STM32F09x	STM32F0x other than STM32F07x and STM32F09x
CRC-32 polynomial	X	X
Fully programmable polynomial	X	-
Data swapping	-	-

## 12.4 CRC functional description

### 12.4.1 CRC block diagram

Figure 25. CRC calculation unit block diagram



### 12.4.2 CRC internal signals

Table 39. CRC internal input/output signals

Signal name	Signal type	Description
crc_hclk	Digital input	AHB clock

### 12.4.3 CRC operation

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

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

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

The duration of the computation depends on data width:

- 4 AHB clock cycles for 32 bits
- 2 AHB clock cycles for 16 bits
- 1 AHB clock cycles for 8 bits

An input buffer allows a second data to be immediately written without waiting for any wait states due to the previous CRC calculation.

The data size can be dynamically adjusted to minimize the number of write accesses for a given number of bytes. For instance, a CRC for 5 bytes can be computed with a word write followed by a byte write.

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

For example, 0x1A2B3C4D input data are used for CRC calculation as:

- 0x58D43CB2 with bit-reversal done by byte
- 0xD458B23C with bit-reversal done by half-word
- 0xB23CD458 with bit-reversal done on the full word

The output data can also be reversed by setting the REV\_OUT bit in the CRC\_CR register.

The operation is done at bit level. For example, 0x11223344 output data are converted to 0x22CC4488.

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

The initial CRC value can be programmed with the CRC\_INIT register. The CRC\_DR register is automatically initialized upon CRC\_INIT register write access.

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

## Polynomial programmability

The polynomial coefficients are fully programmable through the CRC\_POL register, and the polynomial size can be configured to be 7, 8, 16 or 32 bits by programming the POLYSIZE[1:0] bits in the CRC\_CR register. Even polynomials are not supported.

*Note:* The type of an even polynomial is  $X+X^2+\dots+X^n$ , while the type of an odd polynomial is  $1+X+X^2+\dots+X^n$ .

If the CRC data is less than 32-bit, its value can be read from the least significant bits of the CRC\_DR register.

To obtain a reliable CRC calculation, the change on-fly of the polynomial value or size can not be performed during a CRC calculation. As a result, if a CRC calculation is ongoing, the application must either reset it or perform a CRC\_DR read before changing the polynomial.

The default polynomial value is the CRC-32 (Ethernet) polynomial: 0x4C11DB7.

## 12.5 CRC registers

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

### 12.5.1 CRC data register (CRC\_DR)

Address offset: 0x00

Reset value: 0xFFFF FFFF

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

Bits 31:0 **DR[31:0]**: Data register bits

This register is used to write new data to the CRC calculator.

It holds the previous CRC calculation result when it is read.

If the data size is less than 32 bits, the least significant bits are used to write/read the correct value.

### 12.5.2 CRC independent data register (CRC\_IDR)

Address offset: 0x04

Reset value: 0x0000 0000

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

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

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

These bits can be used as a temporary storage location for one byte.

This register is not affected by CRC resets generated by the RESET bit in the CRC\_CR register

### 12.5.3 CRC control register (CRC\_CR)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	REV_OUT	REV_IN[1:0]	POLYSIZE[1:0]	Res.	Res.	RES									
								rw	rw	rw	rw	rw			rs

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

Bit 7 **REV\_OUT**: Reverse output data

This bit controls the reversal of the bit order of the output data.

0: Bit order not affected

1: Bit-reversed output format

Bits 6:5 **REV\_IN[1:0]**: Reverse input data

This bitfield controls the reversal of the bit order of the input data

00: Bit order not affected

01: Bit reversal done by byte

10: Bit reversal done by half-word

11: Bit reversal done by word

Bits 4:3 **POLYSIZE[1:0]**: Polynomial size

These bits control the size of the polynomial. This bitfield is reserved if the fully programmable polynomial is not available (refer to [Section 12.3: CRC implementation](#)).

00: 32 bit polynomial

01: 16 bit polynomial

10: 8 bit polynomial

11: 7 bit polynomial

Bits 2:1 Reserved, must be kept at reset value.

Bit 0 **RESET**: RESET bit

This bit is set by software to reset the CRC calculation unit and set the data register to the value stored in the CRC\_INIT register. This bit can only be set, it is automatically cleared by hardware

### 12.5.4 CRC initial value (CRC\_INIT)

Address offset: 0x10

Reset value: 0xFFFF FFFF

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

Bits 31:0 **CRC\_INIT[31:0]**: Programmable initial CRC value

This register is used to write the CRC initial value.

### 12.5.5 CRC polynomial (CRC\_POL)

Address offset: 0x14

Reset value: 0x04C1 1DB7

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

Bits 31:0 **POL[31:0]**: Programmable polynomial (for STM32F07x and STM32F09x)

This register is used to write the coefficients of the polynomial to be used for CRC calculation.

If the polynomial size is less than 32 bits, the least significant bits have to be used to program the correct value.

For STM32F03x, STM32F04x, and STM32F05x, the field is read-only.

## 12.5.6 CRC register map

Table 40. CRC register map and reset values

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	<b>CRC_DR</b>																																
		Res.																															
0x04	<b>CRC_IDR</b>																																
		Res.																															
0x08	<b>CRC_CR</b>																																
		Res.																															
0x10	<b>CRC_INIT</b>																																
		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
0x14	<b>CRC_POL</b>																																
		0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 13 Analog-to-digital converter (ADC)

### 13.1 Introduction

The 12-bit ADC is a successive approximation analog-to-digital converter. It has up to 19 multiplexed channels allowing it to measure signals from 16 external and 3 internal sources. A/D conversion of the various channels can be performed in single, continuous, scan or discontinuous mode. The result of the ADC is stored in a left-aligned or right-aligned 16-bit data register.

The analog watchdog feature allows the application to detect if the input voltage goes outside the user-defined higher or lower thresholds.

An efficient low-power mode is implemented to allow very low consumption at low frequency.

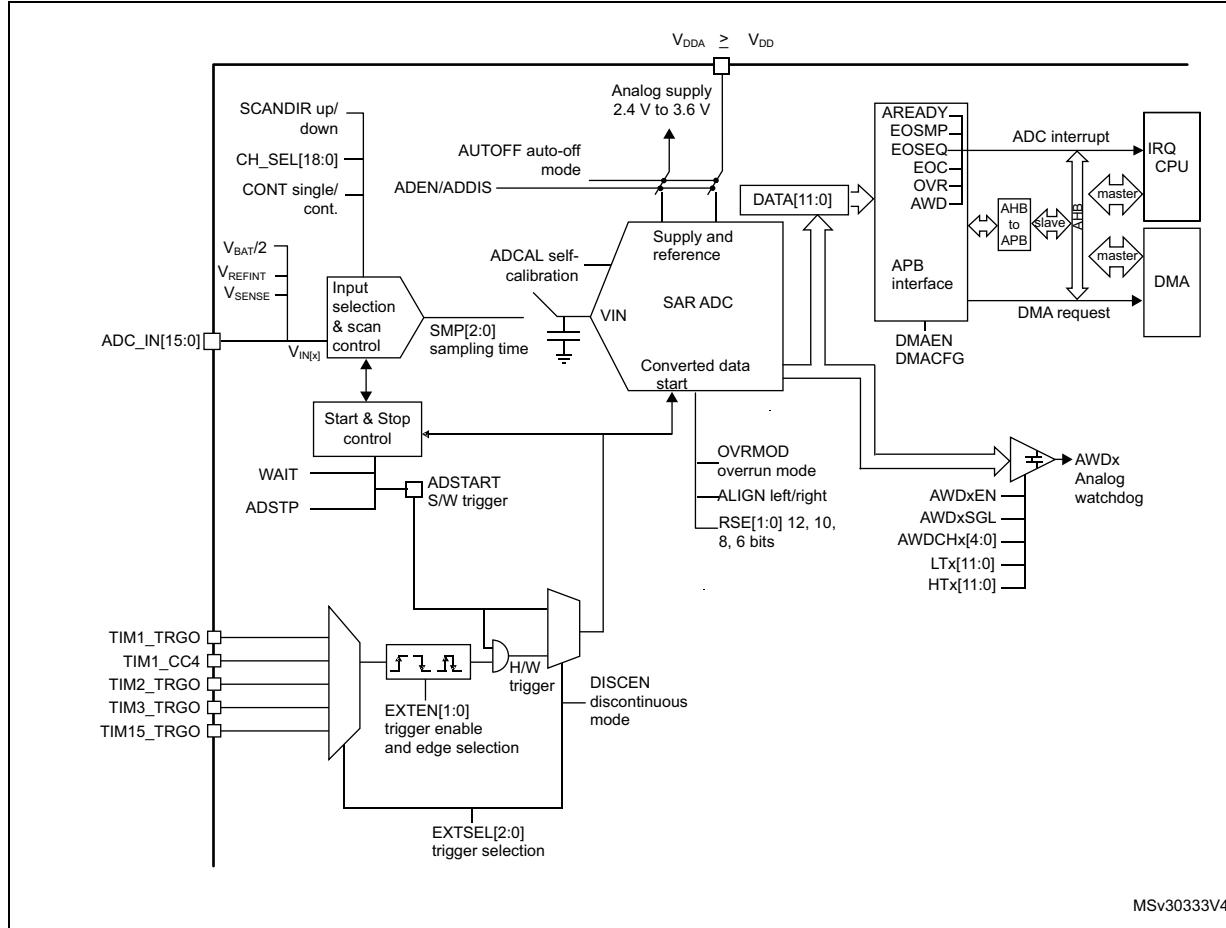
## 13.2 ADC main features

- High performance
  - 12-bit, 10-bit, 8-bit or 6-bit configurable resolution
  - ADC conversion time: 1.0  $\mu$ s for 12-bit resolution (1 MHz), 0.93  $\mu$ s conversion time for 10-bit resolution, faster conversion times can be obtained by lowering resolution.
  - Self-calibration
  - Programmable sampling time
  - Data alignment with built-in data coherency
  - DMA support
- Low-power
  - The application can reduce PCLK frequency for low-power operation while still keeping optimum ADC performance. For example, 1.0  $\mu$ s conversion time is kept, whatever the PCLK frequency
  - Wait mode: prevents ADC overrun in applications with low PCLK frequency
  - Auto off mode: ADC is automatically powered off except during the active conversion phase. This dramatically reduces the power consumption of the ADC.
- Analog input channels
  - 16 external analog inputs
  - 1 channel for internal temperature sensor ( $V_{SENSE}$ )
  - 1 channel for internal reference voltage ( $V_{REFINT}$ )
  - 1 channel for monitoring external  $V_{BAT}$  power supply pin
- Start-of-conversion can be initiated:
  - By software
  - By hardware triggers with configurable polarity (timer events)
- Conversion modes
  - Can convert a single channel or can scan a sequence of channels.
  - Single mode converts selected inputs once per trigger
  - Continuous mode converts selected inputs continuously
  - Discontinuous mode
- Interrupt generation at the end of sampling, end of conversion, end of sequence conversion, and in case of analog watchdog or overrun events
- Analog watchdog
- ADC input range:  $V_{SSA} \leq V_{IN} \leq V_{DDA}$

### 13.3 ADC functional description

Figure 26 shows the ADC block diagram and Table 41 gives the ADC pin description.

Figure 26. ADC block diagram



MSv30333V4

#### 13.3.1 ADC pins and internal signals

Table 41. ADC input/output pins

Name	Signal type	Remarks
VDDA	Input, analog power supply	Analog power supply and positive reference voltage for the ADC
VSSA	Input, analog supply ground	Ground for analog power supply
ADC_INx	Analog input signals	16 external analog input channels

**Table 42. ADC internal input/output signals**

Internal signal name	Signal type	Description
$V_{IN[X]}$	Analog Input channels	Connected either to internal channels or to ADC_IN/external channels
TRGx	Input	ADC conversion triggers
$V_{SENSE}$	Input	Internal temperature sensor output voltage
$V_{REFINT}$	Input	Internal voltage reference output voltage
$V_{BAT/2}$	Input	VBAT pin input voltage divided by 2
ADC_AWDx_OUT	Output	Internal analog watchdog output signal connected to on-chip timers (x = Analog watchdog number = 1)

**Table 43. External triggers**

Name	Source	EXTSEL[2:0]
TRG0	TIM1_TRGO	000
TRG1	TIM1_CC4	001
TRG2	TIM2_TRGO	010
TRG3	TIM3_TRGO	011
TRG4	TIM15_TRGO	100
TRG5	Reserved	101
TRG6	Reserved	110
TRG7	Reserved	111

### 13.3.2 Calibration (ADCAL)

The ADC has a calibration feature. During the calibration phase, the ADC calculates a calibration factor which is internally applied to the ADC until the next ADC power-off. The application must not use the ADC during calibration and must wait until it is complete.

Calibration should be performed before starting A/D conversion. It removes the offset error which may vary from chip to chip due to process variation.

The calibration is initiated by software by setting ADCAL bit to 1. It can only be initiated when the ADC is disabled (when ADEN = 0). ADCAL bit stays at 1 during the whole calibration sequence. It is then cleared by hardware as soon the calibration completes. After this, the calibration factor can be read from the ADC\_DR register (from bits 6 to 0).

The internal analog calibration is kept if the ADC is disabled (ADEN = 0). When the ADC operating conditions change ( $V_{DDA}$  changes are the main contributor to ADC offset variations and temperature change to a lesser extend), it is recommended to re-run a calibration cycle.

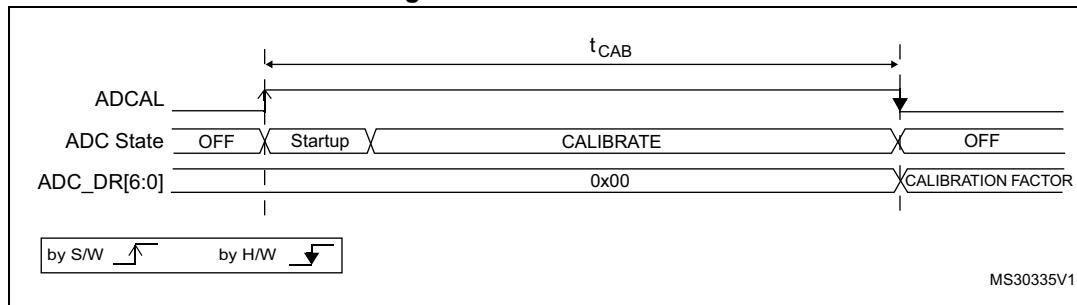
The calibration factor is lost each time power is removed from the ADC (for example when the product enters Standby mode).

### Calibration software procedure

1. Ensure that ADEN = 0 and DMAEN = 0.
2. Set ADCAL = 1.
3. Wait until ADCAL = 0.
4. The calibration factor can be read from bits 6:0 of ADC\_DR.

For code example refer to the Appendix section [A.7.1: ADC calibration code example](#).

**Figure 27. ADC calibration**



### 13.3.3 ADC on-off control (ADEN, ADDIS, ADRDY)

At power-up, the ADC is disabled and put in power-down mode (ADEN = 0).

As shown in [Figure 28](#), the ADC needs a stabilization time of  $t_{STAB}$  before it starts converting accurately.

Two control bits are used to enable or disable the ADC:

- Set ADEN = 1 to enable the ADC. The ADRDY flag is set as soon as the ADC is ready for operation.
- Set ADDIS = 1 to disable the ADC and put the ADC in power down mode. The ADEN and ADDIS bits are then automatically cleared by hardware as soon as the ADC is fully disabled.

Conversion can then start either by setting ADSTART to 1 (refer to [Section 13.4: Conversion on external trigger and trigger polarity \(EXTSEL, EXTEN\) on page 244](#)) or when an external trigger event occurs if triggers are enabled.

Follow this procedure to enable the ADC:

1. Clear the ADRDY bit in ADC\_ISR register by programming this bit to 1.
2. Set ADEN = 1 in the ADC\_CR register.
3. Wait until ADRDY = 1 in the ADC\_ISR register and continue to write ADEN = 1 (ADRDY is set after the ADC startup time). This can be handled by interrupt if the interrupt is enabled by setting the ADRDYIE bit in the ADC\_IER register.

For code example refer to the Appendix section [A.7.2: ADC enable sequence code example](#).

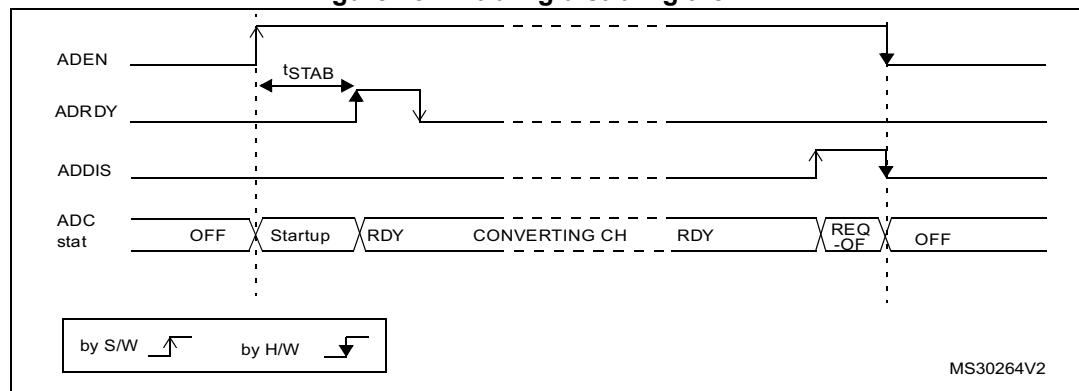
Follow this procedure to disable the ADC:

1. Check that ADSTART = 0 in the ADC\_CR register to ensure that no conversion is ongoing. If required, stop any ongoing conversion by writing 1 to the ADSTP bit in the ADC\_CR register and waiting until this bit is read at 0.
2. Set ADDIS = 1 in the ADC\_CR register.
3. If required by the application, wait until ADEN = 0 in the ADC\_CR register, indicating that the ADC is fully disabled (ADDIS is automatically reset once ADEN = 0).
4. Clear the ADRDY bit in ADC\_ISR register by programming this bit to 1 (optional).

For code example refer to the Appendix section [A.7.3: ADC disable sequence code example](#).

**Caution:** ADEN bit cannot be set when ADCAL = 1 and during four ADC clock cycles after the ADCAL bit is cleared by hardware (end of calibration).

**Figure 28. Enabling/disabling the ADC**

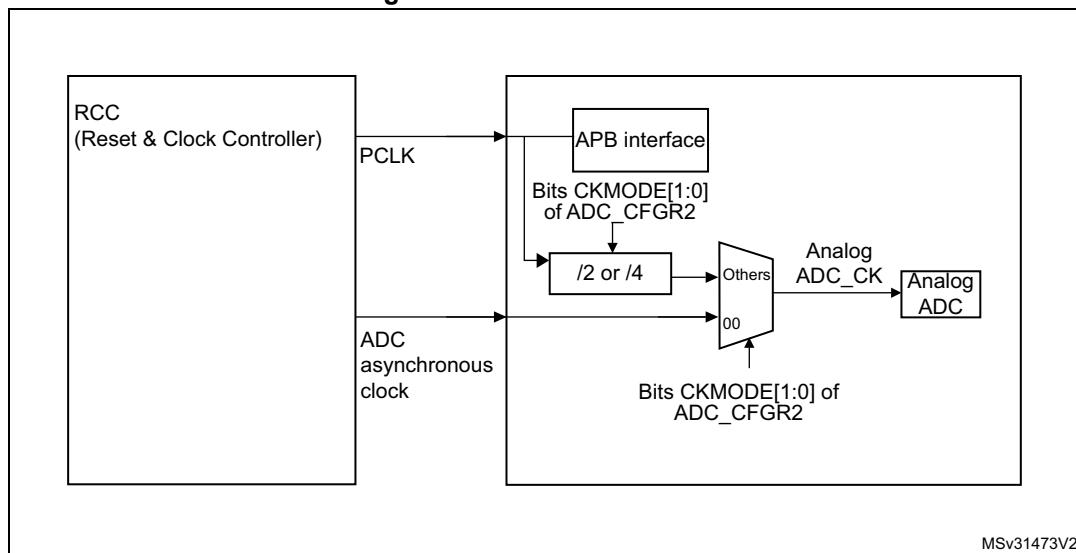


**Note:** In Auto-off mode ( $AUTOFF = 1$ ) the power-on/off phases are performed automatically, by hardware and the ADRDY flag is not set.

### 13.3.4 ADC clock (CKMODE)

The ADC has a dual clock-domain architecture, so that the ADC can be fed with a clock (ADC asynchronous clock) independent from the APB clock (PCLK).

**Figure 29. ADC clock scheme**



1. Refer to *Section Reset and clock control (RCC)* for how the PCLK clock and ADC asynchronous clock are enabled.

The input clock of the analog ADC can be selected between two different clock sources (see [Figure 29: ADC clock scheme](#) to see how the PCLK clock and the ADC asynchronous clock are enabled):

- a) The ADC clock can be a specific clock source, named “ADC asynchronous clock” which is independent and asynchronous with the APB clock.  
Refer to RCC Section for more information on generating this clock source.  
To select this scheme, bits CKMODE[1:0] of the ADC\_CFGR2 register must be reset.  
For code example refer to the Appendix section [A.7.4: ADC clock selection code example](#).
- b) The ADC clock can be derived from the APB clock of the ADC bus interface, divided by a programmable factor (1, 2 or 4) according to bits CKMODE[1:0].  
To select this scheme, bits CKMODE[1:0] of the ADC\_CFGR2 register must be different from “00”.

Option a) has the advantage of reaching the maximum ADC clock frequency whatever the APB clock scheme selected.

Option b) has the advantage of bypassing the clock domain resynchronizations. This can be useful when the ADC is triggered by a timer and if the application requires that the ADC is precisely triggered without any uncertainty (otherwise, an uncertainty of the trigger instant is added by the resynchronizations between the two clock domains).

**Table 44. Latency between trigger and start of conversion<sup>(1)</sup>**

ADC clock source	CKMODE[1:0]	Latency between the trigger event and the start of conversion
Dedicated 14MHz clock	00	Latency is not deterministic (jitter)
PCLK divided by 2	01	Latency is deterministic (no jitter) and equal to 2.75 ADC clock cycles
PCLK divided by 4	10	Latency is deterministic (no jitter) and equal to 2.625 ADC clock cycles

1. Refer to the device datasheet for the maximum ADC\_CLK frequency.

### 13.3.5 Configuring the ADC

The software must write the ADCAL and ADEN bits in the ADC\_CR register only when the ADC is disabled (ADEN cleared).

The software must only write to the ADSTART and ADDIS bits in the ADC\_CR register only if the ADC is enabled and there is no pending request to disable the ADC (ADEN = 1 and ADDIS = 0).

For all the other control bits in the ADC\_IER, ADC\_CFGRI, ADC\_SMPR, ADC\_TR, ADC\_CHSELR and ADC\_CCR registers, refer to the description of the corresponding control bit in [Section 13.11: ADC registers](#).

The software must only write to the ADSTP bit in the ADC\_CR register if the ADC is enabled (and possibly converting) and there is no pending request to disable the ADC (ADSTART = 1 and ADDIS = 0).

Note:

*There is no hardware protection preventing software from making write operations forbidden by the above rules. If such a forbidden write access occurs, the ADC may enter an undefined state. To recover correct operation in this case, the ADC must be disabled (clear ADEN = 0 and all the bits in the ADC\_CR register).*

### 13.3.6 Channel selection (CHSEL, SCANDIR)

There are up to 19 multiplexed channels:

- 16 analog inputs from GPIO pins (ADC\_INx)
- 3 internal analog inputs (temperature sensor, internal reference voltage, V<sub>BAT</sub> channel)

It is possible to convert a single channel or a sequence of channels.

The sequence of the channels to be converted can be programmed in the ADC\_CHSELR channel selection register: each analog input channel has a dedicated selection bit (CHSELx).

The order in which the channels are scanned can be configured by programming the bit SCANDIR bit in the ADC\_CFGR1 register:

- SCANDIR = 0: forward scan Channel 0 to Channel 18
- SCANDIR = 1: backward scan Channel 18 to Channel 0

#### Temperature sensor, V<sub>REFINT</sub> and V<sub>BAT</sub> internal channels

The temperature sensor is connected to channel ADC V<sub>IN</sub>[16].

The internal voltage reference  $V_{REFINT}$  is connected to channel ADC  $V_{IN}[17]$ .

$V_{BAT}$  channel is connected to ADC  $V_{IN}[18]$  channel.

### 13.3.7 Programmable sampling time (SMP)

Before starting a conversion, the ADC needs to establish a direct connection between the voltage source to be measured and the embedded sampling capacitor of the ADC. This sampling time must be enough for the input voltage source to charge the sample and hold capacitor to the input voltage level.

Having a programmable sampling time allows the conversion speed to be trimmed according to the input resistance of the input voltage source.

The ADC samples the input voltage for a number of ADC clock cycles that can be modified using the SMP[2:0] bits in the ADC\_SMPR register.

This programmable sampling time is common to all channels. If required by the application, the software can change and adapt this sampling time between each conversions.

The total conversion time is calculated as follows:

$$t_{CONV} = \text{Sampling time} + 12.5 \times \text{ADC clock cycles}$$

Example:

With  $\text{ADC\_CLK} = 14 \text{ MHz}$  and a sampling time of 1.5 ADC clock cycles:

$$t_{CONV} = 1.5 + 12.5 = 14 \text{ ADC clock cycles} = 1 \mu\text{s}$$

The ADC indicates the end of the sampling phase by setting the EOSMP flag.

### 13.3.8 Single conversion mode (CONT = 0)

In Single conversion mode, the ADC performs a single sequence of conversions, converting all the channels once. This mode is selected when CONT = 0 in the ADC\_CFGR1 register. Conversion is started by either:

- Setting the ADSTART bit in the ADC\_CR register
- Hardware trigger event

Inside the sequence, after each conversion is complete:

- The converted data are stored in the 16-bit ADC\_DR register
- The EOC (end of conversion) flag is set
- An interrupt is generated if the EOCIE bit is set

After the sequence of conversions is complete:

- The EOS (end of sequence) flag is set
- An interrupt is generated if the EOSIE bit is set

Then the ADC stops until a new external trigger event occurs or the ADSTART bit is set again.

*Note:* To convert a single channel, program a sequence with a length of 1.

### 13.3.9 Continuous conversion mode (CONT = 1)

In continuous conversion mode, when a software or hardware trigger event occurs, the ADC performs a sequence of conversions, converting all the channels once and then automatically re-starts and continuously performs the same sequence of conversions. This mode is selected when CONT = 1 in the ADC\_CFGR1 register. Conversion is started by either:

- Setting the ADSTART bit in the ADC\_CR register
- Hardware trigger event

Inside the sequence, after each conversion is complete:

- The converted data are stored in the 16-bit ADC\_DR register
- The EOC (end of conversion) flag is set
- An interrupt is generated if the EOCIE bit is set

After the sequence of conversions is complete:

- The EOS (end of sequence) flag is set
- An interrupt is generated if the EOSIE bit is set

Then, a new sequence restarts immediately and the ADC continuously repeats the conversion sequence.

**Note:** *To convert a single channel, program a sequence with a length of 1.*

*It is not possible to have both discontinuous mode and continuous mode enabled: it is forbidden to set both bits DISCEN = 1 and CONT = 1.*

### 13.3.10 Starting conversions (ADSTART)

Software starts ADC conversions by setting ADSTART = 1.

When ADSTART is set, the conversion:

- Starts immediately if EXTN = 00 (software trigger)
- At the next active edge of the selected hardware trigger if EXTN ≠ 00

The ADSTART bit is also used to indicate whether an ADC operation is currently ongoing. It is possible to re-configure the ADC while ADSTART = 0, indicating that the ADC is idle.

The ADSTART bit is cleared by hardware:

- In single mode with software trigger (CONT = 0, EXTN = 00)
  - At any end of conversion sequence (EOS = 1)
- In discontinuous mode with software trigger (CONT = 0, DISCEN = 1, EXTN = 00)
  - At end of conversion (EOC = 1)
- In all cases (CONT = x, EXTN = XX)
  - After execution of the ADSTP procedure invoked by software (see [Section 13.3.12: Stopping an ongoing conversion \(ADSTP\) on page 244](#))

**Note:** *In continuous mode (CONT = 1), the ADSTART bit is not cleared by hardware when the EOS flag is set because the sequence is automatically relaunched.*

*When hardware trigger is selected in single mode (CONT = 0 and EXTN = 01), ADSTART is not cleared by hardware when the EOS flag is set (except if DMAEN = 1 and DMACFG = 0 in which case ADSTART is cleared at end of the DMA transfer). This avoids*

the need for software having to set the ADSTART bit again and ensures the next trigger event is not missed.

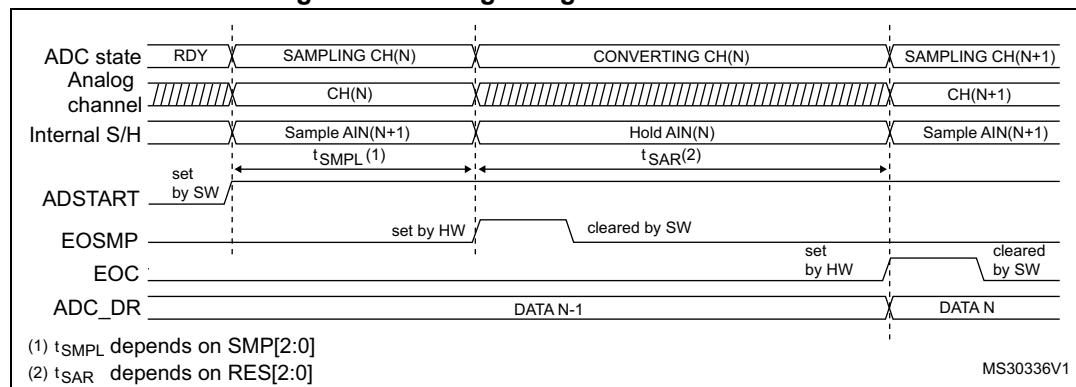
### 13.3.11 Timings

The elapsed time between the start of a conversion and the end of conversion is the sum of the configured sampling time plus the successive approximation time depending on data resolution:

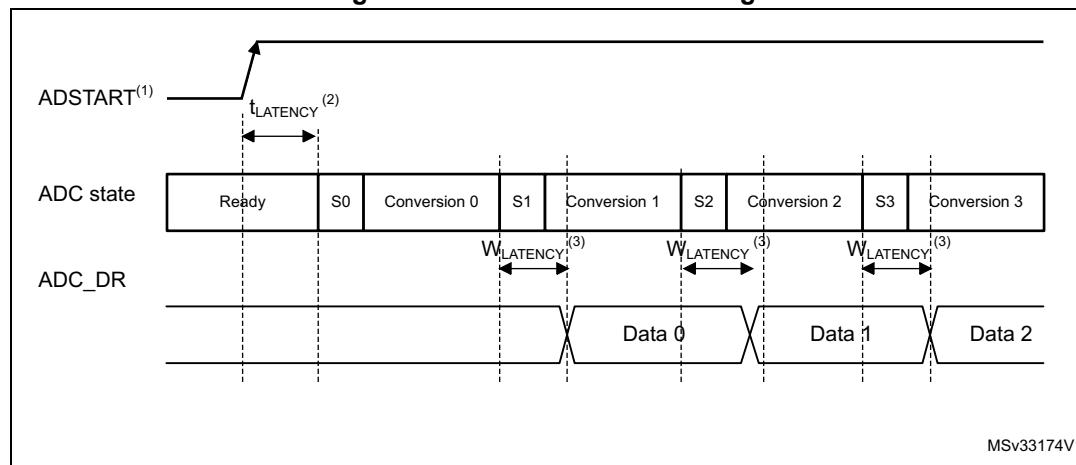
$$t_{\text{CONV}} = t_{\text{SMPL}} + t_{\text{SAR}} = [1.5 \text{ } \mu\text{s}_{\text{min}} + 12.5 \text{ } \mu\text{s}_{\text{12bit}}] \times t_{\text{ADC\_CLK}}$$

$$t_{\text{CONV}} = t_{\text{SMPL}} + t_{\text{SAR}} = 107.1 \text{ ns}_{\text{min}} + 892.8 \text{ ns}_{\text{12bit}} = 1 \mu\text{s}_{\text{min}} \text{ (for } f_{\text{ADC\_CLK}} = 14 \text{ MHz)}$$

**Figure 30. Analog to digital conversion time**



**Figure 31. ADC conversion timings**



1. EXTEN = 00 or EXTEN ≠ 00
2. Trigger latency (refer to datasheet for more details)
3. ADC\_DR register write latency (refer to datasheet for more details)

### 13.3.12 Stopping an ongoing conversion (ADSTP)

The software can decide to stop any ongoing conversions by setting ADSTP = 1 in the ADC\_CR register.

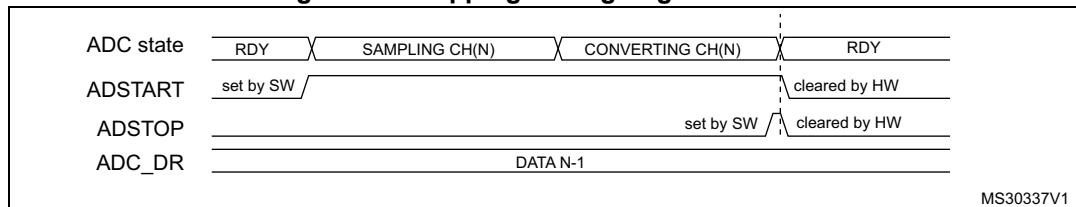
This resets the ADC operation and the ADC is idle, ready for a new operation.

When the ADSTP bit is set by software, any ongoing conversion is aborted and the result is discarded (ADC\_DR register is not updated with the current conversion).

The scan sequence is also aborted and reset (meaning that restarting the ADC would restart a new sequence).

Once this procedure is complete, the ADSTP and ADSTART bits are both cleared by hardware and the software must wait until ADSTART=0 before starting new conversions.

**Figure 32. Stopping an ongoing conversion**



## 13.4 Conversion on external trigger and trigger polarity (EXTSEL, EXTEN)

A conversion or a sequence of conversion can be triggered either by software or by an external event (for example timer capture). If the EXTEN[1:0] control bits are not equal to “0b00”, then external events are able to trigger a conversion with the selected polarity. The trigger selection is effective once software has set bit ADSTART = 1.

Any hardware triggers which occur while a conversion is ongoing are ignored.

If bit ADSTART = 0, any hardware triggers which occur are ignored.

**Table 45** provides the correspondence between the EXTEN[1:0] values and the trigger polarity.

**Table 45. Configuring the trigger polarity**

Source	EXTEN[1:0]
Trigger detection disabled	00
Detection on rising edge	01
Detection on falling edge	10
Detection on both rising and falling edges	11

**Note:** The polarity of the external trigger can be changed only when the ADC is not converting (ADSTART = 0).

The EXTSEL[2:0] control bits are used to select which of 8 possible events can trigger conversions.

Refer to [Table 43: External triggers](#) in [Section 13.3.1: ADC pins and internal signals](#) for the list of all the external triggers that can be used for regular conversion.

The software source trigger events can be generated by setting the ADSTART bit in the ADC\_CR register.

*Note:* *The trigger selection can be changed only when the ADC is not converting (ADSTART = 0).*

### 13.4.1 Discontinuous mode (DISCEN)

This mode is enabled by setting the DISCEN bit in the ADC\_CFGR1 register.

In this mode (DISCEN = 1), a hardware or software trigger event is required to start each conversion defined in the sequence. On the contrary, if DISCEN = 0, a single hardware or software trigger event successively starts all the conversions defined in the sequence.

Example:

- DISCEN = 1, channels to be converted = 0, 3, 7, 10
  - 1st trigger: channel 0 is converted and an EOC event is generated
  - 2nd trigger: channel 3 is converted and an EOC event is generated
  - 3rd trigger: channel 7 is converted and an EOC event is generated
  - 4th trigger: channel 10 is converted and both EOC and EOS events are generated.
  - 5th trigger: channel 0 is converted an EOC event is generated
  - 6th trigger: channel 3 is converted and an EOC event is generated
  - ...
- DISCEN = 0, channels to be converted = 0, 3, 7, 10
  - 1st trigger: the complete sequence is converted: channel 0, then 3, 7 and 10. Each conversion generates an EOC event and the last one also generates an EOS event.
  - Any subsequent trigger events restarts the complete sequence.

*Note:* *It is not possible to have both discontinuous mode and continuous mode enabled: it is forbidden to set both bits DISCEN = 1 and CONT = 1.*

### 13.4.2 Programmable resolution (RES) - Fast conversion mode

It is possible to obtain faster conversion times ( $t_{SAR}$ ) by reducing the ADC resolution.

The resolution can be configured to be either 12, 10, 8, or 6 bits by programming the RES[1:0] bits in the ADC\_CFGR1 register. Lower resolution allows faster conversion times for applications where high data precision is not required.

*Note:* *The RES[1:0] bit must only be changed when the ADEN bit is reset.*

The result of the conversion is always 12 bits wide and any unused LSB bits are read as zeros.

Lower resolution reduces the conversion time needed for the successive approximation steps as shown in [Table 46](#).

**Table 46.  $t_{SAR}$  timings depending on resolution**

<b>RES[1:0] bits</b>	<b><math>t_{SAR}</math> (ADC clock cycles)</b>	<b><math>t_{SAR}</math> (ns) at <math>f_{ADC} = 14</math> MHz</b>	<b><math>t_{SMPL}</math> (min) (ADC clock cycles)</b>	<b><math>t_{CONV}</math> (ADC clock cycles) (with min. <math>t_{SMPL}</math>)</b>	<b><math>t_{CONV}</math> (ns) at <math>f_{ADC} = 14</math> MHz</b>
12	12.5	893	1.5	14	1000
10	11.5	821	1.5	13	928
8	9.5	678	1.5	11	785
6	7.5	535	1.5	9	643

### 13.4.3 End of conversion, end of sampling phase (EOC, EOSMP flags)

The ADC indicates each end of conversion (EOC) event.

The ADC sets the EOC flag in the ADC\_ISR register as soon as a new conversion data result is available in the ADC\_DR register. An interrupt can be generated if the EOCIE bit is set in the ADC\_IER register. The EOC flag is cleared by software either by writing 1 to it, or by reading the ADC\_DR register.

The ADC also indicates the end of sampling phase by setting the EOSMP flag in the ADC\_ISR register. The EOSMP flag is cleared by software by writing 1 to it. An interrupt can be generated if the EOSMPIE bit is set in the ADC\_IER register.

The aim of this interrupt is to allow the processing to be synchronized with the conversions. Typically, an analog multiplexer can be accessed in hidden time during the conversion phase, so that the multiplexer is positioned when the next sampling starts.

**Note:** *As there is only a very short time left between the end of the sampling and the end of the conversion, it is recommended to use polling or a WFE instruction rather than an interrupt and a WFI instruction.*

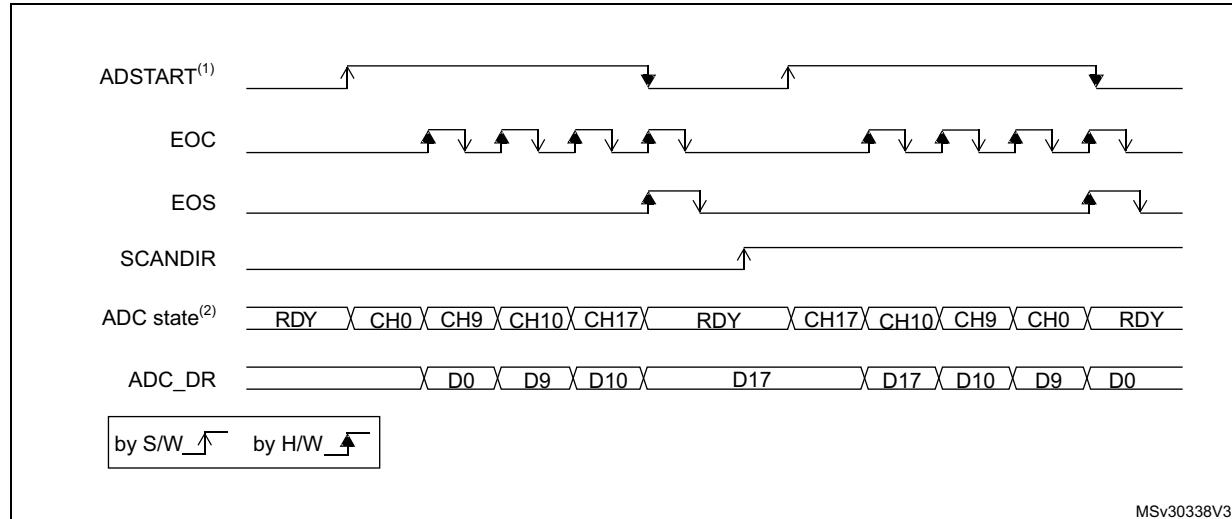
### 13.4.4 End of conversion sequence (EOS flag)

The ADC notifies the application of each end of sequence (EOS) event.

The ADC sets the EOS flag in the ADC\_ISR register as soon as the last data result of a conversion sequence is available in the ADC\_DR register. An interrupt can be generated if the EOSIE bit is set in the ADC\_IER register. The EOS flag is cleared by software by writing 1 to it.

### 13.4.5 Example timing diagrams (single/continuous modes hardware/software triggers)

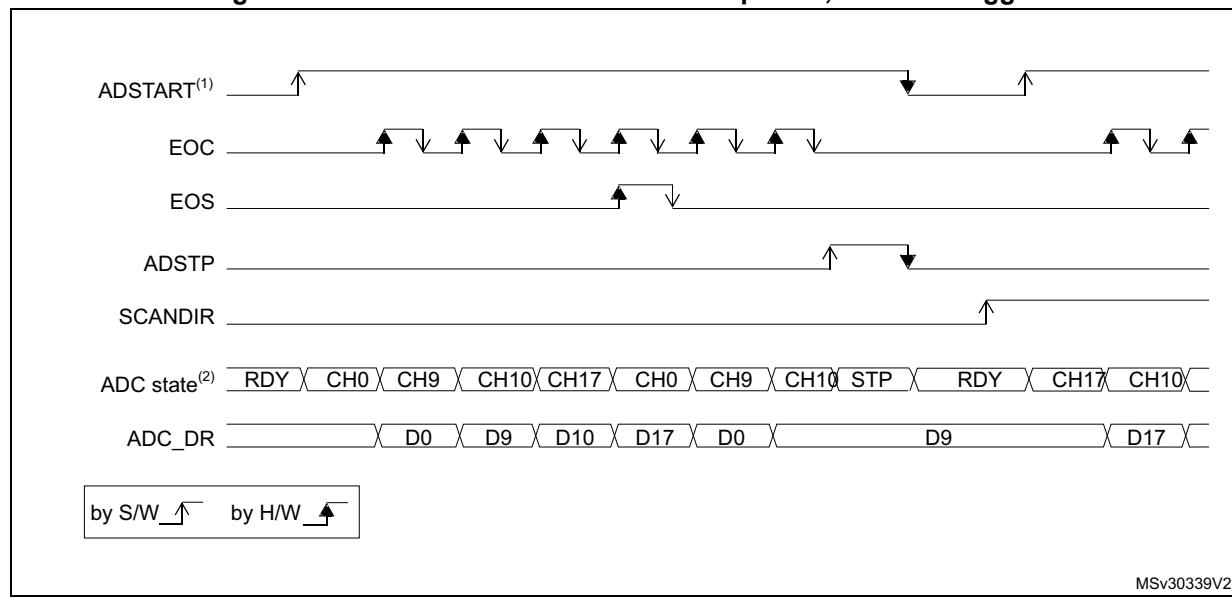
Figure 33. Single conversions of a sequence, software trigger



1. EXTEN = 00, CONT = 0
2. CHSEL = 0x20601, WAIT = 0, AUTOFF = 0

For code example refer to the Appendix section [A.7.5: Single conversion sequence code example - Software trigger](#).

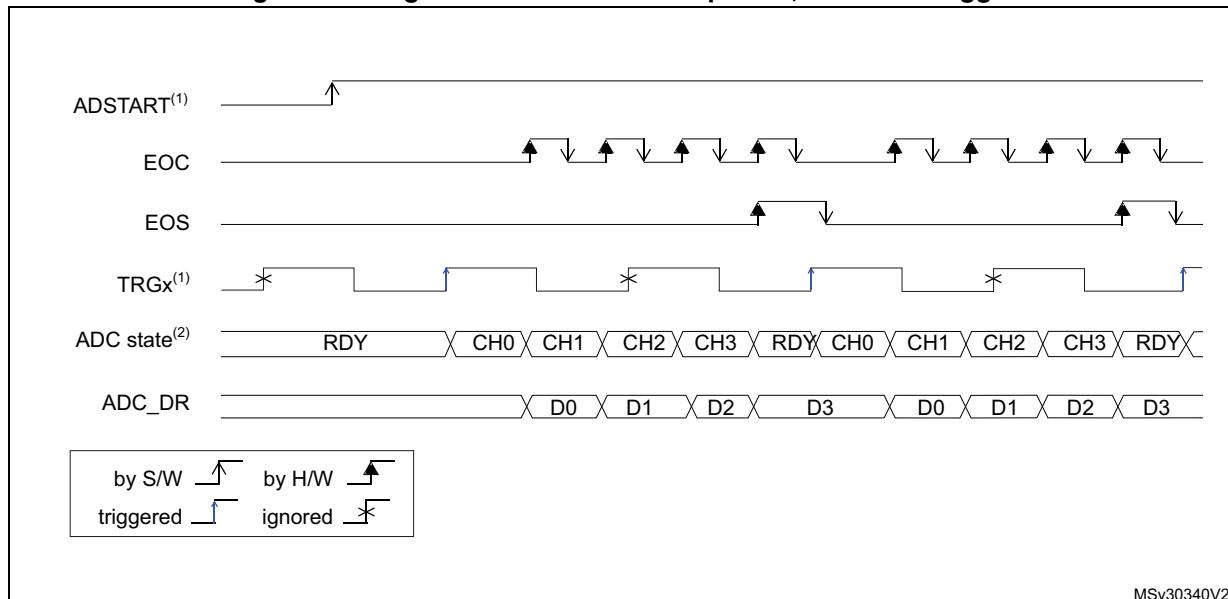
Figure 34. Continuous conversion of a sequence, software trigger



1. EXTEN = 00, CONT = 1,
2. CHSEL = 0x20601, WAIT = 0, AUTOFF = 0

For code example refer to the Appendix section [A.7.6: Continuous conversion sequence code example - Software trigger](#).

Figure 35. Single conversions of a sequence, hardware trigger

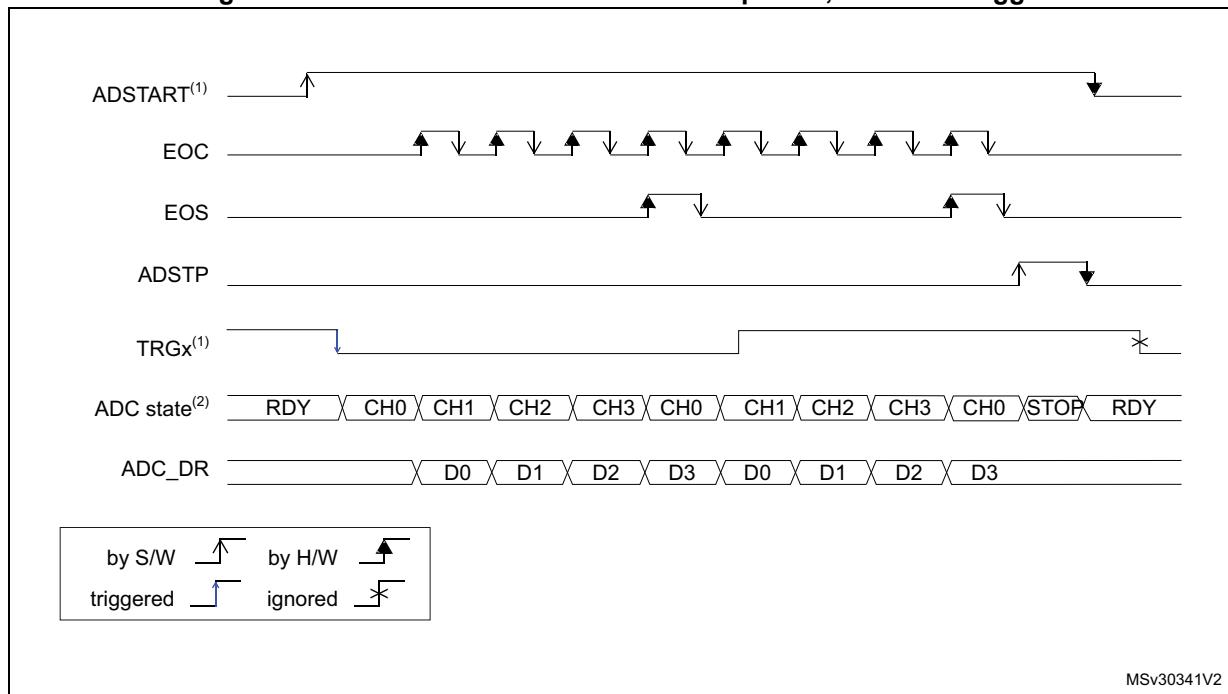


1. EXTSEL = TRGx (over-frequency), EXTEN = 01 (rising edge), CONT = 0

2. CHSEL = 0xF, SCANDIR = 0, WAIT = 0, AUTOFF = 0

For code example refer to the Appendix section [A.7.7: Single conversion sequence code example - Hardware trigger](#).

Figure 36. Continuous conversions of a sequence, hardware trigger



1. EXTSEL = TRGx, EXTEN = 10 (falling edge), CONT = 1

2. CHSEL = 0xF, SCANDIR = 0, WAIT = 0, AUTOFF = 0

For code example refer to the Appendix section [A.7.8: Continuous conversion sequence code example - Hardware trigger](#).

## 13.5 Data management

### 13.5.1 Data register and data alignment (ADC\_DR, ALIGN)

At the end of each conversion (when an EOC event occurs), the result of the converted data is stored in the ADC\_DR data register which is 16-bit wide.

The format of the ADC\_DR depends on the configured data alignment and resolution.

The ALIGN bit in the ADC\_CFGR1 register selects the alignment of the data stored after conversion. Data can be right-aligned (ALIGN = 0) or left-aligned (ALIGN = 1) as shown in *Figure 37*.

**Figure 37. Data alignment and resolution**

ALIGN	RES	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0x0	0x0															DR[11:0]
	0x1		0x00														DR[9:0]
	0x2			0x00													DR[7:0]
	0x3				0x00												DR[5:0]
1	0x0					DR[11:0]											0x0
	0x1					DR[9:0]											0x00
	0x2					DR[7:0]											0x00
	0x3					0x00					DR[5:0]						0x0

MS30342V1

### 13.5.2 ADC overrun (OVR, OVRMOD)

The overrun flag (OVR) indicates a data overrun event, when the converted data was not read in time by the CPU or the DMA, before the data from a new conversion is available.

The OVR flag is set in the ADC\_ISR register if the EOC flag is still at '1' at the time when a new conversion completes. An interrupt can be generated if the OVRIE bit is set in the ADC\_IER register.

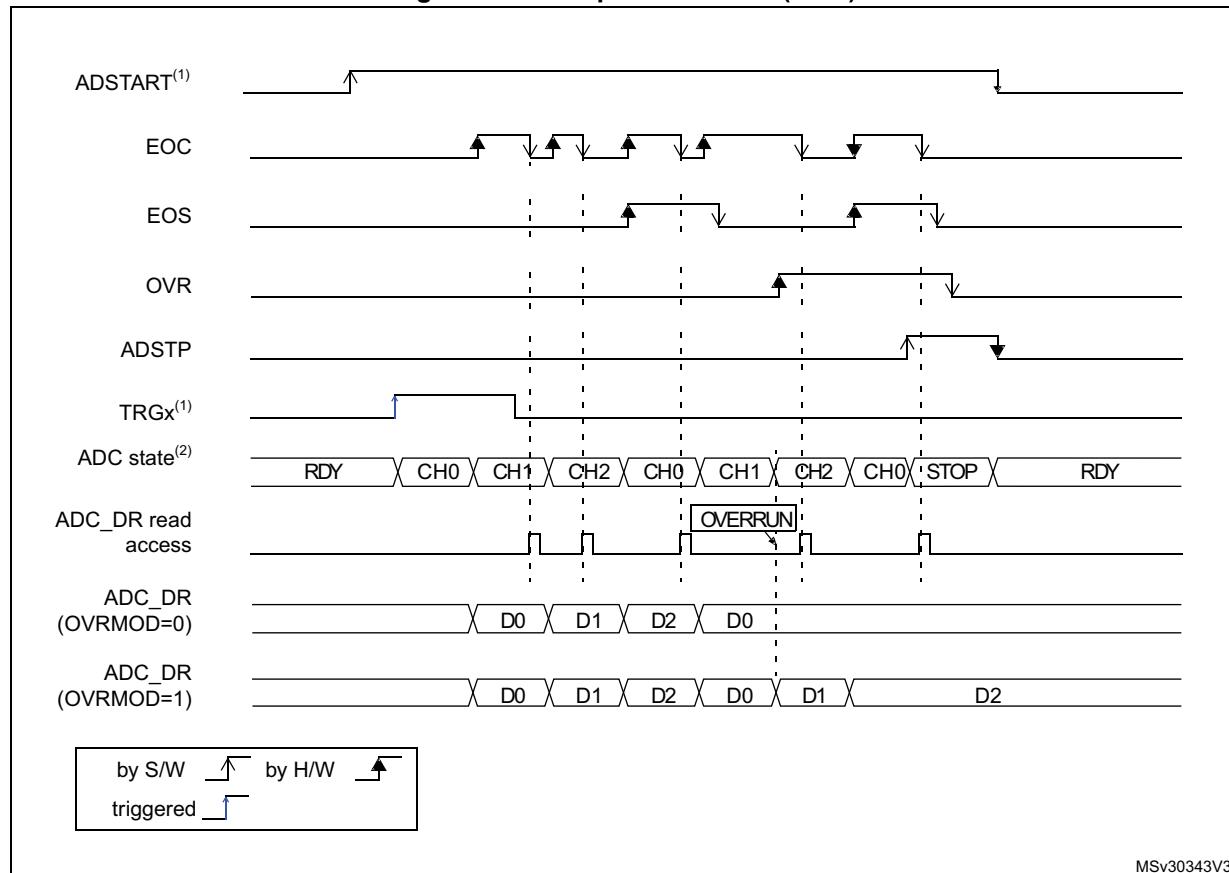
When an overrun condition occurs, the ADC keeps operating and can continue to convert unless the software decides to stop and reset the sequence by setting the ADSTP bit in the ADC\_CR register.

The OVR flag is cleared by software by writing 1 to it.

It is possible to configure if the data is preserved or overwritten when an overrun event occurs by programming the OVRMOD bit in the ADC\_CFGR1 register:

- OVRMOD = 0
  - An overrun event preserves the data register from being overwritten: the old data is maintained and the new conversion is discarded. If OVR remains at 1, further conversions can be performed but the resulting data is discarded.
- OVRMOD = 1
  - The data register is overwritten with the last conversion result and the previous unread data is lost. If OVR remains at 1, further conversions can be performed and the ADC\_DR register always contains the data from the latest conversion.

Figure 38. Example of overrun (OVR)



### 13.5.3 Managing a sequence of data converted without using the DMA

If the conversions are slow enough, the conversion sequence can be handled by software. In this case the software must use the EOC flag and its associated interrupt to handle each data result. Each time a conversion is complete, the EOC bit is set in the ADC\_ISR register and the ADC\_DR register can be read. The OVRMOD bit in the ADC\_CFGR1 register should be configured to 0 to manage overrun events as an error.

### 13.5.4 Managing converted data without using the DMA without overrun

It may be useful to let the ADC convert one or more channels without reading the data after each conversion. In this case, the OVRMOD bit must be configured at 1 and the OVR flag should be ignored by the software. When OVRMOD = 1, an overrun event does not prevent the ADC from continuing to convert and the ADC\_DR register always contains the latest conversion data.

### 13.5.5 Managing converted data using the DMA

Since all converted channel values are stored in a single data register, it is efficient to use DMA when converting more than one channel. This avoids losing the conversion data results stored in the ADC\_DR register.

When DMA mode is enabled (DMAEN bit set in the ADC\_CFGR1 register), a DMA request is generated after the conversion of each channel. This allows the transfer of the converted data from the ADC\_DR register to the destination location selected by the software.

**Note:** *The DMAEN bit in the ADC\_CFGR1 register must be set after the ADC calibration phase.*

Despite this, if an overrun occurs ( $OVR = 1$ ) because the DMA could not serve the DMA transfer request in time, the ADC stops generating DMA requests and the data corresponding to the new conversion is not transferred by the DMA. Which means that all the data transferred to the RAM can be considered as valid.

Depending on the configuration of OVRMOD bit, the data is either preserved or overwritten (refer to [Section 13.5.2: ADC overrun \(OVR, OVRMOD\) on page 249](#)).

The DMA transfer requests are blocked until the software clears the OVR bit.

Two different DMA modes are proposed depending on the application use and are configured with bit DMACFG in the ADC\_CFGR1 register:

- DMA one shot mode (DMACFG = 0).  
This mode should be selected when the DMA is programmed to transfer a fixed number of data words.
- DMA circular mode (DMACFG = 1)  
This mode should be selected when programming the DMA in circular mode or double buffer mode.

### DMA one shot mode (DMACFG = 0)

In this mode, the ADC generates a DMA transfer request each time a new conversion data word is available and stops generating DMA requests once the DMA has reached the last DMA transfer (when a transfer complete interrupt occurs, see [Section 10: Direct memory access controller \(DMA\) on page 188](#)) even if a conversion has been started again.

For code example refer to the Appendix section [A.7.9: DMA one shot mode sequence code example](#).

When the DMA transfer is complete (all the transfers configured in the DMA controller have been done):

- The content of the ADC data register is frozen.
- Any ongoing conversion is aborted and its partial result discarded
- No new DMA request is issued to the DMA controller. This avoids generating an overrun error if there are still conversions which are started.
- The scan sequence is stopped and reset
- The DMA is stopped

### DMA circular mode (DMACFG = 1)

In this mode, the ADC generates a DMA transfer request each time a new conversion data word is available in the data register, even if the DMA has reached the last DMA transfer. This allows the DMA to be configured in circular mode to handle a continuous analog input data stream.

For code example refer to the Appendix section [A.7.10: DMA circular mode sequence code example](#).

## 13.6 Low-power features

### 13.6.1 Wait mode conversion

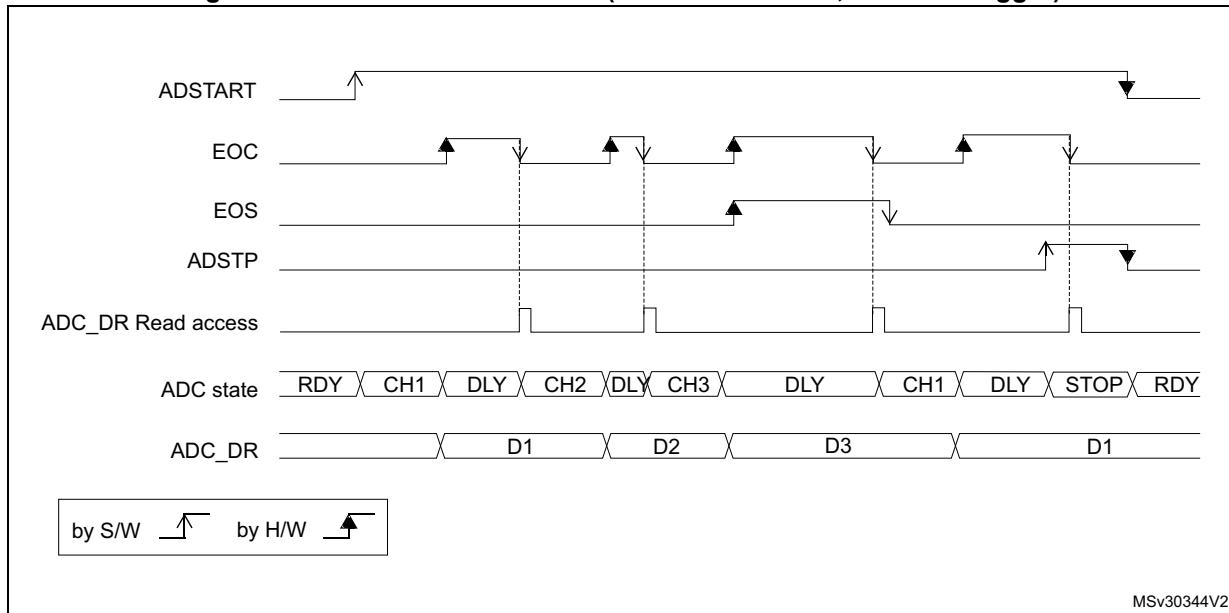
Wait mode conversion can be used to simplify the software as well as optimizing the performance of applications clocked at low frequency where there might be a risk of ADC overrun occurring.

When the WAIT bit is set in the ADC\_CFGR1 register, a new conversion can start only if the previous data has been treated, once the ADC\_DR register has been read or if the EOC bit has been cleared.

This is a way to automatically adapt the speed of the ADC to the speed of the system that reads the data.

**Note:** Any hardware triggers which occur while a conversion is ongoing or during the wait time preceding the read access are ignored.

**Figure 39. Wait mode conversion (continuous mode, software trigger)**



1. EXTEN = 00, CONT = 1
2. CHSEL = 0x3, SCANDIR = 0, WAIT = 1, AUTOFF = 0

For code example refer to the Appendix section [A.7.11: Wait mode sequence code example](#).

### 13.6.2 Auto-off mode (AUTOFF)

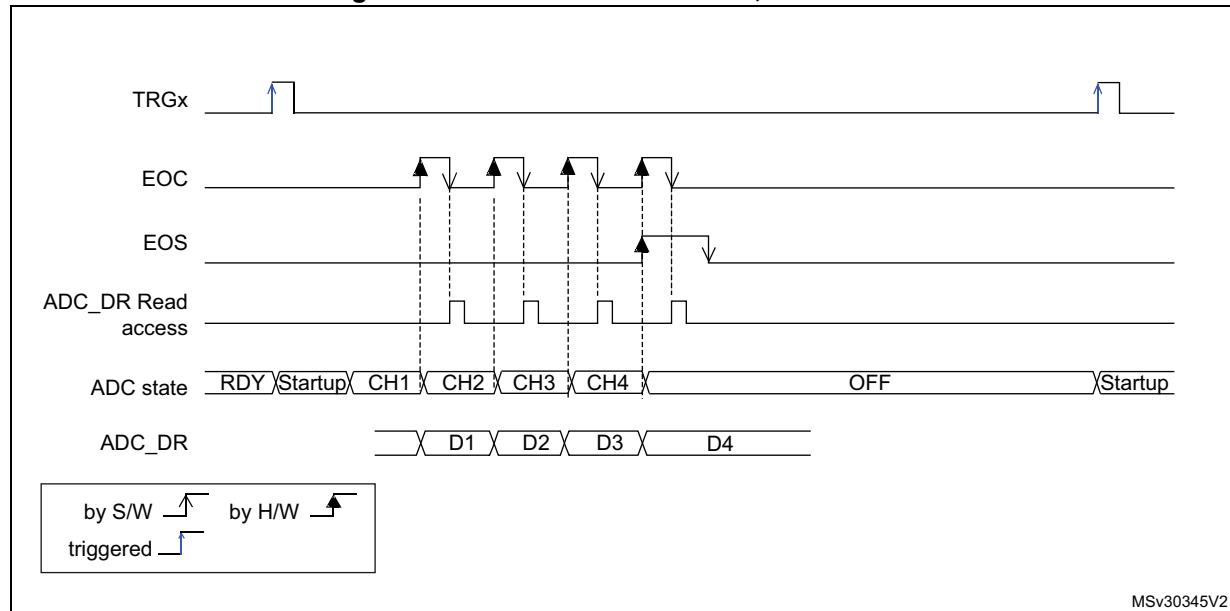
The ADC has an automatic power management feature which is called auto-off mode, and is enabled by setting AUTOFF = 1 in the ADC\_CFGR1 register.

When AUTOFF = 1, the ADC is always powered off when not converting and automatically wakes-up when a conversion is started (by software or hardware trigger). A startup-time is automatically inserted between the trigger event which starts the conversion and the sampling time of the ADC. The ADC is then automatically disabled once the sequence of conversions is complete.

Auto-off mode can cause a dramatic reduction in the power consumption of applications which need relatively few conversions or when conversion requests are timed far enough apart (for example with a low frequency hardware trigger) to justify the extra power and extra time used for switching the ADC on and off.

Auto-off mode can be combined with the wait mode conversion (WAIT = 1) for applications clocked at low frequency. This combination can provide significant power savings if the ADC is automatically powered-off during the wait phase and restarted as soon as the ADC\_DR register is read by the application (see [Figure 41: Behavior with WAIT = 1, AUTOFF = 1](#)).

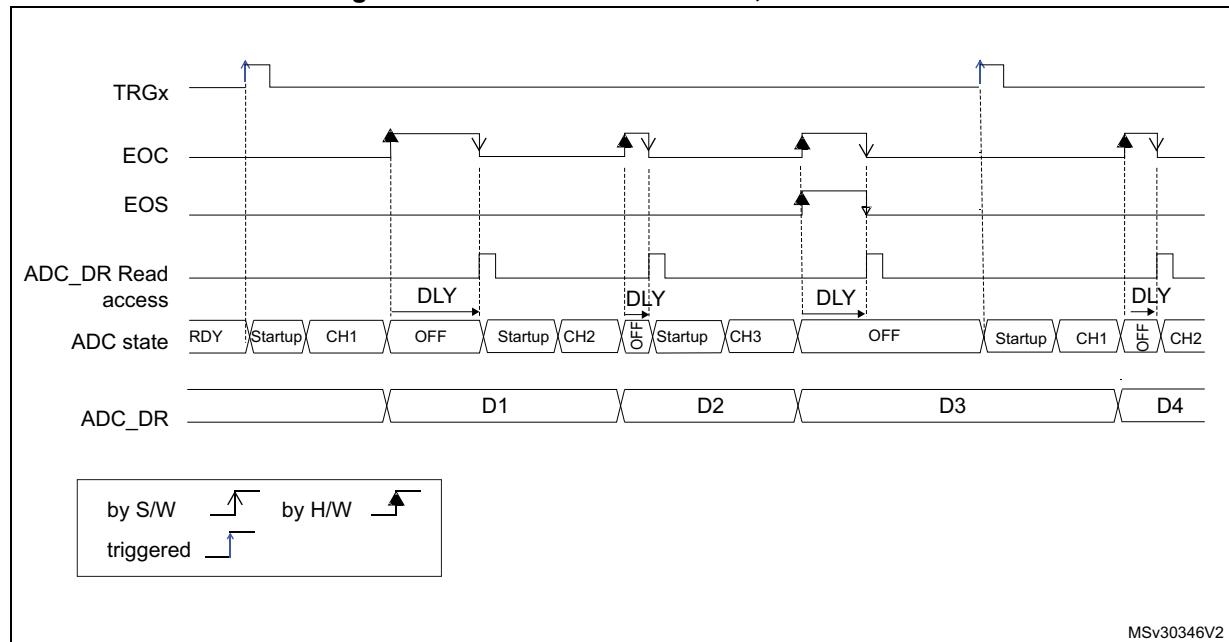
**Figure 40. Behavior with WAIT = 0, AUTOFF = 1**



- EXTSEL = TRGx, EXTEN = 01 (rising edge), CONT = x, ADSTART = 1, CHSEL = 0xF, SCANDIR = 0, WAIT = 1, AUTOFF = 1

For code example refer to the Appendix section [A.7.12: Auto Off and no wait mode sequence code example](#).

Figure 41. Behavior with WAIT = 1, AUTOFF = 1



1. EXTSEL = TRGx, EXTEN = 01 (rising edge), CONT = x, ADSTART = 1, CHSEL = 0xF, SCANDIR = 0, WAIT = 1, AUTOFF = 1

For code example refer to the Appendix section [A.7.13: Auto Off and wait mode sequence code example](#).

## 13.7 Analog window watchdog

### 13.7.1 Description of the analog watchdog

The AWD analog watchdog is enabled by setting the AWDEN bit in the ADC\_CFGR1 register. It is used to monitor that either one selected channel or all enabled channels (see [Table 48: Analog watchdog channel selection](#)) remain within a configured voltage range (window) as shown in [Figure 42](#).

The AWD analog watchdog status bit is set if the analog voltage converted by the ADC is below a lower threshold or above a higher threshold. These thresholds are programmed in HT[11:0] and LT[11:0] bit of ADC\_TR register. An interrupt can be enabled by setting the AWDIE bit in the ADC\_IER register.

The AWD flag is cleared by software by programming it to it.

When converting data with a resolution of less than 12-bit (according to bits RES[1:0]), the LSB of the programmed thresholds must be kept cleared because the internal comparison is always performed on the full 12-bit raw converted data (left aligned).

For code example refer to the Appendix section [A.7.14: Analog watchdog code example](#).

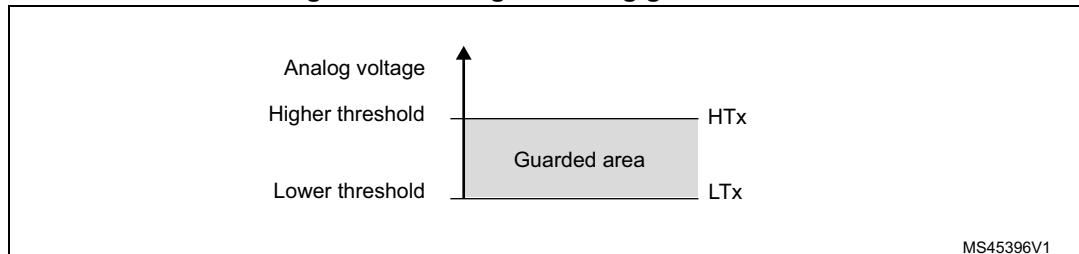
[Table 47](#) describes how the comparison is performed for all the possible resolutions.

**Table 47. Analog watchdog comparison**

Resolution bits RES[1:0]	Analog Watchdog comparison between:		Comments
	Raw converted data, left aligned <sup>(1)</sup>	Thresholds	
00: 12-bit	DATA[11:0]	LT[11:0] and HT[11:0]	-
01: 10-bit	DATA[11:2],00	LT[11:0] and HT[11:0]	The user must configure LT1[1:0] and HT1[1:0] to "00"
10: 8-bit	DATA[11:4],0000	LT[11:0] and HT[11:0]	The user must configure LT1[3:0] and HT1[3:0] to "0000"
11: 6-bit	DATA[11:6],000000	LT[11:0] and HT[11:0]	The user must configure LT1[5:0] and HT1[5:0] to "000000"

1. The watchdog comparison is performed on the raw converted data before any alignment calculation.

**Table 48** shows how to configure the AWDSGL and AWDEN bits in the ADC\_CFGR1 register to enable the analog watchdog on one or more channels.

**Figure 42. Analog watchdog guarded area****Table 48. Analog watchdog channel selection**

Channels guarded by the analog watchdog	AWDSGL bit	AWDEN bit
None	x	0
All channels	0	1
Single <sup>(1)</sup> channel	1	1

1. Selected by the AWDCH[4:0] bits

### 13.7.2 ADC\_AWD1\_OUT output signal generation

The analog watchdog is associated to an internal hardware signal, ADC\_AWD1\_OUT that is directly connected to the ETR input (external trigger) of some on-chip timers (refer to the timers section for details on how to select the ADC\_AWD1\_OUT signal as ETR).

ADC\_AWD1\_OUT is activated when the analog watchdog is enabled:

- ADC\_AWD1\_OUT is set when a guarded conversion is outside the programmed thresholds.
- ADC\_AWD1\_OUT is reset after the end of the next guarded conversion which is inside the programmed thresholds. It remains at 1 if the next guarded conversions are still outside the programmed thresholds.
- ADC\_AWD1\_OUT is also reset when disabling the ADC (when setting ADDIS to 1). Note that stopping conversions (ADSTP set), might clear the ADC\_AWD1\_OUT state.
- ADC\_AWD1\_OUT state does not change when the ADC converts the none-guarded channel (see [Figure 43](#))

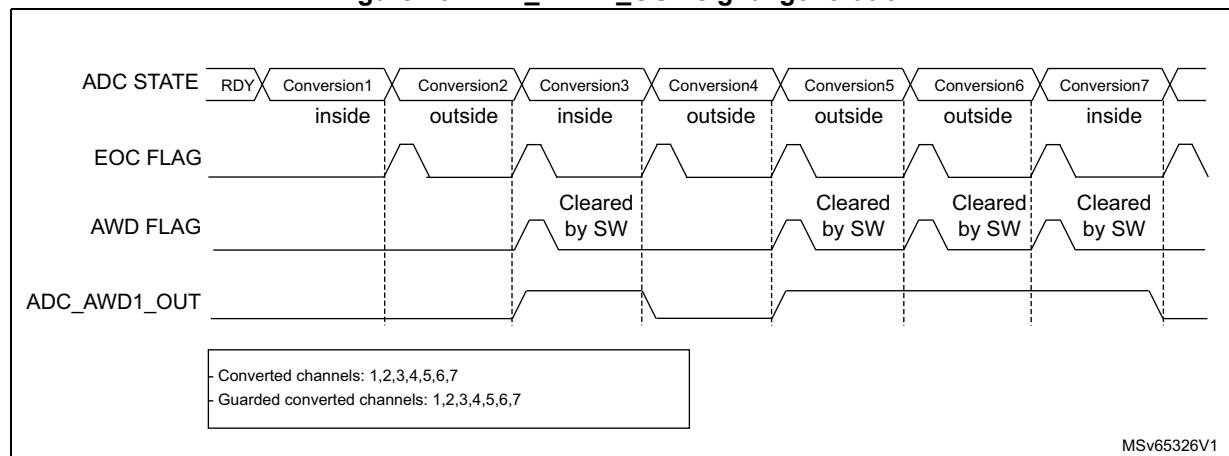
AWD flag is set by hardware and reset by software: AWD flag has no influence on the generation of ADC\_AWD1\_OUT (as an example, ADC\_AWD1\_OUT can toggle while AWD flag remains at 1 if the software has not cleared the flag).

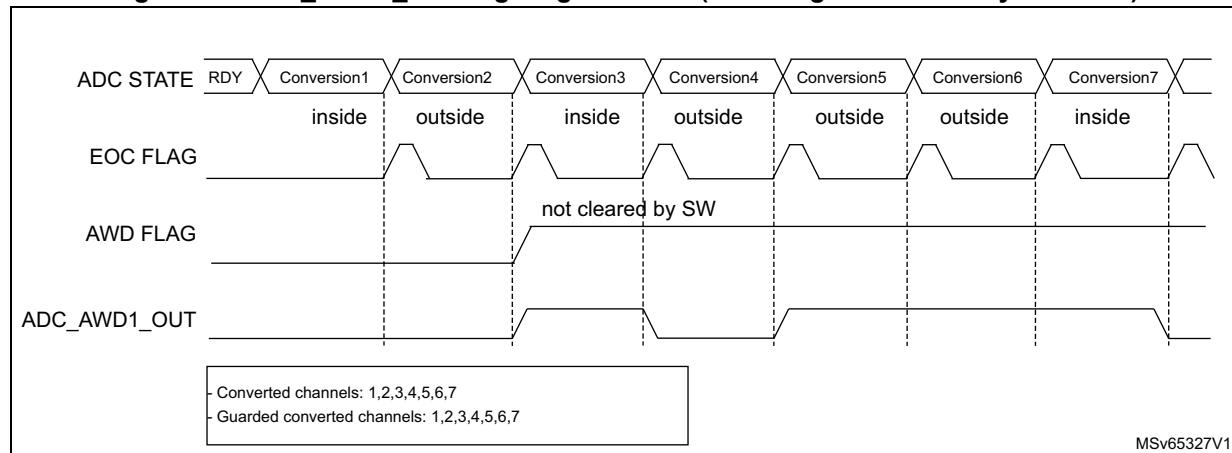
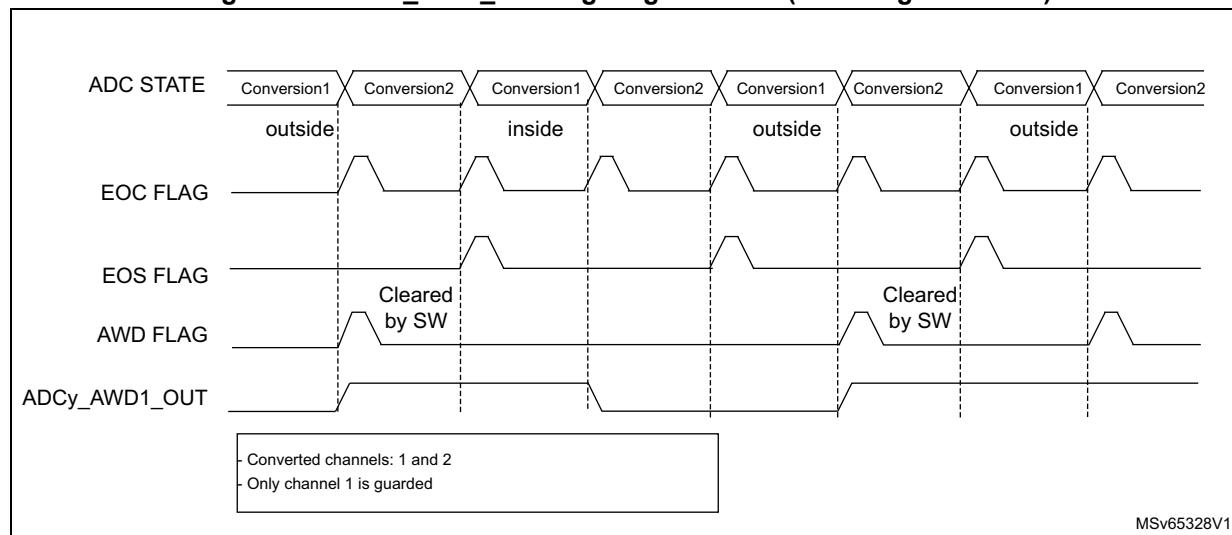
The ADC\_AWD1\_OUT signal is generated by the ADC\_CLK domain. This signal can be generated even the APB clock is stopped.

The AWD comparison is performed at the end of each ADC conversion. The ADC\_AWD1\_OUT rising edge and falling edge occurs two ADC\_CLK clock cycles after the comparison.

As ADC\_AWD1\_OUT is generated by the ADC\_CLK domain and AWD flag is generated by the APB clock domain, the rising edges of these signals are not synchronized.

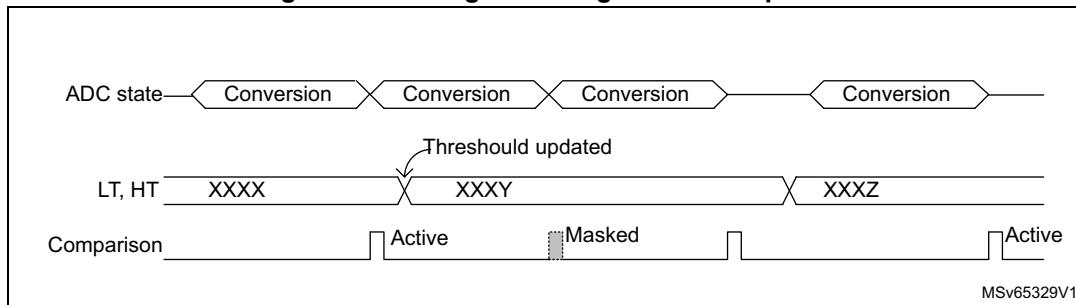
**Figure 43. ADC\_AWD1\_OUT signal generation**



**Figure 44. ADC\_AWD1\_OUT signal generation (AWD flag not cleared by software)****Figure 45. ADC1\_AWD\_OUT signal generation (on a single channel)**

### 13.7.3 Analog watchdog threshold control

LT[11:0] and HT[11:0] can be changed during an analog-to-digital conversion (that is between the start of the conversion and the end of conversion of the ADC internal state). If LT and HT bits are programmed during the ADC guarded channel conversion, the watchdog function is masked for this conversion. This mask is cleared when starting a new conversion, and the resulting new AWD threshold is applied starting the next ADC conversion result. AWD comparison is performed at each end of conversion. If the current ADC data are out of the new threshold interval, this does not generate any interrupt or an ADC\_AWD1\_OUT signal. The Interrupt and the ADC\_AWD1\_OUT generation only occurs at the end of the ADC conversion that started after the threshold update. If ADC\_AWD1\_OUT is already asserted, programming the new threshold does not deassert the ADC\_AWD1\_OUT signal.

**Figure 46. Analog watchdog threshold update**

## 13.8 Temperature sensor and internal reference voltage

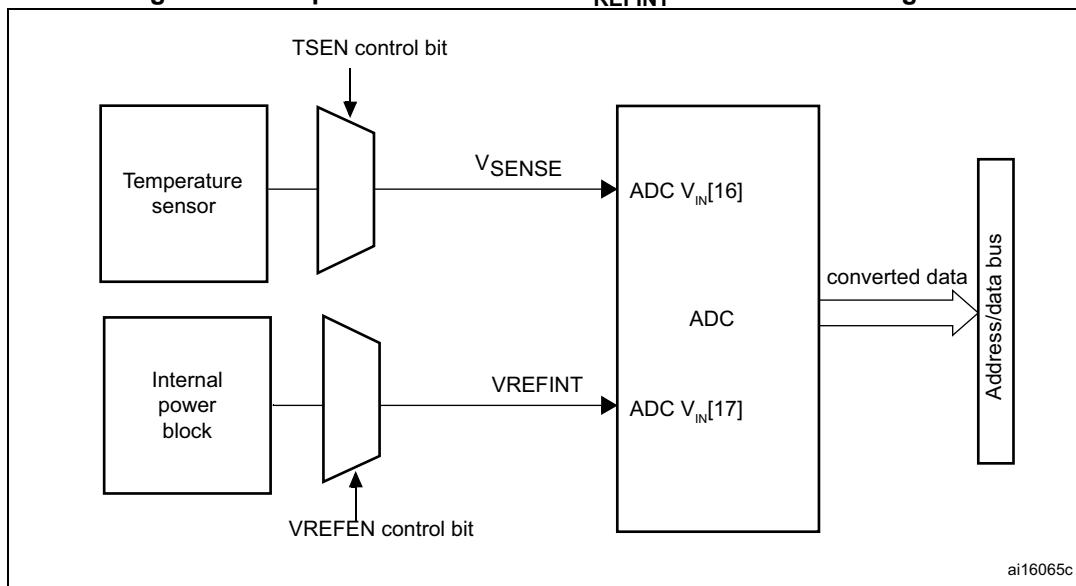
The temperature sensor can be used to measure the junction temperature ( $T_J$ ) of the device. The temperature sensor is internally connected to the ADC  $V_{IN}[16]$  input channel which is used to convert the sensor's output voltage to a digital value. The sampling time for the temperature sensor analog pin must be greater than the minimum  $T_{S\_temp}$  value specified in the datasheet. When not in use, the sensor can be put in power down mode.

The temperature sensor output voltage changes linearly with temperature, however its characteristics may vary significantly from chip to chip due to the process variations. To improve the accuracy of the temperature sensor (especially for absolute temperature measurement), calibration values are individually measured for each part by ST during production test and stored in the system memory area. Refer to the specific device datasheet for additional information.

The internal voltage reference (VREFINT) provides a stable (bandgap) voltage output for the ADC and comparators. VREFINT is internally connected to the ADC  $V_{IN}[17]$  input channel. The precise voltage of VREFINT is individually measured for each part by ST during production test and stored in the system memory area.

*Figure 47* shows the block diagram of connections between the temperature sensor, the internal voltage reference and the ADC.

The TSEN bit must be set to enable the conversion of ADC  $V_{IN}[16]$  (temperature sensor) and the VREFEN bit must be set to enable the conversion of ADC  $V_{IN}[17]$  (VREFINT).

**Figure 47. Temperature sensor and V<sub>REFINT</sub> channel block diagram**

ai16065c

### Reading the temperature

1. Select the ADC V<sub>IN[16]</sub> input channel.
2. Select an appropriate sampling time specified in the device datasheet (T<sub>S\_temp</sub>).
3. Set the TSEN bit in the ADC\_CCR register to wake up the temperature sensor from power down mode and wait for its stabilization time (t<sub>START</sub>).  
For code example refer to the Appendix section [A.7.15: Temperature configuration code example](#).
4. Start the ADC conversion by setting the ADSTART bit in the ADC\_CR register (or by external trigger).
5. Read the resulting V<sub>SENSE</sub> data in the ADC\_DR register.
6. Calculate the temperature using the following formula

$$\text{Temperature (in } ^\circ\text{C)} = \frac{\text{TS\_CAL2\_TEMP} - \text{TS\_CAL1\_TEMP}}{\text{TS\_CAL2} - \text{TS\_CAL1}} \times (\text{TS\_DATA} - \text{TS\_CAL1}) + \text{TS\_CAL1\_TEMP}$$

Where:

- TS\_CAL2 is the temperature sensor calibration value acquired at TS\_CAL2\_TEMP (refer to the datasheet for TS\_CAL2 value)
- TS\_CAL1 is the temperature sensor calibration value acquired at TS\_CAL1\_TEMP (refer to the datasheet for TS\_CAL1 value)
- TS\_DATA is the actual temperature sensor output value converted by ADC  
Refer to the specific device datasheet for more information about TS\_CAL1 and TS\_CAL2 calibration points.

For code example refer to the [A.7.16: Temperature computation code example](#).

**Note:**

The sensor has a startup time after waking from power down mode before it can output V<sub>SENSE</sub> at the correct level. The ADC also has a startup time after power-on, so to minimize the delay, the ADEN and TSEN bits should be set at the same time.

### Calculating the actual V<sub>DDA</sub> voltage using the internal reference voltage

The V<sub>DDA</sub> power supply voltage applied to the device may be subject to variation or not precisely known. The embedded internal voltage reference (V<sub>REFINT</sub>) and its calibration data, acquired by the ADC during the manufacturing process at V<sub>DDA\_Charac</sub>, can be used to evaluate the actual V<sub>DDA</sub> voltage level.

The following formula gives the actual V<sub>DDA</sub> voltage supplying the device:

$$V_{DDA} = V_{DDA\_Charac} \times VREFINT\_CAL / VREFINT\_DATA$$

Where:

- V<sub>DDA\_Charac</sub> is the value of V<sub>DDA</sub> voltage characterized at V<sub>REFINT</sub> during the manufacturing process. It is specified in the device datasheet.
- VREFINT\_CAL is the VREFINT calibration value
- VREFINT\_DATA is the actual VREFINT output value converted by ADC

### Converting a supply-relative ADC measurement to an absolute voltage value

The ADC is designed to deliver a digital value corresponding to the ratio between the analog power supply and the voltage applied on the converted channel. For most application use cases, it is necessary to convert this ratio into a voltage independent of V<sub>DDA</sub>. For applications where V<sub>DDA</sub> is known and ADC converted values are right-aligned you can use the following formula to get this absolute value:

$$V_{CHANNELx} = \frac{V_{DDA}}{FULL\_SCALE} \times ADC\_DATA_x$$

For applications where V<sub>DDA</sub> value is not known, you must use the internal voltage reference and V<sub>DDA</sub> can be replaced by the expression provided in *Section : Calculating the actual V<sub>DDA</sub> voltage using the internal reference voltage*, resulting in the following formula:

$$V_{CHANNELx} = \frac{V_{DDA\_Charac} \times VREFINT\_CAL \times ADC\_DATA_x}{VREFINT\_DATA \times FULL\_SCALE}$$

Where:

- V<sub>DDA\_Charac</sub> is the value of V<sub>DDA</sub> voltage characterized at V<sub>REFINT</sub> during the manufacturing process. It is specified in the device datasheet.
- VREFINT\_CAL is the VREFINT calibration value
- ADC\_DATA<sub>x</sub> is the value measured by the ADC on channelx (right-aligned)
- VREFINT\_DATA is the actual VREFINT output value converted by the ADC
- full\_SCALE is the maximum digital value of the ADC output. For example with 12-bit resolution, it is 2<sup>12</sup> - 1 = 4095 or with 8-bit resolution, 2<sup>8</sup> - 1 = 255.

**Note:** If ADC measurements are done using an output format other than 12 bit right-aligned, all the parameters must first be converted to a compatible format before the calculation is done.

## 13.9 Battery voltage monitoring

The VBATEN bit in the ADC\_CCR register allows the application to measure the backup battery voltage on the VBAT pin. As the V<sub>BAT</sub> voltage could be higher than V<sub>DDA</sub>, to ensure the correct operation of the ADC, the VBAT pin is internally connected to a bridge divider. This bridge is automatically enabled when VBATEN is set, to connect V<sub>BAT</sub> to the ADC V<sub>IN</sub>[18] input channel. As a consequence, the converted digital value is V<sub>BAT</sub>/2. To prevent

any unwanted consumption on the battery, it is recommended to enable the bridge divider only when needed for ADC conversion.

## 13.10 ADC interrupts

An interrupt can be generated by any of the following events:

- ADC power-up, when the ADC is ready (ADRDY flag)
- End of any conversion (EOC flag)
- End of a sequence of conversions (EOS flag)
- When an analog watchdog detection occurs (AWD flag)
- When the end of sampling phase occurs (EOSMP flag)
- when a data overrun occurs (OVR flag)

Separate interrupt enable bits are available for flexibility.

**Table 49. ADC interrupts**

Interrupt event	Event flag	Enable control bit
ADC ready	ADRDY	ADRDYIE
End of conversion	EOC	EOCIE
End of sequence of conversions	EOS	EOSIE
Analog watchdog status bit is set	AWD	AWDIE
End of sampling phase	EOSMP	EOSMPIE
Overrun	OVR	OVRIE

## 13.11 ADC registers

Refer to [Section 1.2](#) for a list of abbreviations used in register descriptions.

### 13.11.1 ADC interrupt and status register (ADC\_ISR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	AWD	Res.	Res.	OVR	EOS	EOC	EOSMP	ADRDY							
								rc_w1			rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:13 Reserved, must be kept at reset value.

Bits 12:10 Reserved, must be kept at reset value.

Bits 9:8 Reserved, must be kept at reset value.

#### Bit 7 AWD: Analog watchdog flag

This bit is set by hardware when the converted voltage crosses the values programmed in ADC\_TR register. It is cleared by software by programming it to 1.

0: No analog watchdog event occurred (or the flag event was already acknowledged and cleared by software)  
1: Analog watchdog event occurred

Bits 6:5 Reserved, must be kept at reset value.

#### Bit 4 OVR: ADC overrun

This bit is set by hardware when an overrun occurs, meaning that a new conversion has complete while the EOC flag was already set. It is cleared by software writing 1 to it.

0: No overrun occurred (or the flag event was already acknowledged and cleared by software)  
1: Overrun has occurred

#### Bit 3 EOS: End of sequence flag

This bit is set by hardware at the end of the conversion of a sequence of channels selected by the CHSEL bits. It is cleared by software writing 1 to it.

0: Conversion sequence not complete (or the flag event was already acknowledged and cleared by software)  
1: Conversion sequence complete

**Bit 2 EOC:** End of conversion flag

This bit is set by hardware at the end of each conversion of a channel when a new data result is available in the ADC\_DR register. It is cleared by software writing 1 to it or by reading the ADC\_DR register.

0: Channel conversion not complete (or the flag event was already acknowledged and cleared by software)  
1: Channel conversion complete

**Bit 1 EOSMP:** End of sampling flag

This bit is set by hardware during the conversion, at the end of the sampling phase. It is cleared by software by programming it to '1'.

0: Not at the end of the sampling phase (or the flag event was already acknowledged and cleared by software)  
1: End of sampling phase reached

**Bit 0 ADRDY:** ADC ready

This bit is set by hardware after the ADC has been enabled (ADEN = 1) and when the ADC reaches a state where it is ready to accept conversion requests.

It is cleared by software writing 1 to it.

0: ADC not yet ready to start conversion (or the flag event was already acknowledged and cleared by software)  
1: ADC is ready to start conversion

**Note:** In auto-off mode (AUTOFF = 1) the power-on/off phases are performed automatically, by hardware and the ADRDY flag is not set.

### 13.11.2 ADC interrupt enable register (ADC\_IER)

Address offset: 0x04

Reset value: 0x0000 0000

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

Bits 31:13 Reserved, must be kept at reset value.

Bits 12:10 Reserved, must be kept at reset value.

Bits 9:8 Reserved, must be kept at reset value.

**Bit 7 AWDIE:** Analog watchdog interrupt enable

This bit is set and cleared by software to enable/disable the analog watchdog interrupt.

0: Analog watchdog interrupt disabled

1: Analog watchdog interrupt enabled

**Note:** The Software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).

Bits 6:5 Reserved, must be kept at reset value.

**Bit 4 OVRIE:** Overrun interrupt enable

This bit is set and cleared by software to enable/disable the overrun interrupt.

0: Overrun interrupt disabled

1: Overrun interrupt enabled. An interrupt is generated when the OVR bit is set.

*Note: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).*

**Bit 3 EOSIE:** End of conversion sequence interrupt enable

This bit is set and cleared by software to enable/disable the end of sequence of conversions interrupt.

0: EOS interrupt disabled

1: EOS interrupt enabled. An interrupt is generated when the EOS bit is set.

*Note: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).*

**Bit 2 EOCIE:** End of conversion interrupt enable

This bit is set and cleared by software to enable/disable the end of conversion interrupt.

0: EOC interrupt disabled

1: EOC interrupt enabled. An interrupt is generated when the EOC bit is set.

*Note: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).*

**Bit 1 EOSMPIE:** End of sampling flag interrupt enable

This bit is set and cleared by software to enable/disable the end of the sampling phase interrupt.

0: EOSMP interrupt disabled.

1: EOSMP interrupt enabled. An interrupt is generated when the EOSMP bit is set.

*Note: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).*

**Bit 0 ADRDYIE:** ADC ready interrupt enable

This bit is set and cleared by software to enable/disable the ADC Ready interrupt.

0: ADRDY interrupt disabled.

1: ADRDY interrupt enabled. An interrupt is generated when the ADRDY bit is set.

*Note: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).*

### 13.11.3 ADC control register (ADC\_CR)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADCAL	Res.	Res.	Res.	Res.	Res.										
rs															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ADSTP	Res.	ADSTA RT	ADDIS	ADEN
											rs		rs	rs	rs

Bit 31 **ADCAL**: ADC calibration

This bit is set by software to start the calibration of the ADC.

It is cleared by hardware after calibration is complete.

0: Calibration complete

1: Write 1 to calibrate the ADC. Read at 1 means that a calibration is in progress.

*Note: The software is allowed to set ADCAL only when the ADC is disabled (ADCAL = 0, ADSTART = 0, ADSTP = 0, ADDIS = 0, AUTOFF = 0, and ADEN = 0).*

*is allowed to*

Bits 30:28 Reserved, must be kept at reset value.

Bits 27:5 Reserved, must be kept at reset value.

Bit 4 **ADSTP**: ADC stop conversion command

This bit is set by software to stop and discard an ongoing conversion (ADSTP Command).

It is cleared by hardware when the conversion is effectively discarded and the ADC is ready to accept a new start conversion command.

0: No ADC stop conversion command ongoing

1: Write 1 to stop the ADC. Read 1 means that an ADSTP command is in progress.

*Note: Setting ADSTP to '1' is only effective when ADSTART = 1 and ADDIS = 0 (ADC is enabled and may be converting and there is no pending request to disable the ADC)*

Bit 3 Reserved, must be kept at reset value.

**Bit 2 ADSTART:** ADC start conversion command

This bit is set by software to start ADC conversion. Depending on the EXLEN [1:0] configuration bits, a conversion either starts immediately (software trigger configuration) or once a hardware trigger event occurs (hardware trigger configuration).

It is cleared by hardware:

- In single conversion mode (CONT = 0, DISCEN = 0), when software trigger is selected (EXLEN = 00): at the assertion of the end of Conversion Sequence (EOS) flag.
- In discontinuous conversion mode (CONT = 0, DISCEN = 1), when the software trigger is selected (EXLEN = 00): at the assertion of the end of Conversion (EOC) flag.
- In all other cases: after the execution of the ADSTP command, at the same time as the ADSTP bit is cleared by hardware.

0: No ADC conversion is ongoing.

1: Write 1 to start the ADC. Read 1 means that the ADC is operating and may be converting.

*Note: The software is allowed to set ADSTART only when ADEN = 1 and ADDIS = 0 (ADC is enabled and there is no pending request to disable the ADC).*

**Bit 1 ADDIS:** ADC disable command

This bit is set by software to disable the ADC (ADDIS command) and put it into power-down state (OFF state).

It is cleared by hardware once the ADC is effectively disabled (ADEN is also cleared by hardware at this time).

0: No ADDIS command ongoing

1: Write 1 to disable the ADC. Read 1 means that an ADDIS command is in progress.

*Note: Setting ADDIS to ‘1’ is only effective when ADEN = 1 and ADSTART = 0 (which ensures that no conversion is ongoing)*

**Bit 0 ADEN:** ADC enable command

This bit is set by software to enable the ADC. The ADC is effectively ready to operate once the ADRDY flag has been set.

It is cleared by hardware when the ADC is disabled, after the execution of the ADDIS command.

0: ADC is disabled (OFF state)

1: Write 1 to enable the ADC.

*Note: The software is allowed to set ADEN only when all bits of ADC\_CR registers are 0 (ADCAL = 0, ADSTP = 0, ADSTART = 0, ADDIS = 0 and ADEN = 0)*

### 13.11.4 ADC configuration register 1 (ADC\_CFGR1)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	AWDCH[4:0]					Res.	Res.	AWDEN	AWDSGL	Res.	Res.	Res.	Res.	Res.	DISCEN
	rw	rw	rw	rw	rw			rw	rw						rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AUTOFF	WAIT	CONT	OVRMOD	EXTEN[1:0]		Res.	EXTSEL[2:0]			ALIGN	RES[1:0]		SCANDIR	DMACFG	DMAEN
rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 Reserved, must be kept at reset value.

Bits 30:26 **AWDCH[4:0]**: Analog watchdog channel selection

These bits are set and cleared by software. They select the input channel to be guarded by the analog watchdog.

00000: ADC analog input Channel 0 monitored by AWD

00001: ADC analog input Channel 1 monitored by AWD

.....

10001: ADC analog input Channel 17 monitored by AWD

10010: ADC analog input Channel 18 monitored by AWD

Others: Reserved

*Note: The channel selected by the AWDCH[4:0] bits must be also set into the CHSEL register.*

*The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).*

Bits 25:24 Reserved, must be kept at reset value.

Bit 23 **AWDEN**: Analog watchdog enable

This bit is set and cleared by software.

0: Analog watchdog disabled

1: Analog watchdog enabled

*Note: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).*

Bit 22 **AWDSGL**: Enable the watchdog on a single channel or on all channels

This bit is set and cleared by software to enable the analog watchdog on the channel identified by the AWDCH[4:0] bits or on all the channels

0: Analog watchdog enabled on all channels

1: Analog watchdog enabled on a single channel

*Note: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).*

Bits 21:17 Reserved, must be kept at reset value.

Bit 16 **DISCEN**: Discontinuous mode

This bit is set and cleared by software to enable/disable discontinuous mode.

0: Discontinuous mode disabled

1: Discontinuous mode enabled

*Note:* It is not possible to have both discontinuous mode and continuous mode enabled: it is forbidden to set both bits DISCEN = 1 and CONT = 1.

The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).

Bit 15 **AUTOFF**: Auto-off mode

This bit is set and cleared by software to enable/disable auto-off mode.

0: Auto-off mode disabled

1: Auto-off mode enabled

*Note:* The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).

Bit 14 **WAIT**: Wait conversion mode

This bit is set and cleared by software to enable/disable wait conversion mode.

0: Wait conversion mode off

1: Wait conversion mode on

*Note:* The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).

Bit 13 **CONT**: Single / continuous conversion mode

This bit is set and cleared by software. If it is set, conversion takes place continuously until it is cleared.

0: Single conversion mode

1: Continuous conversion mode

*Note:* It is not possible to have both discontinuous mode and continuous mode enabled: it is forbidden to set both bits DISCEN = 1 and CONT = 1.

The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).

Bit 12 **OVRMOD**: Overrun management mode

This bit is set and cleared by software and configures the way data overruns are managed.

0: ADC\_DR register is preserved with the old data when an overrun is detected.

1: ADC\_DR register is overwritten with the last conversion result when an overrun is detected.

*Note:* The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).

Bits 11:10 **EXTEN[1:0]**: External trigger enable and polarity selection

These bits are set and cleared by software to select the external trigger polarity and enable the trigger.

00: Hardware trigger detection disabled (conversions can be started by software)

01: Hardware trigger detection on the rising edge

10: Hardware trigger detection on the falling edge

11: Hardware trigger detection on both the rising and falling edges

*Note:* The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).

Bit 9 Reserved, must be kept at reset value.

**Bits 8:6 EXTSEL[2:0]: External trigger selection**

These bits select the external event used to trigger the start of conversion (refer to [Table 43: External triggers](#) for details):

000: TRG0

001: TRG1

010: TRG2

011: TRG3

100: TRG4

101: TRG5

110: TRG6

111: TRG7

*Note:* The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).

**Bit 5 ALIGN: Data alignment**

This bit is set and cleared by software to select right or left alignment. Refer to [Figure 37: Data alignment and resolution on page 249](#)

0: Right alignment

1: Left alignment

*Note:* The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).

**Bits 4:3 RES[1:0]: Data resolution**

These bits are written by software to select the resolution of the conversion.

00: 12 bits

01: 10 bits

10: 8 bits

11: 6 bits

*Note:* The software is allowed to write these bits only when ADEN is cleared.

**Bit 2 SCANDIR:** Scan sequence direction

This bit is set and cleared by software to select the direction in which the channels is scanned in the sequence.

- 0: Upward scan (from CHSEL0 to CHSEL18)
- 1: Backward scan (from CHSEL18 to CHSEL0)

*Note: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).*

**Bit 1 DMACFG:** Direct memory access configuration

This bit is set and cleared by software to select between two DMA modes of operation and is effective only when DMAEN = 1.

- 0: DMA one shot mode selected
- 1: DMA circular mode selected

For more details, refer to [Section 13.5.5: Managing converted data using the DMA on page 250](#)

*Note: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).*

**Bit 0 DMAEN:** Direct memory access enable

This bit is set and cleared by software to enable the generation of DMA requests. This allows the DMA controller to be used to manage automatically the converted data. For more details, refer to [Section 13.5.5: Managing converted data using the DMA on page 250](#).

- 0: DMA disabled
- 1: DMA enabled

*Note: The software is allowed to write this bit only when ADSTART bit is cleared (this ensures that no conversion is ongoing).*

### 13.11.5 ADC configuration register 2 (ADC\_CFGR2)

Address offset: 0x10

Reset value: 0x0000 0000

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

Bits 31:30 CKMODE[1:0]: ADC clock mode

These bits are set and cleared by software to define how the analog ADC is clocked:

00: ADCCLK (Asynchronous clock mode), generated at product level (refer to RCC section)

01: PCLK/2 (Synchronous clock mode)

10: PCLK/4 (Synchronous clock mode)

11: Reserved

In all synchronous clock modes, there is no jitter in the delay from a timer trigger to the start of a conversion.

*Note: The software is allowed to write these bits only when the ADC is disabled (ADCAL = 0, ADSTART = 0, ADSTP = 0, ADDIS = 0 and ADEN = 0).*

Bits 29:10 Reserved, must be kept at reset value.

Bits 9:0 Reserved, must be kept at reset value.

### 13.11.6 ADC sampling time register (ADC\_SMPR)

Address offset: 0x14

Reset value: 0x0000 0000

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

Bits 31:3 Reserved, must be kept at reset value.

#### Bits 2:0 **SMP[2:0]: Sampling time selection**

These bits are written by software to select the sampling time that applies to all channels.

000: 1.5 ADC clock cycles

001: 7.5 ADC clock cycles

010: 13.5 ADC clock cycles

011: 28.5 ADC clock cycles

100: 41.5 ADC clock cycles

101: 55.5 ADC clock cycles

110: 71.5 ADC clock cycles

111: 239.5 ADC clock cycles

*Note: The software is allowed to write this bit only when ADSTART = 0 (which ensures that no conversion is ongoing).*

### 13.11.7 ADC watchdog threshold register (ADC\_TR)

Address offset: 0x20

Reset value: 0xFFFF 0000

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

Bits 31:28 Reserved, must be kept at reset value.

#### Bits 27:16 **HT[11:0]: Analog watchdog higher threshold**

These bits are written by software to define the higher threshold for the analog watchdog. Refer to [Section 13.7: Analog window watchdog on page 254](#)

*Note: The software is allowed to write this bit only when ADSTART = 0 (which ensures that no conversion is ongoing).*

Bits 15:12 Reserved, must be kept at reset value.

#### Bits 11:0 **LT[11:0]: Analog watchdog lower threshold**

These bits are written by software to define the lower threshold for the analog watchdog.

Refer to [Section 13.7: Analog window watchdog on page 254](#).

*Note: The software is allowed to write this bit only when ADSTART = 0 (which ensures that no conversion is ongoing).*

### 13.11.8 ADC channel selection register (ADC\_CHSEL\_R)

Address offset: 0x28

Reset value: 0x0000 0000

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

Bits 31:19 Reserved, must be kept at reset value.

#### Bits 18:0 **CHSELx**: Channel-x selection

These bits are written by software and define which channels are part of the sequence of channels to be converted.

0: Input Channel-x is not selected for conversion

1: Input Channel-x is selected for conversion

*Note: The software is allowed to write this bit only when ADSTART = 0 (which ensures that no conversion is ongoing).*

### 13.11.9 ADC data register (ADC\_DR)

Address offset: 0x40

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

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

#### Bits 15:0 **DATA[15:0]**: Converted data

These bits are read-only. They contain the conversion result from the last converted channel. The data are left- or right-aligned as shown in [Figure 37: Data alignment and resolution on page 249](#).

Just after a calibration is complete, DATA[6:0] contains the calibration factor.

### 13.11.10 ADC common configuration register (ADC\_CCR)

Address offset: 0x308

Reset value: 0x0000 0000

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

Bits 31:25 Reserved, must be kept at reset value.

#### Bit 24 VBATEN: V<sub>BAT</sub> enable

This bit is set and cleared by software to enable/disable the V<sub>BAT</sub> channel.

0: V<sub>BAT</sub> channel disabled

1: V<sub>BAT</sub> channel enabled

*Note: The software is allowed to write this bit only when ADSTART = 0 (which ensures that no conversion is ongoing)*

#### Bit 23 TSEN: Temperature sensor enable

This bit is set and cleared by software to enable/disable the temperature sensor.

0: Temperature sensor disabled

1: Temperature sensor enabled

*Note: Software is allowed to write this bit only when ADSTART = 0 (which ensures that no conversion is ongoing).*

#### Bit 22 VREFEN: V<sub>REFINT</sub> enable

This bit is set and cleared by software to enable/disable the V<sub>REFINT</sub>.

0: V<sub>REFINT</sub> disabled

1: V<sub>REFINT</sub> enabled

*Note: Software is allowed to write this bit only when ADSTART = 0 (which ensures that no conversion is ongoing).*

Bits 21:0 Reserved, must be kept at reset value.

## 13.12 ADC register map

The following table summarizes the ADC registers.

**Table 50. ADC register map and reset values**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x00	ADC_ISR	Res.															
	Reset value																
0x04	ADC_IER	Res.															
	Reset value																

**Table 50. ADC register map and reset values (continued)**

Offset	Register	Field	Description	Type	Reset value	Min	Max	Unit	
0x08	ADC_CR	ADCAL	ADC calibration enable	Bit	0	0	1		
		Res.		Bit	0	0	1		
0x0C	ADC_CFGR1	CKMODE[1:0]	CK MODE	Bit	0	0	1		
		AWDCH[4:0]		Bit	0	0	1		
0x10	ADC_CFGR2	AWDEN		Bit	0	0	1		
		AWDSGL		Bit	0	0	1		
0x14	ADC_SMPR	Res.		Bit	0	0	1		
		Res.		Bit	0	0	1		
0x18	Reserved								
0x1C	Reserved								
0x20	ADC_TR	HT[11:0]							
		Res.		Bit	1	1	1		
0x24	Reserved								
		LT[11:0]							
0x28	ADC_CHSELR	CHSEL18		Bit	0	0	1		
		CHSEL17		Bit	0	0	1		
0x2C	Reserved	CHSEL16		Bit	0	0	1		
		CHSEL15		Bit	0	0	1		
0x30		CHSEL14		Bit	0	0	1		
		CHSEL13		Bit	0	0	1		
0x34		CHSEL12		Bit	0	0	1		
		CHSEL11		Bit	0	0	1		
0x38	Reserved	CHSEL10		Bit	0	0	1		
		CHSEL9		Bit	0	0	1		
0x3C		CHSEL8		Bit	0	0	1		
		CHSEL7		Bit	0	0	1		
0x40	ADC_DR	EXTEN[1:0]		Bit	0	0	1		
		DATA[15:0]							
...:	Reserved	Res.		Bit	0	0	1		
		Res.		Bit	0	0	1		
...:	Reserved	Res.		Bit	0	0	1		
		Res.		Bit	0	0	1		
...:	Reserved	Res.		Bit	0	0	1		
		Res.		Bit	0	0	1		
0x308	ADC_CCR	RES[2:0]		Bit	0	0	1		
		ADSTP		Bit	0	0	1		
...:	Reset value	SCANDIR		Bit	0	0	1		
		ADCFG		Bit	0	0	1		
...:	Reset value	ADDIS		Bit	0	0	1		
		ADEN		Bit	0	0	1		
...:	Reset value	SMP[2:0]		Bit	0	0	1		
		Res.		Bit	0	0	1		

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.



## 14 Digital-to-analog converter (DAC)

This section applies to STM32F05x, STM32F07x and STM32F09x devices only. The second DAC channel (DAC\_OUT2) and some other features are available only on STM32F07x and STM32F09x devices.

### 14.1 Introduction

The DAC module is a 12-bit, voltage output digital-to-analog converter. The DAC can be configured in 8- or 12-bit mode and may be used in conjunction with the DMA controller. In 12-bit mode, the data could be left- or right-aligned. An input reference voltage,  $V_{DDA}$  (shared with ADC), is available. The output can optionally be buffered for higher current drive.

### 14.2 DAC main features

The devices integrate one 12-bit DAC channel DAC\_OUT1. A second channel DAC\_OUT2 is available on STM32F07x and STM32F09x devices.

DAC main features are the following:

- Left or right data alignment in 12-bit mode
- Synchronized update capability
- Noise-wave generation (STM32F07x and STM32F09x devices)
- Triangular-wave generation (STM32F07x and STM32F09x devices)
- Independent or simultaneous conversions (dual mode only)
- DMA capability
- DMA underrun error detection
- External triggers for conversion
- Programmable internal buffer
- Input voltage reference,  $V_{DDA}$

*Figure 48* shows the block diagram of a DAC channel and *Table 51* gives the pin description.

Figure 48. DAC block diagram

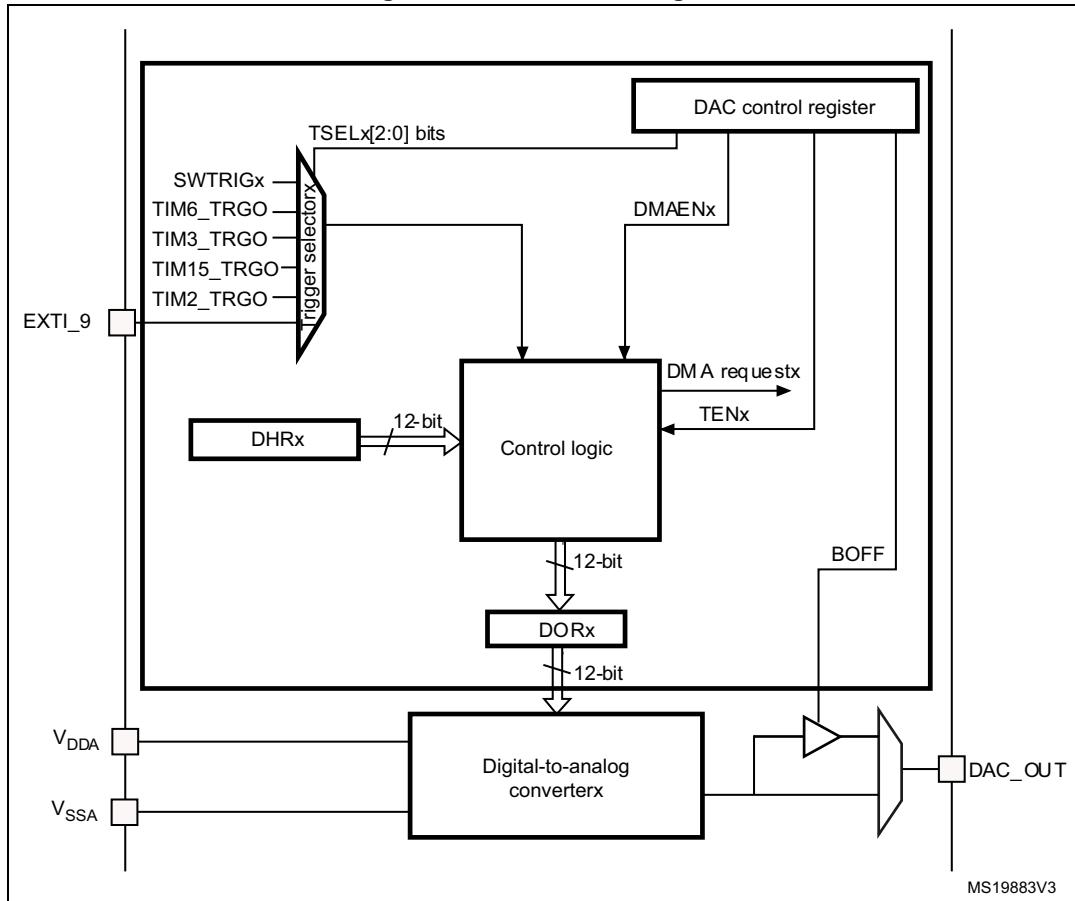


Table 51. DAC pins

Name	Signal type	Remarks
V <sub>DDA</sub>	Input, analog supply	Analog power supply
V <sub>SSA</sub>	Input, analog supply ground	Ground for analog power supply
DAC_OUT	Analog output signal	DAC channelx analog output

**Note:** Once DAC\_Channelx is enabled, the corresponding GPIO pin (PA4 or PA5) is automatically connected to the analog converter output (DAC\_OUTx). In order to avoid parasitic consumption, the PA4 or PA5 pin should first be configured to analog (AIN).

### 14.3 DAC output buffer enable

The DAC integrates one output buffer that can be used to reduce the output impedance and to drive external loads directly without having to add an external operational amplifier.

The DAC channel output buffers can be enabled and disabled through the corresponding BOFFx bit in the DAC\_CR register.

## 14.4 DAC channel enable

Each DAC channel can be powered on by setting the corresponding ENx bit in the DAC\_CR register. Each DAC channel is then enabled after a startup time  $t_{WAKEUP}$ .

**Note:** *The ENx bit enables the analog DAC Channelx macrocell only. The DAC Channelx digital interface is enabled even if the ENx bit is reset.*

## 14.5 Single mode functional description

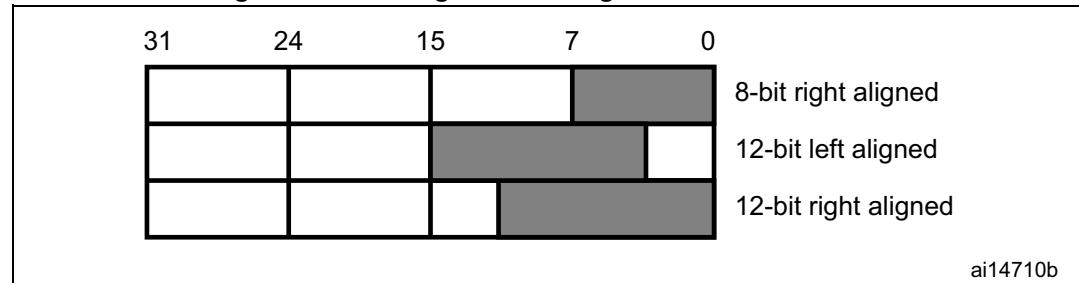
### 14.5.1 DAC data format

There are three possibilities:

- 8-bit right alignment: the software has to load data into the DAC\_DHR8Rx [7:0] bits (stored into the DHRx[11:4] bits)
- 12-bit left alignment: the software has to load data into the DAC\_DHR12Lx [15:4] bits (stored into the DHRx[11:0] bits)
- 12-bit right alignment: the software has to load data into the DAC\_DHR12Rx [11:0] bits (stored into the DHRx[11:0] bits)

Depending on the loaded DAC\_DHRyyx register, the data written by the user is shifted and stored into the corresponding DHRx (data holding registerx, which are internal non-memory-mapped registers). The DHRx register is then loaded into the DORx register either automatically, by software trigger or by an external event trigger.

**Figure 49. Data registers in single DAC channel mode**

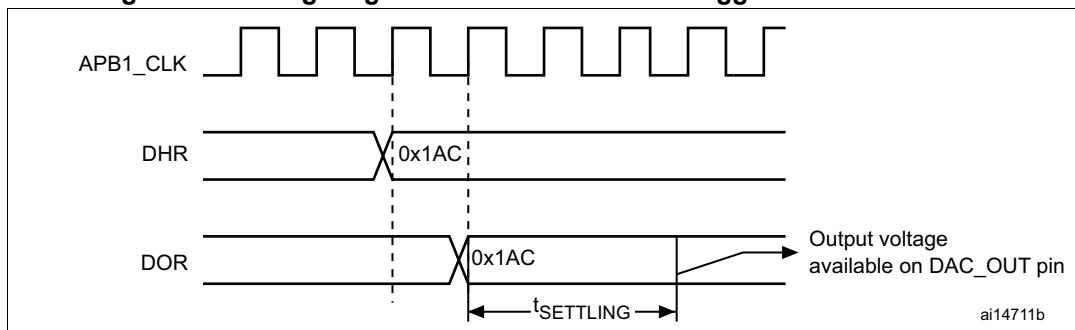


### 14.5.2 DAC channel conversion

The DAC\_DORx cannot be written directly and any data transfer to the DAC channelx must be performed by loading the DAC\_DHRx register (write to DAC\_DHR8Rx, DAC\_DHR12Lx, DAC\_DHR12Rx).

Data stored in the DAC\_DHRx register are automatically transferred to the DAC\_DORx register after one APB clock cycle, if no hardware trigger is selected (TENx bit in DAC\_CR register is reset). However, when a hardware trigger is selected (TENx bit in DAC\_CR register is set) and a trigger occurs, the transfer is performed three PCLK clock cycles later.

When DAC\_DORx is loaded with the DAC\_DHRx contents, the analog output voltage becomes available after a time  $t_{SETTLING}$  that depends on the power supply voltage and the analog output load.

**Figure 50. Timing diagram for conversion with trigger disabled TEN = 0**

### Independent trigger with single LFSR generation

To configure the DAC in this conversion mode (see [Section 14.7: Noise generation\(STM32F07x and STM32F09x devices\)](#)), the following sequence is required:

1. Set the DAC channel trigger enable bit TENx.
2. Configure the trigger source by setting TSELx[2:0] bits.
3. Configure the DAC channel WAVEx[1:0] bits as “01” and the same LFSR mask value in the MAMPx[3:0] bits
4. Load the DAC channel data into the desired DAC\_DHRx register (DHR12RD, DHR12LD or DHR8RD).

When a DAC channelx trigger arrives, the LFSRx counter, with the same mask, is added to the DHRx register and the sum is transferred into DAC\_DORx (three APB clock cycles later). Then the LFSRx counter is updated.

### Independent trigger with single triangle generation

To configure the DAC in this conversion mode (see [Section 14.8: Triangle-wave generation \(STM32F07x and STM32F09x devices\)](#)), the following sequence is required:

1. Set the DAC channelx trigger enable TENx bits.
2. Configure the trigger source by setting TSELx[2:0] bits.
3. Configure the DAC channelx WAVEx[1:0] bits as “1x” and the same maximum amplitude value in the MAMPx[3:0] bits
4. Load the DAC channelx data into the desired DAC\_DHRx register. (DHR12RD, DHR12LD or DHR8RD).

When a DAC channelx trigger arrives, the DAC channelx triangle counter, with the same triangle amplitude, is added to the DHRx register and the sum is transferred into DAC\_DORx (three APB clock cycles later). The DAC channelx triangle counter is then updated.

### 14.5.3 DAC output voltage

Digital inputs are converted to output voltages on a linear conversion between 0 and V<sub>DDA</sub>.

The analog output voltages on each DAC channel pin are determined by the following equation:

$$\text{DACoutput} = V_{\text{DDA}} \times \frac{\text{DOR}}{4096}$$

#### 14.5.4 DAC trigger selection

If the TENx control bit is set, conversion can then be triggered by an external event (timer counter, external interrupt line). The TSELx[2:0] control bits determine which possible events will trigger conversion as shown in [Table 52](#).

**Table 52. External triggers**

Source	Type	TSEL[2:0]
TIM6_TRGO event	Internal signal from on-chip timers	000
TIM3_TRGO event		001
TIM7_TRGO event		010
TIM15_TRGO event		011
TIM2_TRGO event		100
Reserved		101
EXTI line9	External pin	110
SWTRIG	Software control bit	111

Each time a DAC interface detects a rising edge on the selected timer TRGO output, or on the selected external interrupt line 9, the last data stored into the DAC\_DHRx register are transferred into the DAC\_DORx register. The DAC\_DORx register is updated three APB cycles after the trigger occurs.

If the software trigger is selected, the conversion starts once the SWTRIG bit is set. SWTRIG is reset by hardware once the DAC\_DORx register has been loaded with the DAC\_DHRx register contents.

*Note:* *TSELx[2:0] bit cannot be changed when the ENx bit is set. When software trigger is selected, the transfer from the DAC\_DHRx register to the DAC\_DORx register takes only one APB clock cycle.*

### 14.6 Dual-mode functional description (STM32F07x and STM32F09x devices)

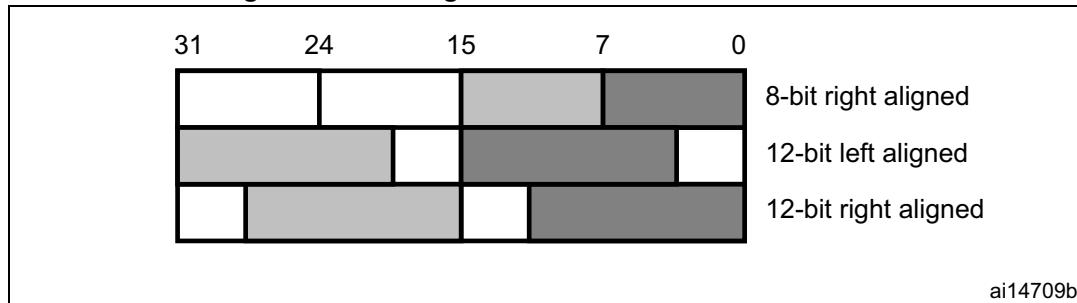
#### 14.6.1 DAC data format

In Dual DAC channel mode, there are three possibilities:

- 8-bit right alignment: data for DAC channel1 to be loaded in the DAC\_DHR8RD [7:0] bits (stored in the DHR1[11:4] bits) and data for DAC channel2 to be loaded in the DAC\_DHR8RD [15:8] bits (stored in the DHR2[11:4] bits)
- 12-bit left alignment: data for DAC channel1 to be loaded into the DAC\_DHR12LD [15:4] bits (stored into the DHR1[11:0] bits) and data for DAC channel2 to be loaded into the DAC\_DHR12LD [31:20] bits (stored in the DHR2[11:0] bits)
- 12-bit right alignment: data for DAC channel1 to be loaded into the DAC\_DHR12RD [11:0] bits (stored in the DHR1[11:0] bits) and data for DAC channel2 to be loaded into the DAC\_DHR12LD [27:16] bits (stored in the DHR2[11:0] bits)

Depending on the loaded DAC\_DHRyyD register, the data written by the user is shifted and stored in DHR1 and DHR2 (data holding registers, which are internal non-memory-mapped registers). The DHR1 and DHR2 registers are then loaded into the DOR1 and DOR2 registers, respectively, either automatically, by software trigger or by an external event trigger.

**Figure 51. Data registers in dual DAC channel mode**



## 14.6.2 DAC channel conversion in dual mode

The DAC channel conversion in dual mode is performed in the same way as in single mode (refer to [Section 14.5.2](#)) except that the data have to be loaded by writing to DAC\_DHR8Rx, DAC\_DHR12Lx, DAC\_DHR12Rx, DAC\_DHR8RD, DAC\_DHR12LD or DAC\_DHR12RD.

## 14.6.3 Description of dual conversion modes

To efficiently use the bus bandwidth in applications that require the two DAC channels at the same time, three dual registers are implemented: DHR8RD, DHR12RD and DHR12LD. A unique register access is then required to drive both DAC channels at the same time.

Eleven conversion modes are possible using the two DAC channels and these dual registers. All the conversion modes can nevertheless be obtained using separate DHRx registers if needed.

All modes are described in the paragraphs below.

Refer to [Section 14.5.2: DAC channel conversion](#) for details on the APB bus (APB or APB1) that clocks the DAC conversions.

### Independent trigger without wave generation

To configure the DAC in this conversion mode, the following sequence is required:

1. Set the two DAC channel trigger enable bits TEN1 and TEN2
2. Configure different trigger sources by setting different values in the TSEL1[2:0] and TSEL2[2:0] bits
3. Load the dual DAC channel data into the desired DHR register (DAC\_DHR12RD, DAC\_DHR12LD or DAC\_DHR8RD)

When a DAC channel1 trigger arrives, the DHR1 register is transferred into DAC\_DOR1 (three APB clock cycles later).

When a DAC channel2 trigger arrives, the DHR2 register is transferred into DAC\_DOR2 (three APB clock cycles later).

For code example refer to the Appendix section [A.8.1: Independent trigger without wave generation code example](#)

### Independent trigger with single LFSR generation

To configure the DAC in this conversion mode (refer to [Section 14.7: Noise generation\(STM32F07x and STM32F09x devices\)](#)), the following sequence is required:

1. Set the two DAC channel trigger enable bits TEN1 and TEN2
2. Configure different trigger sources by setting different values in the TSEL1[2:0] and TSEL2[2:0] bits
3. Configure the two DAC channel WAVEx[1:0] bits as “01” and the same LFSR mask value in the MAMPx[3:0] bits
4. Load the dual DAC channel data into the desired DHR register (DHR12RD, DHR12LD or DHR8RD)

For code example refer to the Appendix section [A.8.2: Independent trigger with single LFSR generation code example](#)

When a DAC channel1 trigger arrives, the LFSR1 counter, with the same mask, is added to the DHR1 register and the sum is transferred into DAC\_DOR1 (three APB clock cycles later). Then the LFSR1 counter is updated.

When a DAC channel2 trigger arrives, the LFSR2 counter, with the same mask, is added to the DHR2 register and the sum is transferred into DAC\_DOR2 (three APB clock cycles later). Then the LFSR2 counter is updated.

### Independent trigger with different LFSR generation

To configure the DAC in this conversion mode (refer to [Section 14.7: Noise generation\(STM32F07x and STM32F09x devices\)](#)), the following sequence is required:

1. Set the two DAC channel trigger enable bits TEN1 and TEN2
2. Configure different trigger sources by setting different values in the TSEL1[2:0] and TSEL2[2:0] bits
3. Configure the two DAC channel WAVEx[1:0] bits as “01” and set different LFSR masks values in the MAMP1[3:0] and MAMP2[3:0] bits
4. Load the dual DAC channel data into the desired DHR register (DAC\_DHR12RD, DAC\_DHR12LD or DAC\_DHR8RD)

For code example refer to the Appendix section [A.8.3: Independent trigger with different LFSR generation code example](#).

When a DAC channel1 trigger arrives, the LFSR1 counter, with the mask configured by MAMP1[3:0], is added to the DHR1 register and the sum is transferred into DAC\_DOR1 (three APB clock cycles later). Then the LFSR1 counter is updated.

When a DAC channel2 trigger arrives, the LFSR2 counter, with the mask configured by MAMP2[3:0], is added to the DHR2 register and the sum is transferred into DAC\_DOR2 (three APB clock cycles later). Then the LFSR2 counter is updated.

### Independent trigger with single triangle generation

To configure the DAC in this conversion mode (refer to [Section 14.8: Triangle-wave generation \(STM32F07x and STM32F09x devices\)](#)), the following sequence is required:

1. Set the DAC channelx trigger enable TENx bits.
2. Configure different trigger sources by setting different values in the TSELx[2:0] bits
3. Configure the DAC channelx WAVEx[1:0] bits as “1x” and the same maximum amplitude value in the MAMPx[3:0] bits
4. Load the DAC channelx data into the desired DAC\_DHRx register.

For code example refer to the Appendix section [A.8.4: Independent trigger with single triangle generation code example](#).

Refer to [Section 14.5.2: DAC channel conversion](#) for details on the APB bus (APB or APB1) that clocks the DAC conversions.

When a DAC channelx trigger arrives, the DAC channelx triangle counter, with the same triangle amplitude, is added to the DHRx register and the sum is transferred into DAC\_DORx (three APB clock cycles later). The DAC channelx triangle counter is then updated.

### **Independent trigger with different triangle generation**

To configure the DAC in this conversion mode (refer to [Section 14.8: Triangle-wave generation \(STM32F07x and STM32F09x devices\)](#)), the following sequence is required:

1. Set the DAC channelx trigger enable TENx bits.
2. Configure different trigger sources by setting different values in the TSELx[2:0] bits
3. Configure the DAC channelx WAVEx[1:0] bits as “1x” and set different maximum amplitude values in the MAMPx[3:0] bits
4. Load the DAC channelx data into the desired DAC\_DHRx register.

For code example refer to the Appendix section [A.8.5: Independent trigger with different triangle generation code example](#).

When a DAC channelx trigger arrives, the DAC channelx triangle counter, with a triangle amplitude configured by MAMPx[3:0], is added to the DHRx register and the sum is transferred into DAC\_DORx (three APB clock cycles later). The DAC channelx triangle counter is then updated.

### **Simultaneous software start**

To configure the DAC in this conversion mode, the following sequence is required:

1. Load the dual DAC channel data to the desired DHR register (DAC\_DHR12RD, DAC\_DHR12LD or DAC\_DHR8RD)

In this configuration, one APB clock cycles).

For code example refer to the Appendix section [A.8.6: Simultaneous software start code example](#).

### **Simultaneous trigger without wave generation**

To configure the DAC in this conversion mode, the following sequence is required:

1. Set the two DAC channel trigger enable bits TEN1 and TEN2
2. Configure the same trigger source for both DAC channels by setting the same value in the TSEL1[2:0] and TSEL2[2:0] bits
3. Load the dual DAC channel data to the desired DHR register (DAC\_DHR12RD, DAC\_DHR12LD or DAC\_DHR8RD)

When a trigger arrives, the DHR1 and DHR2 registers are transferred into DAC\_DOR1 and DAC\_DOR2, respectively (after three APB clock cycles).

For code example refer to the Appendix section [A.8.7: Simultaneous trigger without wave generation code example](#).

### Simultaneous trigger with single LFSR generation

To configure the DAC in this conversion mode (refer to [Section 14.7: Noise generation\(STM32F07x and STM32F09x devices\)](#)), the following sequence is required:

1. Set the two DAC channel trigger enable bits TEN1 and TEN2
2. Configure the same trigger source for both DAC channels by setting the same value in the TSEL1[2:0] and TSEL2[2:0] bits
3. Configure the two DAC channel WAVEEx[1:0] bits as “01” and the same LFSR mask value in the MAMPx[3:0] bits
4. Load the dual DAC channel data to the desired DHR register (DHR12RD, DHR12LD or DHR8RD)

For code example refer to the Appendix section [A.8.8: Simultaneous trigger with single LFSR generation code example](#).

When a trigger arrives, the LFSR1 counter, with the same mask, is added to the DHR1 register and the sum is transferred into DAC\_DOR1 (three APB clock cycles later). The LFSR1 counter is then updated. At the same time, the LFSR2 counter, with the same mask, is added to the DHR2 register and the sum is transferred into DAC\_DOR2 (three APB clock cycles later). The LFSR2 counter is then updated.

### Simultaneous trigger with different LFSR generation

To configure the DAC in this conversion mode (refer to [Section 14.7: Noise generation\(STM32F07x and STM32F09x devices\)](#)), the following sequence is required:

1. Set the two DAC channel trigger enable bits TEN1 and TEN2
2. Configure the same trigger source for both DAC channels by setting the same value in the TSEL1[2:0] and TSEL2[2:0] bits
3. Configure the two DAC channel WAVEEx[1:0] bits as “01” and set different LFSR mask values using the MAMP1[3:0] and MAMP2[3:0] bits
4. Load the dual DAC channel data into the desired DHR register (DAC\_DHR12RD, DAC\_DHR12LD or DAC\_DHR8RD)

For code example refer to the Appendix section [A.8.9: Simultaneous trigger with different LFSR generation code example](#).

When a trigger arrives, the LFSR1 counter, with the mask configured by MAMP1[3:0], is added to the DHR1 register and the sum is transferred into DAC\_DOR1 (three APB clock cycles later). The LFSR1 counter is then updated.

At the same time, the LFSR2 counter, with the mask configured by MAMP2[3:0], is added to the DHR2 register and the sum is transferred into DAC\_DOR2 (three APB clock cycles later). The LFSR2 counter is then updated.

### Simultaneous trigger with single triangle generation

To configure the DAC in this conversion mode (refer to [Section 14.8: Triangle-wave generation \(STM32F07x and STM32F09x devices\)](#)), the following sequence is required:

1. Set the DAC channelx trigger enable TEN1x bits.
2. Configure the same trigger source for both DAC channels by setting the same value in the TSELx[2:0] bits.
3. Configure the DAC channelx WAVEx[1:0] bits as “1x” and the same maximum amplitude value using the MAMPx[3:0] bits
4. Load the DAC channelx data into the desired DAC\_DHRx registers.

For code example refer to the Appendix section [A.8.10: Simultaneous trigger with single triangle generation code example](#).

When a trigger arrives, the DAC channelx triangle counter, with the same triangle amplitude, is added to the DHRx register and the sum is transferred into DAC\_DORx (three APB clock cycles later). The DAC channelx triangle counter is then updated.

#### Simultaneous trigger with different triangle generation

To configure the DAC in this conversion mode ‘refer to [Section 14.8: Triangle-wave generation \(STM32F07x and STM32F09x devices\)](#)’, the following sequence is required:

1. Set the DAC channelx trigger enable TENx bits.
2. Configure the same trigger source for DAC channelx by setting the same value in the TSELx[2:0] bits
3. Configure the DAC channelx WAVEx[1:0] bits as “1x” and set different maximum amplitude values in the MAMPx[3:0] bits.
4. Load the DAC channelx data into the desired DAC\_DHRx registers.

For code example refer to the Appendix section [A.8.11: Simultaneous trigger with different triangle generation code example](#).

When a trigger arrives, the DAC channelx triangle counter, with a triangle amplitude configured by MAMPx[3:0], is added to the DHRx register and the sum is transferred into DAC\_DORx (three APB clock cycles later). Then the DAC channelx triangle counter is updated.

### 14.6.4 DAC output voltage

Refer to [Section 14.5.3: DAC output voltage](#).

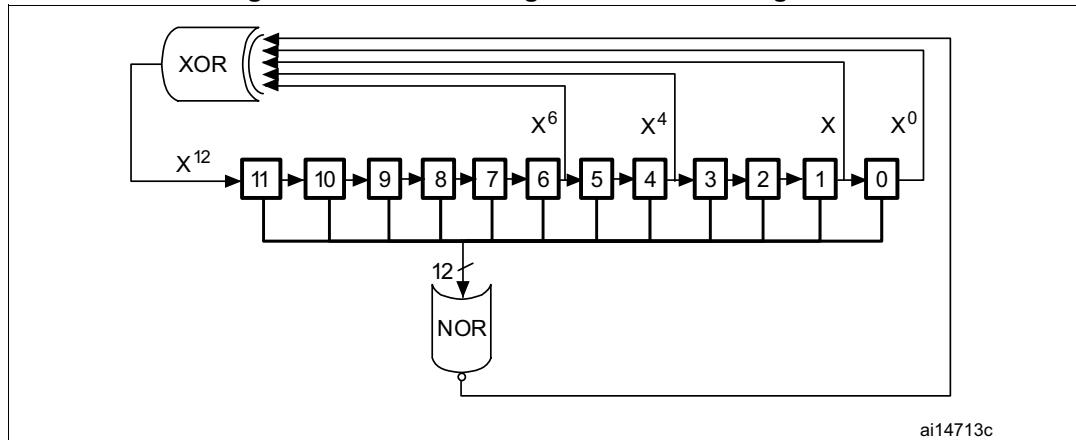
### 14.6.5 DAC trigger selection

Refer to [Section 14.5.4: DAC trigger selection](#)

## 14.7 Noise generation(STM32F07x and STM32F09x devices)

In order to generate a variable-amplitude pseudonoise, an LFSR (linear feedback shift register) is available. DAC noise generation is selected by setting WAVE<sub>x</sub>[1:0] to “01”. The preloaded value in LFSR is 0xAAA. This register is updated three APB clock cycles after each trigger event, following a specific calculation algorithm.

**Figure 52. DAC LFSR register calculation algorithm**

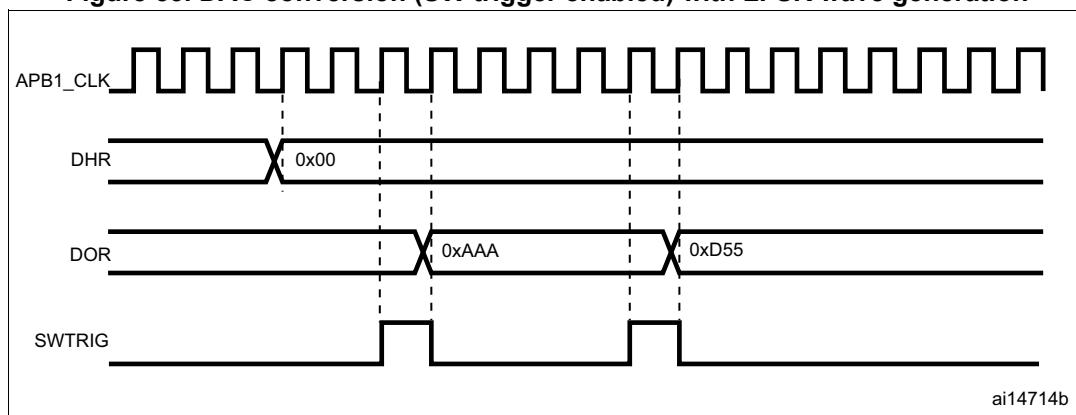


The LFSR value, that may be masked partially or totally by means of the MAMP<sub>x</sub>[3:0] bits in the DAC\_CR register, is added up to the DAC\_DHR<sub>x</sub> contents without overflow and this value is then stored into the DAC\_DOR<sub>x</sub> register.

If LFSR is 0x0000, a ‘1’ is injected into it (antilock-up mechanism).

It is possible to reset LFSR wave generation by resetting the WAVE<sub>x</sub>[1:0] bits.

**Figure 53. DAC conversion (SW trigger enabled) with LFSR wave generation**



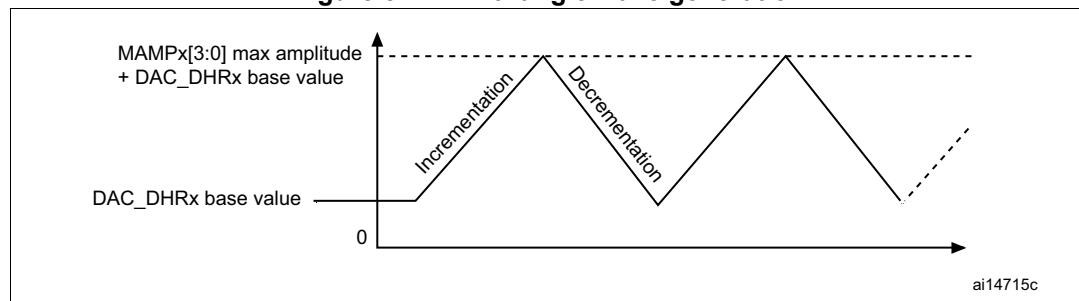
**Note:** The DAC trigger must be enabled for noise generation by setting the TEN<sub>x</sub> bit in the DAC\_CR register.

## 14.8 Triangle-wave generation (STM32F07x and STM32F09x devices)

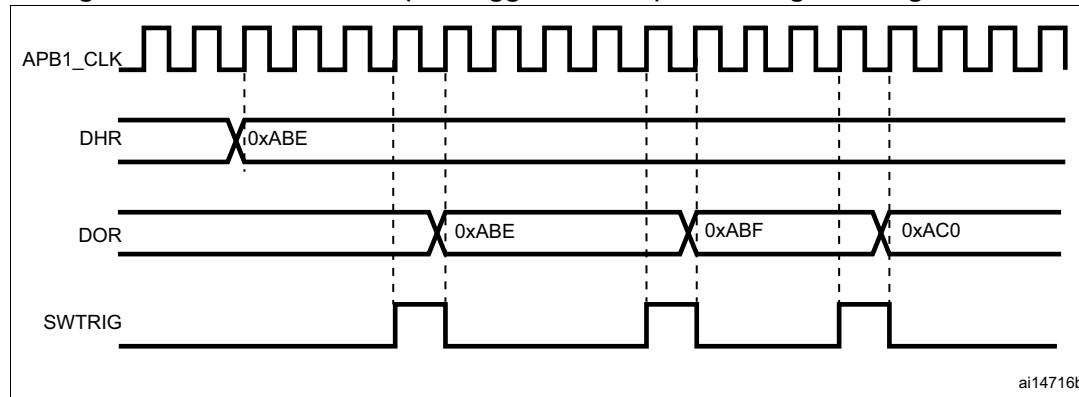
It is possible to add a small-amplitude triangular waveform on a DC or slowly varying signal. DAC triangle-wave generation is selected by setting WAVE[1:0] to “10”. The amplitude is configured through the MAMPx[3:0] bits in the DAC\_CR register. An internal triangle counter is incremented three APB clock cycles after each trigger event. The value of this counter is then added to the DAC\_DHRx register without overflow and the sum is stored into the DAC\_DORx register. The triangle counter is incremented as long as it is less than the maximum amplitude defined by the MAMPx[3:0] bits. Once the configured amplitude is reached, the counter is decremented down to 0, then incremented again and so on.

It is possible to reset triangle wave generation by resetting the WAVE[1:0] bits.

**Figure 54. DAC triangle wave generation**



**Figure 55. DAC conversion (SW trigger enabled) with triangle wave generation**



Note:

The DAC trigger must be enabled for triangle generation by setting the TENx bit in the DAC\_CR register.

The MAMPx[3:0] bits must be configured before enabling the DAC, otherwise they cannot be changed.

## 14.9 DMA request

Each DAC channel has a DMA capability. Two DMA channels are used to service DAC channel DMA requests.

A DAC DMA request is generated when an external trigger (but not a software trigger) occurs while the DMAENx bit is set. The value of the DAC\_DHRx register is then transferred to the DAC\_DORx register.

In dual mode, if both DMAENx bits are set, two DMA requests are generated. If only one DMA request is needed, user should set only the corresponding DMAENx bit. In this way, the application can manage both DAC channels in dual mode by using one DMA request and a unique DMA channel.

For code example refer to the Appendix section [A.8.12: DMA initialization code example](#).

### DMA underrun

The DAC DMA request is not queued so that if a second external trigger arrives before the acknowledgment for the first external trigger is received (first request), then no new request is issued and the DMA channelx underrun flag DMAUDRx in the DAC\_SR register is set, reporting the error condition. DMA data transfers are then disabled and no further DMA request is treated. The DAC channelx continues to convert old data.

The software should clear the DMAUDRx flag by writing “1”, clear the DMAEN bit of the used DMA stream and re-initialize both DMA and DAC channelx to restart the transfer correctly. The software should modify the DAC trigger conversion frequency or lighten the DMA workload to avoid a new DMA. Finally, the DAC conversion can be resumed by enabling both DMA data transfer and conversion trigger.

For each DAC channel, an interrupt is also generated if the corresponding DMAUDRIEx bit in the DAC\_CR register is enabled.

## 14.10 DAC registers

Refer to [Section 1.2 on page 42](#) for a list of abbreviations used in register descriptions.

The peripheral registers have to be accessed by words (32-bit).

### 14.10.1 DAC control register (DAC\_CR)

Address offset: 0x000

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	DMAU DRIE2	DMA EN2	MAMP2[3:0]				WAVE2[1:0]		TSEL2[2:0]			TEN2	BOFF2	EN2
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	DMAU DRIE1	DMA EN1	MAMP1[3:0]				WAVE1[1:0]		TSEL1[2:0]			TEN1	BOFF1	EN1
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30 Reserved, must be kept at reset value.

Bit 29 **DMAUDRIE2**: DAC channel2 DMA underrun interrupt enable

This bit is set and cleared by software.

0: DAC channel2 DMA underrun interrupt disabled

1: DAC channel2 DMA underrun interrupt enabled

*Note: This bit is available in dual mode only. It is reserved in single mode.*

Bit 28 **DMAEN2**: DAC channel2 DMA enable

This bit is set and cleared by software.

0: DAC channel2 DMA mode disabled

1: DAC channel2 DMA mode enabled

*Note: This bit is available in dual mode only. It is reserved in single mode.*

Bits 27:24 **MAMP2[3:0]**: DAC channel2 mask/amplitude selector

These bits are written by software to select mask in wave generation mode or amplitude in triangle generation mode.

0000: Unmask bit0 of LFSR/ triangle amplitude equal to 1

0001: Unmask bits[1:0] of LFSR/ triangle amplitude equal to 3

0010: Unmask bits[2:0] of LFSR/ triangle amplitude equal to 7

0011: Unmask bits[3:0] of LFSR/ triangle amplitude equal to 15

0100: Unmask bits[4:0] of LFSR/ triangle amplitude equal to 31

0101: Unmask bits[5:0] of LFSR/ triangle amplitude equal to 63

0110: Unmask bits[6:0] of LFSR/ triangle amplitude equal to 127

0111: Unmask bits[7:0] of LFSR/ triangle amplitude equal to 255

1000: Unmask bits[8:0] of LFSR/ triangle amplitude equal to 511

1001: Unmask bits[9:0] of LFSR/ triangle amplitude equal to 1023

1010: Unmask bits[10:0] of LFSR/ triangle amplitude equal to 2047

≥1011: Unmask bits[11:0] of LFSR/ triangle amplitude equal to 4095

*Note: These bits are available only in dual mode when wave generation is supported.*

*Otherwise, they are reserved and must be kept at reset value.*

Bits 23:22 **WAVE2[1:0]**: DAC channel2 noise/triangle wave generation enable

These bits are set/reset by software.

00: wave generation disabled

01: Noise wave generation enabled

1x: Triangle wave generation enabled

*Note: Only used if bit TEN2 = 1 (DAC channel2 trigger enabled)*

*These bits are available only in dual mode when wave generation is supported.*

*Otherwise, they are reserved and must be kept at reset value.*

Bits 21:19 **TSEL2[2:0]**: DAC channel2 trigger selection

These bits select the external event used to trigger DAC channel2

000: Timer 6 TRGO event

001: Timer 3 TRGO event

010: Timer 7 TRGO event

011: Timer 15 TRGO event

100: Timer 2 TRGO event

101: Reserved

110: EXTI line9

111: Software trigger

*Note: Only used if bit TEN2 = 1 (DAC channel2 trigger enabled).*

*These bits are available in dual mode only. They are reserved in single mode.*

Bit 18 **TEN2**: DAC channel2 trigger enable

This bit is set and cleared by software to enable/disable DAC channel2 trigger

0: DAC channel2 trigger disabled and data written into the DAC\_DHRx register are transferred one APBclock cycle later to the DAC\_DOR2 register

1: DAC channel2 trigger enabled and data from the DAC\_DHRx register are transferred three APB clock cycles later to the DAC\_DOR2 register

*Note: When software trigger is selected, the transfer from the DAC\_DHRx register to the DAC\_DOR2 register takes only one APB clock cycle.*

*Note: This bit is available in dual mode only. It is reserved in single mode.*

Bit 17 **BOFF2**: DAC channel2 output buffer disable

This bit is set and cleared by software to enable/disable DAC channel2 output buffer.

0: DAC channel2 output buffer enabled

1: DAC channel2 output buffer disabled

*Note: This bit is available in dual mode only. It is reserved in single mode.*

Bit 16 **EN2**: DAC channel2 enable

This bit is set and cleared by software to enable/disable DAC channel2.

0: DAC channel2 disabled

1: DAC channel2 enabled

*Note: This bit is available in dual mode only. It is reserved in single mode.*

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

Bit 13 **DMAUDRIE1**: DAC channel1 DMA Underrun Interrupt enable

This bit is set and cleared by software.

0: DAC channel1 DMA Underrun Interrupt disabled

1: DAC channel1 DMA Underrun Interrupt enabled

Bit 12 **DMAEN1**: DAC channel1 DMA enable

This bit is set and cleared by software.

0: DAC channel1 DMA mode disabled

1: DAC channel1 DMA mode enabled

Bits 11:8 **MAMP1[3:0]**: DAC channel1 mask/amplitude selector

These bits are written by software to select mask in wave generation mode or amplitude in triangle generation mode.

0000: Unmask bit0 of LFSR/ triangle amplitude equal to 1

0001: Unmask bits[1:0] of LFSR/ triangle amplitude equal to 3

0010: Unmask bits[2:0] of LFSR/ triangle amplitude equal to 7

0011: Unmask bits[3:0] of LFSR/ triangle amplitude equal to 15

0100: Unmask bits[4:0] of LFSR/ triangle amplitude equal to 31

0101: Unmask bits[5:0] of LFSR/ triangle amplitude equal to 63

0110: Unmask bits[6:0] of LFSR/ triangle amplitude equal to 127

0111: Unmask bits[7:0] of LFSR/ triangle amplitude equal to 255

1000: Unmask bits[8:0] of LFSR/ triangle amplitude equal to 511

1001: Unmask bits[9:0] of LFSR/ triangle amplitude equal to 1023

1010: Unmask bits[10:0] of LFSR/ triangle amplitude equal to 2047

≥ 1011: Unmask bits[11:0] of LFSR/ triangle amplitude equal to 4095

Bits 7:6 **WAVE1[1:0]**: DAC channel1 noise/triangle wave generation enable

These bits are set and cleared by software.

00: Wave generation disabled

01: Noise wave generation enabled

1x: Triangle wave generation enabled

*Note: Only used if bit TEN1 = 1 (DAC channel1 trigger enabled).*

Bits 5:3 **TSEL1[2:0]**: DAC channel1 trigger selection

These bits select the external event used to trigger DAC channel1.

000: Timer 6 TRGO event

001: Timer 3 TRGO event

010: Timer 7 TRGO event

011: Timer 15 TRGO event

100: Timer 2 TRGO event

101: Reserved

110: EXTI line9

111: Software trigger

*Note: Only used if bit TEN1 = 1 (DAC channel1 trigger enabled).*

**Bit 2 **TEN1**: DAC channel1 trigger enable**

This bit is set and cleared by software to enable/disable DAC channel1 trigger.

0: DAC channel1 trigger disabled and data written into the DAC\_DHRx register are transferred one APB clock cycle later to the DAC\_DOR1 register

1: DAC channel1 trigger enabled and data from the DAC\_DHRx register are transferred three APB clock cycles later to the DAC\_DOR1 register

*Note: When software trigger is selected, the transfer from the DAC\_DHRx register to the DAC\_DOR1 register takes only one APB clock cycle.*

**Bit 1 **BOFF1**: DAC channel1 output buffer disable**

This bit is set and cleared by software to enable/disable DAC channel1 output buffer.

0: DAC channel1 output buffer enabled

1: DAC channel1 output buffer disabled

**Bit 0 **EN1**: DAC channel1 enable**

This bit is set and cleared by software to enable/disable DAC channel1.

0: DAC channel1 disabled

1: DAC channel1 enabled

### 14.10.2 DAC software trigger register (DAC\_SWTRIGR)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SWTRIG2	SWTRIG1													

Bits 31:2 Reserved, must be kept at reset value.

Bit 1 **SWTRIG2**: DAC channel2 software trigger

This bit is set and cleared by software to enable/disable the software trigger.

- 0: Software trigger disabled
- 1: Software trigger enabled

*Note: This bit is cleared by hardware (one APB clock cycle later) once the DAC\_DHR2 register value has been loaded into the DAC\_DOR2 register.*

*This bit is available in dual mode only. It is reserved in single mode.*

Bit 0 **SWTRIG1**: DAC channel1 software trigger

This bit is set and cleared by software to enable/disable the software trigger.

- 0: Software trigger disabled
- 1: Software trigger enabled

*Note: This bit is cleared by hardware (one APB clock cycle later) once the DAC\_DHR1 register value has been loaded into the DAC\_DOR1 register.*

### 14.10.3 DAC channel1 12-bit right-aligned data holding register (DAC\_DHR12R1)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	rw											
DACC1DHR[11:0]															

Bits 31:12 Reserved, must be kept at reset value.

Bits 11:0 **DACC1DHR[11:0]**: DAC channel1 12-bit right-aligned data

These bits are written by software which specifies 12-bit data for DAC channel1.

#### 14.10.4 DAC channel1 12-bit left-aligned data holding register (DAC\_DHR12L1)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DACC1DHR[11:0]												v	Res.	Res.	Res.
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw				

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

Bits 15:4 **DACC1DHR[11:0]**: DAC channel1 12-bit left-aligned data

These bits are written by software which specifies 12-bit data for DAC channel1.

Bits 3:0 Reserved, must be kept at reset value.

#### 14.10.5 DAC channel1 8-bit right-aligned data holding register (DAC\_DHR8R1)

Address offset: 0x10

Reset value: 0x0000 0000

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

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

Bits 7:0 **DACC1DHR[7:0]**: DAC channel1 8-bit right-aligned data

These bits are written by software which specifies 8-bit data for DAC channel1.

#### 14.10.6 DAC channel2 12-bit right-aligned data holding register (DAC\_DHR12R2)

Address offset: 0x14

Reset value: 0x0000 0000

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

DACC2DHR[11:0]

Bits 31:12 Reserved, must be kept at reset value.

Bits 11:0 **DACC2DHR[11:0]**: DAC channel2 12-bit right-aligned data

These bits are written by software which specifies 12-bit data for DAC channel2.

#### 14.10.7 DAC channel2 12-bit left-aligned data holding register (DAC\_DHR12L2)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DACC2DHR[11:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 15:4 **DACC2DHR[11:0]**: DAC channel2 12-bit left-aligned data

These bits are written by software which specify 12-bit data for DAC channel2.

Bits 3:0 Reserved, must be kept at reset value.

#### 14.10.8 DAC channel2 8-bit right-aligned data holding register (DAC\_DHR8R2)

Address offset: 0x1C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	DACC2DHR[7:0]														
								rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 7:0 **DACC2DHR[7:0]**: DAC channel2 8-bit right-aligned data

These bits are written by software which specifies 8-bit data for DAC channel2.

#### 14.10.9 Dual DAC 12-bit right-aligned data holding register (DAC\_DHR12RD)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16		
Res.	Res.	Res.	Res.	DACC2DHR[11:0]													
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
Res.	Res.	Res.	Res.	DACC1DHR[11:0]													
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bits 31:28 Reserved, must be kept at reset value.

Bits 27:16 **DACC2DHR[11:0]**: DAC channel2 12-bit right-aligned data

These bits are written by software which specifies 12-bit data for DAC channel2.

Bits 15:12 Reserved, must be kept at reset value.

Bits 11:0 **DACC1DHR[11:0]**: DAC channel1 12-bit right-aligned data

These bits are written by software which specifies 12-bit data for DAC channel1.

#### 14.10.10 Dual DAC 12-bit left-aligned data holding register (DAC\_DHR12LD)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16		
DACC2DHR[11:0]														Res.	Res.	Res.	Res.
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
DACC1DHR[11:0]														Res.	Res.	Res.	Res.
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw				

Bits 31:20 **DACC2DHR[11:0]**: DAC channel2 12-bit left-aligned data

These bits are written by software which specifies 12-bit data for DAC channel2.

Bits 19:16 Reserved, must be kept at reset value.

Bits 15:4 **DACC1DHR[11:0]**: DAC channel1 12-bit left-aligned data

These bits are written by software which specifies 12-bit data for DAC channel1.

Bits 3:0 Reserved, must be kept at reset value.

#### 14.10.11 Dual DAC 8-bit right-aligned data holding register (DAC\_DHR8RD)

Address offset: 0x28

Reset value: 0x0000 0000

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DACC2DHR[7:0]								DACC1DHR[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 15:8 **DACC2DHR[7:0]**: DAC channel2 8-bit right-aligned data

These bits are written by software which specifies 8-bit data for DAC channel2.

Bits 7:0 **DACC1DHR[7:0]**: DAC channel1 8-bit right-aligned data

These bits are written by software which specifies 8-bit data for DAC channel1.

#### 14.10.12 DAC channel1 data output register (DAC\_DOR1)

Address offset: 0x2C

Reset value: 0x0000 0000

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

Bits 31:12 Reserved, must be kept at reset value.

Bits 11:0 **DACC1DOR[11:0]**: DAC channel1 data output

These bits are read-only, they contain data output for DAC channel1.

#### 14.10.13 DAC channel2 data output register (DAC\_DOR2)

Address offset: 0x30

Reset value: 0x0000 0000

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

Bits 31:12 Reserved, must be kept at reset value.

Bits 11:0 **DACC2DOR[11:0]**: DAC channel2 data output

These bits are read-only, they contain data output for DAC channel2.

#### 14.10.14 DAC status register (DAC\_SR)

Address offset: 0x34

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	DMAUDR2	Res.												
		rc_w1													
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	DMAUDR1	Res.												
		rc_w1													

Bits 31:30 Reserved, must be kept at reset value.

Bit 29 **DMAUDR2**: DAC channel2 DMA underrun flag. Available on STM32F07x and STM32F09x devices.

This bit is set by hardware and cleared by software (by writing it to 1).

0: No DMA underrun error condition occurred for DAC channel2

1: DMA underrun error condition occurred for DAC channel2 (the currently selected trigger is driving DAC channel2 conversion at a frequency higher than the DMA service capability rate)

*Note: This bit is available in dual mode only. It is reserved in single mode.*

Bits 28:14 Reserved, must be kept at reset value.

Bit 13 **DMAUDR1**: DAC channel1 DMA underrun flag

This bit is set by hardware and cleared by software (by writing it to 1).

0: No DMA underrun error condition occurred for DAC channel1

1: DMA underrun error condition occurred for DAC channel1 (the currently selected trigger is driving DAC channel1 conversion at a frequency higher than the DMA service capability rate)

Bits 12:0 Reserved, must be kept at reset value.

#### 14.10.15 DAC register map

*Table 53* summarizes the DAC registers.

**Table 53. DAC register map and reset values**

**Table 53. DAC register map (continued)and reset values (continued)**

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x34	DAC_SR	Res.	Res.	DMAUDR2	Res.	0	Res.																										
	Reset value		0																0														

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 15 Comparator (COMP)

This section applies to STM32F05x and STM32F07x and STM32F09x devices only.

### 15.1 Introduction

STM32F05x and STM32F07x and STM32F09x devices embed two general purpose comparators COMP1 and COMP2, that can be used either as standalone devices (all terminal are available on I/Os) or combined with the timers.

The comparators can be used for a variety of functions including:

- Wake-up from low-power mode triggered by an analog signal,
- Analog signal conditioning,
- Cycle-by-cycle current control loop when combined with the DAC and a PWM output from a timer.

### 15.2 COMP main features

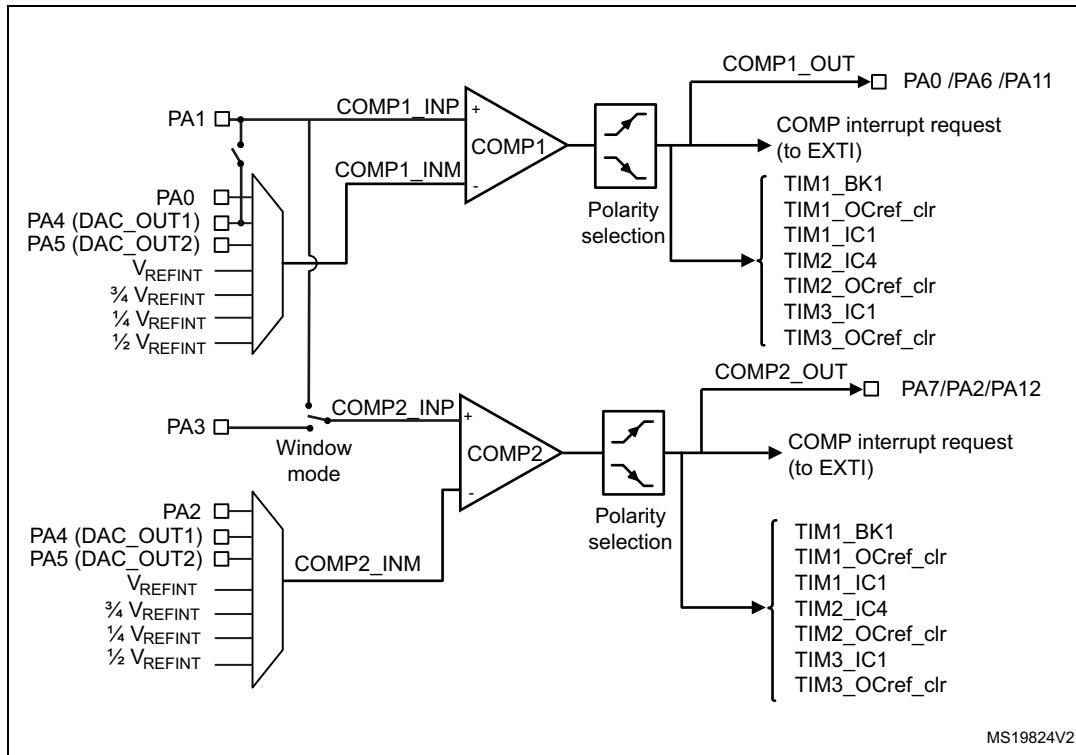
- Rail-to-rail comparators
- Each comparator has positive and configurable negative inputs used for flexible voltage selection:
  - 3 I/O pins
  - DAC
  - Internal reference voltage and three submultiple values (1/4, 1/2, 3/4) provided by scaler (buffered voltage divider)
- Programmable hysteresis
- Programmable speed / consumption
- The outputs can be redirected to an I/O or to timer inputs for triggering:
  - OCREF\_CLR events (for cycle-by-cycle current control)
  - Break events for fast PWM shutdowns
- COMP1 and COMP2 comparators can be combined in a window comparator.
- Each comparator has interrupt generation capability with wake-up from Sleep and Stop modes (through the EXTI controller)

## 15.3 COMP functional description

### 15.3.1 COMP block diagram

The block diagram of the comparators is shown in [Figure 56: Comparator 1 and 2 block diagrams](#).

**Figure 56. Comparator 1 and 2 block diagrams**



### 15.3.2 COMP pins and internal signals

The I/Os used as comparators inputs must be configured in analog mode in the GPIOs registers.

The comparator output can be connected to the I/Os using the alternate function channel given in “Alternate function mapping” table in the datasheet.

The output can also be internally redirected to a variety of timer input for the following purposes:

- Emergency shut-down of PWM signals, using BKIN
- Cycle-by-cycle current control, using OCREF\_CLR inputs
- Input capture for timing measures

It is possible to have the comparator output simultaneously redirected internally and externally.

### 15.3.3 COMP reset and clocks

The COMP clock provided by the clock controller is synchronous with the PCLK (APB clock).

There is no clock enable control bit provided in the RCC controller. Reset and clock enable bits are common for COMP and SYSCFG.

**Important:** The polarity selection logic and the output redirection to the port works independently from the *PCLK* clock. This allows the comparator to work even in Stop mode.

### 15.3.4 Comparator LOCK mechanism

The comparators can be used for safety purposes, such as over-current or thermal protection. For applications having specific functional safety requirements, it is necessary to insure that the comparator programming cannot be altered in case of spurious register access or program counter corruption.

For this purpose, the comparator control and status registers can be write-protected (read-only).

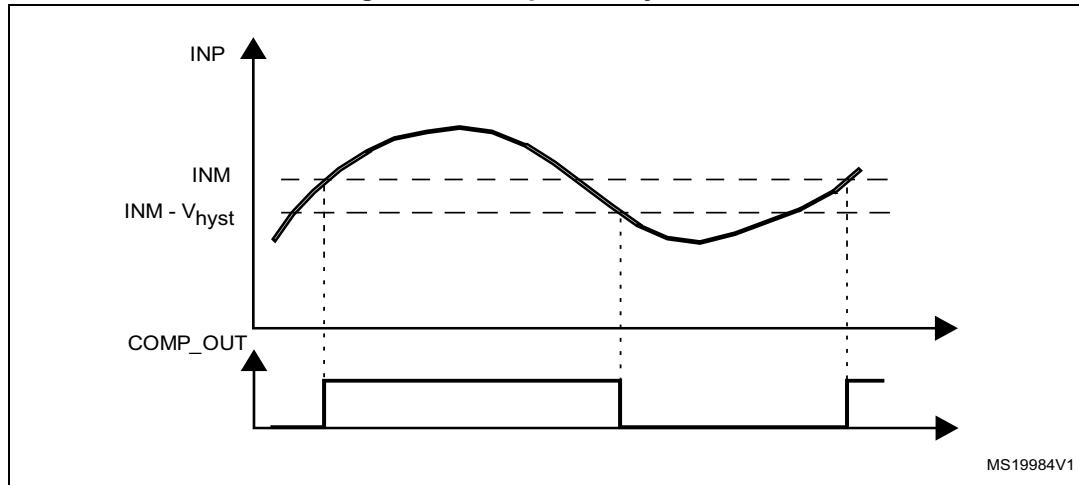
Once the programming is completed, using bits 30:16 and 15:0 of COMP\_CSR, the COMPx LOCK bit can be set to 1. This causes the whole COMP\_CSR register to become read-only, including the COMPx LOCK bit.

The write protection can only be reset by a MCU reset.

### 15.3.5 Hysteresis

The comparator includes a programmable hysteresis to avoid spurious output transitions in case of noisy signals. The hysteresis can be disabled if it is not needed (for instance when exiting from low-power mode) to be able to force the hysteresis value using external components.

Figure 57. Comparator hysteresis



MS19984V1

### 15.3.6 Power mode

The comparator power consumption versus propagation delay can be adjusted to have the optimum trade-off for a given application. The bits COMPxMODE[1:0] in COMP\_CSR register can be programmed as follows:

- 00: High speed / full power
- 01: Medium speed / medium power
- 10: Low speed / low-power
- 11: Very-low speed / ultra-low-power

## 15.4 COMP interrupts

The comparator outputs are internally connected to the Extended interrupts and events controller. Each comparator has its own EXTI line and can generate either interrupts or events. The same mechanism is used to exit from low-power modes.

Refer to Interrupt and events section for more details.

## 15.5 COMP registers

### 15.5.1 COMP control and status register (COMP\_CSR)

Address offset: 0x1C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
COMP2LOCK	COMP2OUT	COMP2HYST [1:0]	COMP2POL	COMP2OUTSEL[2:0]			WNDWEN	COMP2INSEL[2:0]			COMP2MODE [1:0]	Res.	COMP2EN		
rwo	r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r		rw/r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COMP1LOCK	COMP1OUT	COMP1HYST [1:0]	COMP1POL	COMP1OUTSEL[2:0]			Res.	COMP1INSEL[2:0]			COMP1MODE [1:0]	COMP1SW1	COMP1EN		
rwo	r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r		rw/r	rw/r	rw/r	rw/r	rw/r	rw/r	rw/r

**Bit 31 COMP2LOCK:** Comparator 2 lock

This bit is write-once. It is set by software. It can only be cleared by a system reset.

It allows to have all control bits of comparator 2 as read-only.

0: COMP\_CSR[31:16] bits are read-write.

1: COMP\_CSR[31:16] bits are read-only.

**Bit 30 COMP2OUT:** Comparator 2 output

This read-only bit is a copy of comparator 2 output state.

0: Output is low (non-inverting input below inverting input).

1: Output is high (non-inverting input above inverting input).

**Bits 29:28 COMP2HYST[1:0]:** Comparator 2 hysteresis

These bits control the hysteresis level.

00: No hysteresis

01: Low hysteresis

10: Medium hysteresis

11: High hysteresis

Please refer to the electrical characteristics for the hysteresis values.

**Bit 27 COMP2POL:** Comparator 2 output polarity

This bit is used to invert the comparator 2 output.

0: Output is not inverted

1: Output is inverted

**Bits 26:24 COMP2OUTSEL[2:0]:** Comparator 2 output selection

These bits select the destination of the comparator output.

000: No selection

001: Timer 1 break input

010: Timer 1 Input capture 1

011: Timer 1 OCrefclear input

100: Timer 2 input capture 4

101: Timer 2 OCrefclear input

110: Timer 3 input capture 1

111: Timer 3 OCrefclear input

**Bit 23 WNDWEN:** Window mode enable

This bit connects the non-inverting input of COMP2 to COMP1's non-inverting input, which is simultaneously disconnected from PA3.

0: Window mode disabled

1: Window mode enabled

**Bits 22:20 COMP2INSEL[2:0]:** Comparator 2 inverting input selection

These bits allows to select the source connected to the inverting input of the comparator 2.

000: 1/4 of V<sub>REFINT</sub>

001: 1/2 of V<sub>REFINT</sub>

010: 3/4 of V<sub>REFINT</sub>

011: V<sub>REFINT</sub>

100: COMP2\_INM4 (PA4 with DAC\_OUT1 if enabled)

101: COMP2\_INM5 (PA5 with DAC\_OUT2 if present and enabled)

110: COMP2\_INM6 (PA2)

111: Reserved

Bits 19:18 **COMP2MODE[1:0]**: Comparator 2 mode

These bits control the operating mode of the comparator 2 and allows to adjust the speed/consumption.

- 00: High speed / full power
- 01: Medium speed / medium power
- 10: Low speed / low-power
- 11: Very-low speed / ultra-low power

Bit 17 Reserved, must be kept at reset value.

Bit 16 **COMP2EN**: Comparator 2 enable

This bit switches ON/OFF the comparator2.

- 0: Comparator 2 disabled
- 1: Comparator 2 enabled

Bit 15 **COMP1LOCK**: Comparator 1 lock

This bit is write-once. It is set by software. It can only be cleared by a system reset.

It allows to have all control bits of comparator 1 as read-only.

- 0: COMP\_CSR[15:0] bits are read-write.
- 1: COMP\_CSR[15:0] bits are read-only.

Bit 14 **COMP1OUT**: Comparator 1 output

This read-only bit is a copy of comparator 1 output state.

- 0: Output is low (non-inverting input below inverting input).
- 1: Output is high (non-inverting input above inverting input).

Bits 13:12 **COMP1HYST[1:0]** Comparator 1 hysteresis

These bits are controlling the hysteresis level.

- 00: No hysteresis
- 01: Low hysteresis
- 10: Medium hysteresis
- 11: High hysteresis

Please refer to the electrical characteristics for the hysteresis values.

Bit 11 **COMP1POL**: Comparator 1 output polarity

This bit is used to invert the comparator 1 output.

- 0: output is not inverted
- 1: output is inverted

Bits 10:8 **COMP1OUTSEL[2:0]**: Comparator 1 output selection

These bits selects the destination of the comparator 1 output.

- 000: no selection
- 001: Timer 1 break input
- 010: Timer 1 Input capture 1
- 011: Timer 1 OCrefclear input
- 100: Timer 2 input capture 4
- 101: Timer 2 OCrefclear input
- 110: Timer 3 input capture 1
- 111: Timer 3 OCrefclear input

Bit 7 Reserved, must be kept at reset value.

**Bits 6:4 COMP1INSEL[2:0]: Comparator 1 inverting input selection**

These bits select the source connected to the inverting input of the comparator 1.

- 000: 1/4 of  $V_{REFINT}$
- 001: 1/2 of  $V_{REFINT}$
- 010: 3/4 of  $V_{REFINT}$
- 011:  $V_{REFINT}$
- 100: COMP1\_INM4 (PA4 with DAC\_OUT1 if enabled)
- 101: COMP1\_INM5 (PA5 with DAC\_OUT2 if present and enabled)
- 110: COMP1\_INM6 (PA0)
- 111: Reserved

**Bits 3:2 COMP1MODE[1:0]: Comparator 1 mode**

These bits control the operating mode of the comparator 1 and allows to adjust the speed/consumption.

- 00: High speed / full power
- 01: Medium speed / medium power
- 10: Low speed / low-power
- 11: Very-low speed / ultra-low power

**Bit 1 COMP1SW1: Comparator 1 non inverting input DAC switch**

This bit closes a switch between comparator 1 non-inverting input on PA1 and PA4 (DAC) I/O.

- 0: Switch open
- 1: Switch closed

*Note: This switch is solely intended to redirect signals onto high impedance input, such as COMP1 non-inverting input (highly resistive switch).*

**Bit 0 COMP1EN: Comparator 1 enable**

This bit switches COMP1 ON/OFF.

- 0: Comparator 1 disabled
- 1: Comparator 1 enabled

## 15.5.2 COMP register map

The following table summarizes the comparator registers.

**Table 54. COMP register map and reset values**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x1C	<b>COMP_CSR</b>	COMP2LOCK	COMP2OUT	COMP2HYST[1:0]	COMP2POL			COMP2OUTSEL[2:0]		WNDWEN		COMP2INSEL[2:0]		COMP2MODE[1:0]		Res.	COMP2EN	COMP1LOCK	COMP1OUT		COMP1HYST[1:0]	COMP1POL		COMP1OUTSEL[2:0]		Res.	COMP1INSEL[2:0]		COMP1MODE[1:0]		COMP1ISW1	COMP1EN	0
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 16 Touch sensing controller (TSC)

This section applies to STM32F05x, STM32F04x, STM32F07x and STM32F09x devices only.

### 16.1 Introduction

The touch sensing controller provides a simple solution for adding capacitive sensing functionality to any application. Capacitive sensing technology is able to detect finger presence near an electrode that is protected from direct touch by a dielectric (for example glass, plastic). The capacitive variation introduced by the finger (or any conductive object) is measured using a proven implementation based on a surface charge transfer acquisition principle.

The touch sensing controller is fully supported by the STMTouch touch sensing firmware library, which is free to use and allows touch sensing functionality to be implemented reliably in the end application.

### 16.2 TSC main features

The touch sensing controller has the following main features:

- Proven and robust surface charge transfer acquisition principle
- Supports up to 24 capacitive sensing channels
- Up to 8 capacitive sensing channels can be acquired in parallel offering a very good response time
- Spread spectrum feature to improve system robustness in noisy environments
- Full hardware management of the charge transfer acquisition sequence
- Programmable charge transfer frequency
- Programmable sampling capacitor I/O pin
- Programmable channel I/O pin
- Programmable max count value to avoid long acquisition when a channel is faulty
- Dedicated end of acquisition and max count error flags with interrupt capability
- One sampling capacitor for up to 3 capacitive sensing channels to reduce the system components
- Compatible with proximity, touchkey, linear and rotary touch sensor implementation
- Designed to operate with STMTouch touch sensing firmware library

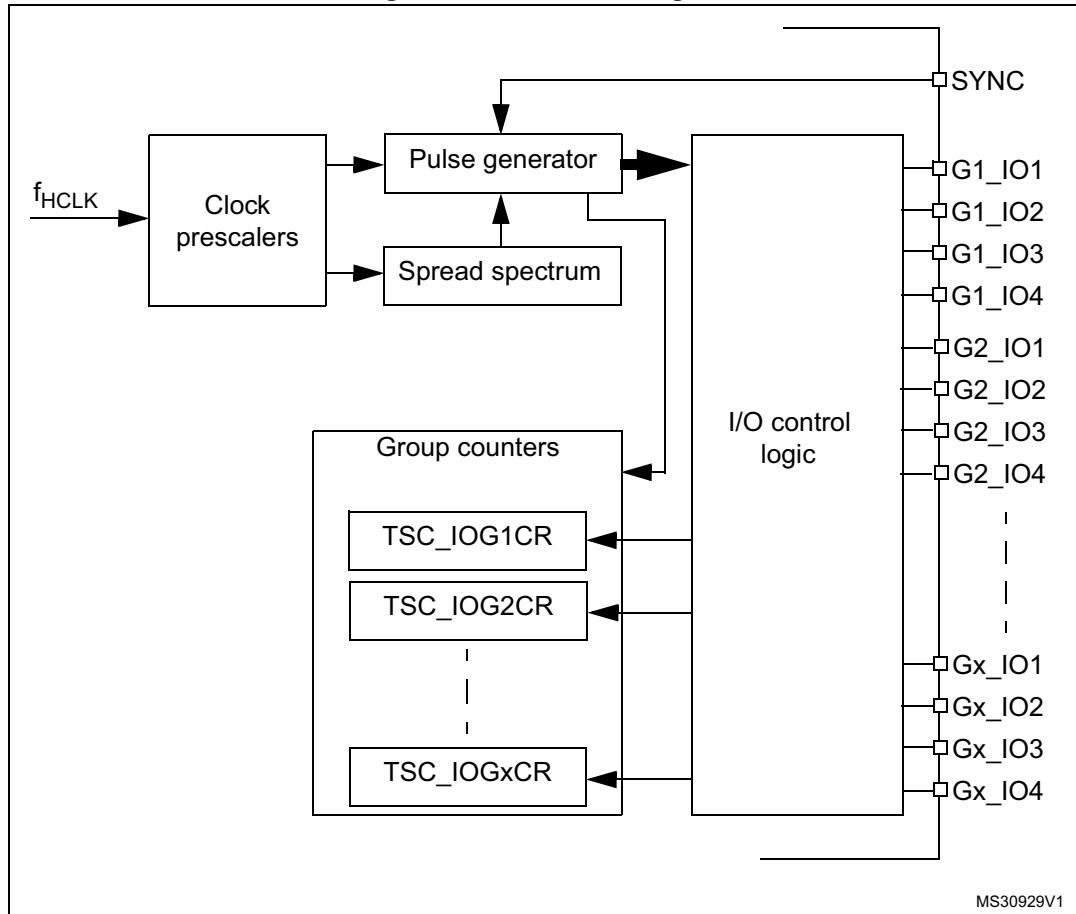
*Note:* The number of capacitive sensing channels is dependent on the size of the packages and subject to IO availability.

## 16.3 TSC functional description

### 16.3.1 TSC block diagram

The block diagram of the touch sensing controller is shown in [Figure 58](#).

**Figure 58. TSC block diagram**



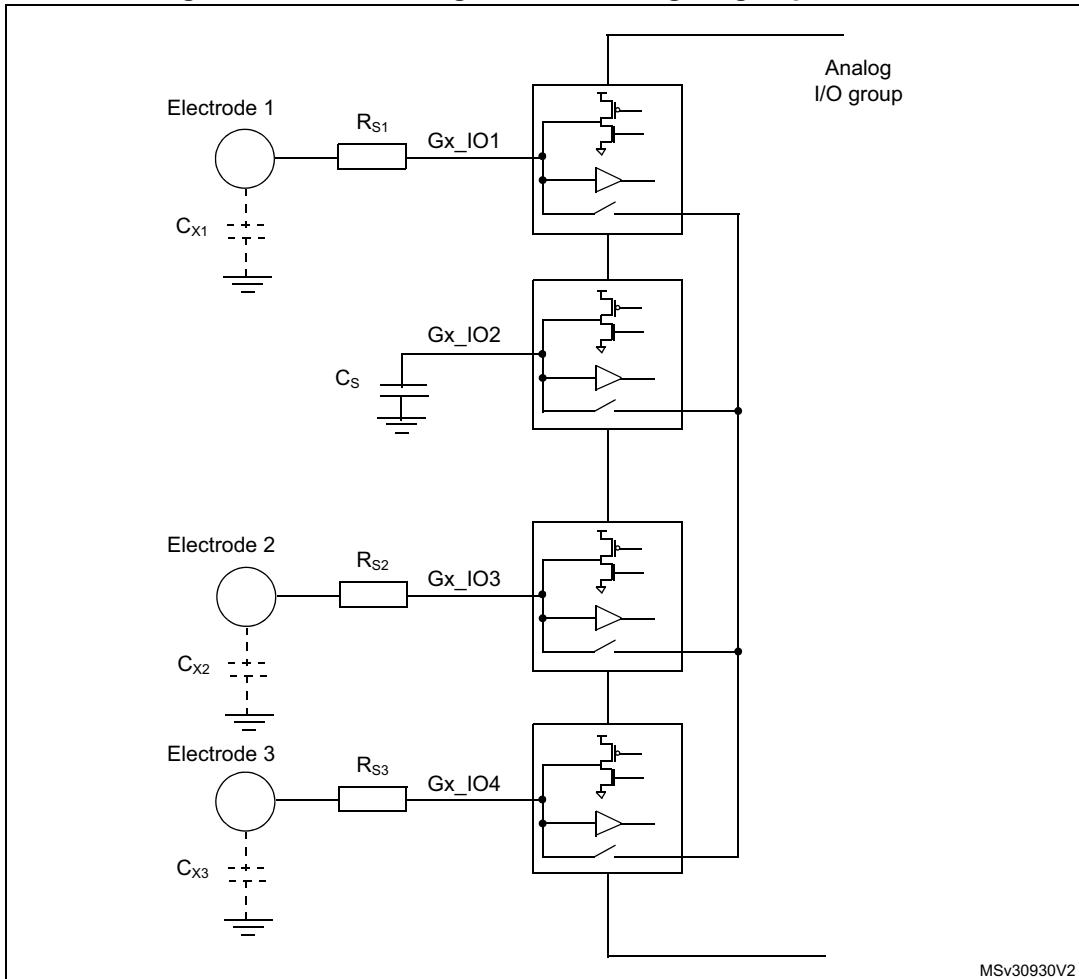
### 16.3.2 Surface charge transfer acquisition overview

The surface charge transfer acquisition is a proven, robust and efficient way to measure a capacitance. It uses a minimum number of external components to operate with a single ended electrode type. This acquisition is designed around an analog I/O group composed of up to four GPIOs (see [Figure 59](#)). Several analog I/O groups are available to allow the acquisition of several capacitive sensing channels simultaneously and to support a larger number of capacitive sensing channels. Within a same analog I/O group, the acquisition of the capacitive sensing channels is sequential.

One of the GPIOs is dedicated to the sampling capacitor  $C_S$ . Only one sampling capacitor I/O per analog I/O group must be enabled at a time.

The remaining GPIOs are dedicated to the electrodes and are commonly called channels. For some specific needs (such as proximity detection), it is possible to simultaneously enable more than one channel per analog I/O group.

Figure 59. Surface charge transfer analog I/O group structure



Note: *Gx<sub>x</sub>\_IOy where x is the analog I/O group number and y the GPIO number within the selected group.*

The surface charge transfer acquisition principle consists of charging an electrode capacitance ( $C_X$ ) and transferring a part of the accumulated charge into a sampling capacitor ( $C_S$ ). This sequence is repeated until the voltage across  $C_S$  reaches a given threshold ( $V_{IH}$  in our case). The number of charge transfers required to reach the threshold is a direct representation of the size of the electrode capacitance.

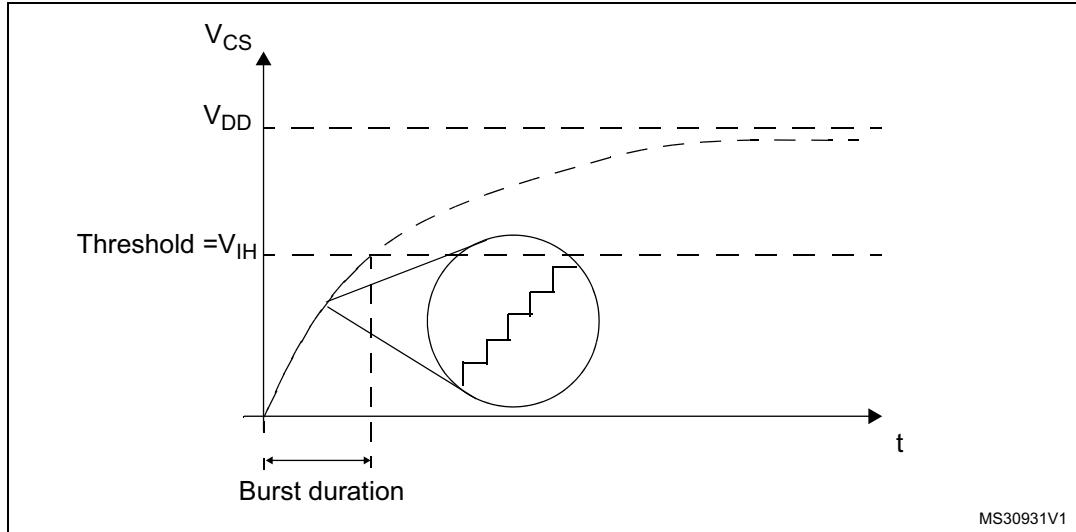
[Table 55](#) details the charge transfer acquisition sequence of the capacitive sensing channel 1. States 3 to 7 are repeated until the voltage across  $C_S$  reaches the given threshold. The same sequence applies to the acquisition of the other channels. The electrode serial resistor  $R_S$  improves the ESD immunity of the solution.

**Table 55. Acquisition sequence summary**

State	Gx_IO1 (channel)	Gx_IO2 (sampling)	Gx_IO3 (channel)	Gx_IO4 (channel)	State description		
#1	Input floating with analog switch closed	Output open-drain low with analog switch closed	Input floating with analog switch closed		Discharge all $C_X$ and $C_S$		
#2	Input floating				Dead time		
#3	Output push-pull high	Input floating			Charge $C_{X1}$		
#4	Input floating				Dead time		
#5	Input floating with analog switch closed	Input floating			Charge transfer from $C_{X1}$ to $C_S$		
#6	Input floating				Dead time		
#7	Input floating				Measure $C_S$ voltage		

Note:  $Gx\_IOy$  where  $x$  is the analog I/O group number and  $y$  the GPIO number within the selected group.

The voltage variation over the time on the sampling capacitor  $C_S$  is detailed below:

**Figure 60. Sampling capacitor voltage variation**

### 16.3.3 Reset and clocks

The TSC clock source is the AHB clock (HCLK). Two programmable prescalers are used to generate the pulse generator and the spread spectrum internal clocks:

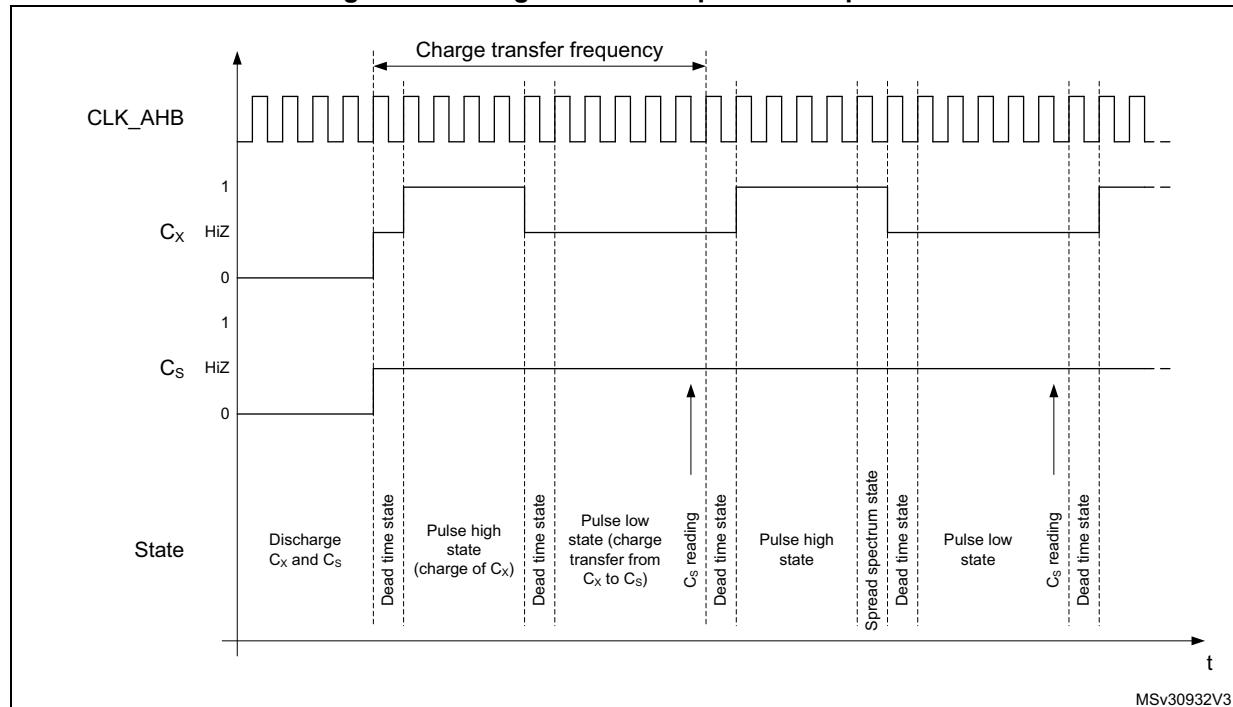
- The pulse generator clock (PGCLK) is defined using the PG\_PSC[2:0] bits of the TSC\_CR register
- The spread spectrum clock (SSCLK) is defined using the SSPSC bit of the TSC\_CR register

The Reset and Clock Controller (RCC) provides dedicated bits to enable the touch sensing controller clock and to reset this peripheral. For more information, refer to [Section 6: Reset and clock control \(RCC\)](#).

### 16.3.4 Charge transfer acquisition sequence

An example of a charge transfer acquisition sequence is detailed in [Figure 61](#).

**Figure 61. Charge transfer acquisition sequence**



MSv30932V3

For higher flexibility, the charge transfer frequency is fully configurable. Both the pulse high state (charge of  $C_X$ ) and the pulse low state (transfer of charge from  $C_X$  to  $C_S$ ) duration can be defined using the CTPH[3:0] and CTPL[3:0] bits in the TSC\_CR register. The standard range for the pulse high and low states duration is 500 ns to 2  $\mu$ s. To ensure a correct measurement of the electrode capacitance, the pulse high state duration must be set to ensure that  $C_X$  is always fully charged.

A dead time where both the sampling capacitor I/O and the channel I/O are in input floating state is inserted between the pulse high and low states to ensure an optimum charge transfer acquisition sequence. This state duration is 2 periods of HCLK.

At the end of the pulse high state and if the spread spectrum feature is enabled, a variable number of periods of the SSCLK clock are added.

The reading of the sampling capacitor I/O, to determine if the voltage across  $C_S$  has reached the given threshold, is performed at the end of the pulse low state.

**Note:**

*The following TSC control register configurations are forbidden:*

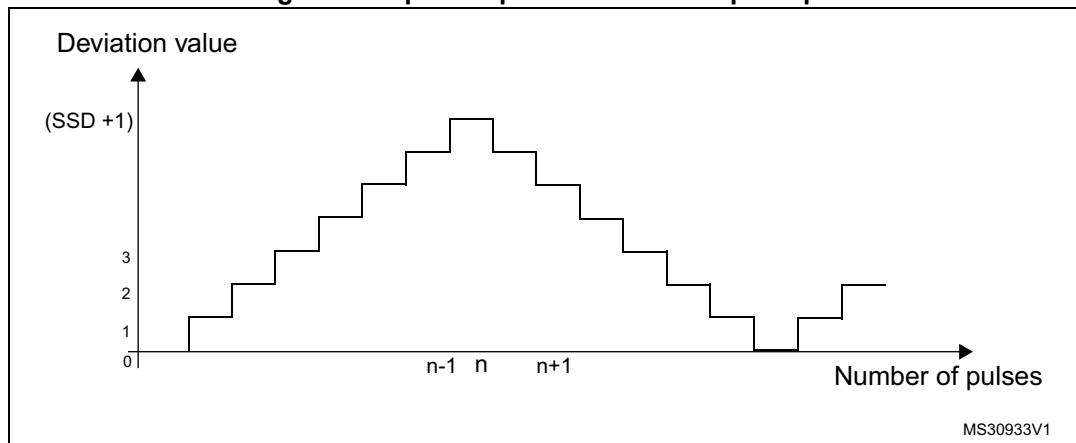
- bits PGPSC are set to '000' and bits CTPL are set to '0000'
- bits PGPSC are set to '000' and bits CTPL are set to '0001'
- bits PGPSC are set to '001' and bits CTPL are set to '0000'

### 16.3.5 Spread spectrum feature

The spread spectrum feature generates a variation of the charge transfer frequency. This is done to improve the robustness of the charge transfer acquisition in noisy environments and also to reduce the induced emission. The maximum frequency variation is in the range of 10% to 50% of the nominal charge transfer period. For instance, for a nominal charge transfer frequency of 250 kHz (4  $\mu$ s), the typical spread spectrum deviation is 10% (400 ns) which leads to a minimum charge transfer frequency of ~227 kHz.

In practice, the spread spectrum consists of adding a variable number of SSCLK periods to the pulse high state using the principle shown below:

**Figure 62. Spread spectrum variation principle**



The table below details the maximum frequency deviation with different HCLK settings:

**Table 56. Spread spectrum deviation versus AHB clock frequency**

$f_{HCLK}$	Spread spectrum step	Maximum spread spectrum deviation
24 MHz	41.6 ns	10666.6 ns
48 MHz	20.8 ns	5333.3 ns

The spread spectrum feature can be disabled/enabled using the SSE bit in the TSC\_CR register. The frequency deviation is also configurable to accommodate the device HCLK clock frequency and the selected charge transfer frequency through the SSPSC and SSD[6:0] bits in the TSC\_CR register.

### 16.3.6 Max count error

The max count error prevents long acquisition times resulting from a faulty capacitive sensing channel. It consists of specifying a maximum count value for the analog I/O group counters. This maximum count value is specified using the MCV[2:0] bits in the TSC\_CR register. As soon as an acquisition group counter reaches this maximum value, the ongoing acquisition is stopped and the end of acquisition (EOAF bit) and max count error (MCEF bit) flags are both set. An interrupt can also be generated if the corresponding end of acquisition (EOAIE bit) or/and max count error (MCEIE bit) interrupt enable bits are set.

### 16.3.7 Sampling capacitor I/O and channel I/O mode selection

To allow the GPIOs to be controlled by the touch sensing controller, the corresponding alternate function must be enabled through the standard GPIO registers and the GPIOxAFR registers.

The GPIOs modes controlled by the TSC are defined using the TSC\_IOSCR and TSC\_IOCCR register.

When there is no ongoing acquisition, all the I/Os controlled by the touch sensing controller are in default state. While an acquisition is ongoing, only unused I/Os (neither defined as sampling capacitor I/O nor as channel I/O) are in default state. The IODEF bit in the TSC\_CR register defines the configuration of the I/Os which are in default state. The table below summarizes the configuration of the I/O depending on its mode.

**Table 57. I/O state depending on its mode and IODEF bit value**

IODEF bit	Acquisition status	Unused I/O mode	Channel I/O mode	Sampling capacitor I/O mode
0 (output push-pull low)	No	Output push-pull low	Output push-pull low	Output push-pull low
0 (output push-pull low)	Ongoing	Output push-pull low	-	-
1 (input floating)	No	Input floating	Input floating	Input floating
1 (input floating)	Ongoing	Input floating	-	-

#### Unused I/O mode

An unused I/O corresponds to a GPIO controlled by the TSC peripheral but not defined as an electrode I/O nor as a sampling capacitor I/O.

#### Sampling capacitor I/O mode

To allow the control of the sampling capacitor I/O by the TSC peripheral, the corresponding GPIO must be first set to alternate output open drain mode and then the corresponding Gx\_IOy bit in the TSC\_IOSCR register must be set.

Only one sampling capacitor per analog I/O group must be enabled at a time.

#### Channel I/O mode

To allow the control of the channel I/O by the TSC peripheral, the corresponding GPIO must be first set to alternate output push-pull mode and the corresponding Gx\_IOy bit in the TSC\_IOCCR register must be set.

For proximity detection where a higher equivalent electrode surface is required or to speed-up the acquisition process, it is possible to enable and simultaneously acquire several channels belonging to the same analog I/O group.

#### Note:

*During the acquisition phase and even if the TSC peripheral alternate function is not enabled, as soon as the TSC\_IOSCR or TSC\_IOCCR bit is set, the corresponding GPIO analog switch is automatically controlled by the touch sensing controller.*

### 16.3.8 Acquisition mode

The touch sensing controller offers two acquisition modes:

- Normal acquisition mode: the acquisition starts as soon as the START bit in the TSC\_CR register is set.
- Synchronized acquisition mode: the acquisition is enabled by setting the START bit in the TSC\_CR register but only starts upon the detection of a falling edge or a rising edge and high level on the SYNC input pin. This mode is useful for synchronizing the capacitive sensing channels acquisition with an external signal without additional CPU load.

The GxE bits in the TSC\_IOGCSR registers specify which analog I/O groups are enabled (corresponding counter is counting). The CS voltage of a disabled analog I/O group is not monitored and this group does not participate in the triggering of the end of acquisition flag. However, if the disabled analog I/O group contains some channels, they are pulsed.

When the CS voltage of an enabled analog I/O group reaches the given threshold, the corresponding GxS bit of the TSC\_IOGCSR register is set. When the acquisition of all enabled analog I/O groups is complete (all GxS bits of all enabled analog I/O groups are set), the EOAF flag in the TSC\_ISR register is set. An interrupt request is generated if the EOAIE bit in the TSC\_IER register is set.

In the case that a max count error is detected, the ongoing acquisition is stopped and both the EOAF and MCEF flags in the TSC\_ISR register are set. Interrupt requests can be generated for both events if the corresponding bits (EOAIE and MCEIE bits of the TSCIER register) are set. Note that when the max count error is detected the remaining GxS bits in the enabled analog I/O groups are not set.

To clear the interrupt flags, the corresponding EOAC and MCEIC bits in the TSC\_ICR register must be set.

The analog I/O group counters are cleared when a new acquisition is started. They are updated with the number of charge transfer cycles generated on the corresponding channel(s) upon the completion of the acquisition.

For code example refer to the Appendix section [A.18.1: TSC configuration code example](#).

### 16.3.9 I/O hysteresis and analog switch control

In order to offer a higher flexibility, the touch sensing controller is able to take the control of the Schmitt trigger hysteresis and analog switch of each Gx\_IOy. This control is available whatever the I/O control mode is (controlled by standard GPIO registers or other peripherals) assuming that the touch sensing controller is enabled. This may be useful to perform a different acquisition sequence or for other purposes.

In order to improve the system immunity, the Schmitt trigger hysteresis of the GPIOs controlled by the TSC must be disabled by resetting the corresponding Gx\_IOy bit in the TSC\_IOHCR register.

## 16.4 TSC low-power modes

**Table 58. Effect of low-power modes on TSC**

Mode	Description
Sleep	No effect TSC interrupts cause the device to exit Sleep mode.
Stop	TSC registers are frozen
Standby	The TSC stops its operation until the Stop or Standby mode is exited.

## 16.5 TSC interrupts

**Table 59. Interrupt control bits**

Interrupt event	Enable control bit	Event flag	Clear flag bit	Exit the Sleep mode	Exit the Stop mode	Exit the Standby mode
End of acquisition	EOAIE	EOAIF	EOAIC	Yes	No	No
Max count error	MCEIE	MCEIF	MCEIC	Yes	No	No

For code example refer to the Appendix section [A.18.2: TSC interrupt code example](#).

## 16.6 TSC registers

Refer to [Section 1.2 on page 42](#) of the reference manual for a list of abbreviations used in register descriptions.

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

### 16.6.1 TSC control register (TSC\_CR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16		
CTPH[3:0]				CTPL[3:0]				SSD[6:0]								SSE	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
SSPSC	PGPSC[2:0]				Res.	Res.	Res.	Res.	MCV[2:0]				IODEF	SYNC POL	AM	START	TSCE
rw	rw	rw	rw					rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bits 31:28 **CTPH[3:0]**: Charge transfer pulse high

These bits are set and cleared by software. They define the duration of the high state of the charge transfer pulse (charge of  $C_X$ ).

0000: 1x  $t_{PGCLK}$   
0001: 2x  $t_{PGCLK}$   
...

1111: 16x  $t_{PGCLK}$

*Note: These bits must not be modified when an acquisition is ongoing.*

Bits 27:24 **CTPL[3:0]**: Charge transfer pulse low

These bits are set and cleared by software. They define the duration of the low state of the charge transfer pulse (transfer of charge from  $C_X$  to  $C_S$ ).

0000: 1x  $t_{PGCLK}$   
0001: 2x  $t_{PGCLK}$   
...

1111: 16x  $t_{PGCLK}$

*Note: These bits must not be modified when an acquisition is ongoing.*

*Note: Some configurations are forbidden. Refer to the [Section 16.3.4: Charge transfer acquisition sequence](#) for details.*

Bits 23:17 **SSD[6:0]**: Spread spectrum deviation

These bits are set and cleared by software. They define the spread spectrum deviation which consists in adding a variable number of periods of the SSCLK clock to the charge transfer pulse high state.

0000000: 1x  $t_{SSCLK}$   
0000001: 2x  $t_{SSCLK}$   
...

1111111: 128x  $t_{SSCLK}$

*Note: These bits must not be modified when an acquisition is ongoing.*

Bit 16 **SSE**: Spread spectrum enable

This bit is set and cleared by software to enable/disable the spread spectrum feature.

0: Spread spectrum disabled

1: Spread spectrum enabled

*Note: This bit must not be modified when an acquisition is ongoing.*

Bit 15 **SSPSC**: Spread spectrum prescaler

This bit is set and cleared by software. It selects the AHB clock divider used to generate the spread spectrum clock (SSCLK).

0:  $f_{HCLK}$

1:  $f_{HCLK}/2$

*Note: This bit must not be modified when an acquisition is ongoing.*

Bits 14:12 **PGPSC[2:0]**: Pulse generator prescaler

These bits are set and cleared by software. They select the AHB clock divider used to generate the pulse generator clock (PGCLK).

000:  $f_{HCLK}$

001:  $f_{HCLK}/2$

010:  $f_{HCLK}/4$

011:  $f_{HCLK}/8$

100:  $f_{HCLK}/16$

101:  $f_{HCLK}/32$

110:  $f_{HCLK}/64$

111:  $f_{HCLK}/128$

*Note: These bits must not be modified when an acquisition is ongoing.*

*Note: Some configurations are forbidden. Refer to the [Section 16.3.4: Charge transfer acquisition sequence](#) for details.*

Bits 11:8 Reserved, must be kept at reset value.

Bits 7:5 **MCV[2:0]**: Max count value

These bits are set and cleared by software. They define the maximum number of charge transfer pulses that can be generated before a max count error is generated.

000: 255

001: 511

010: 1023

011: 2047

100: 4095

101: 8191

110: 16383

111: reserved

*Note: These bits must not be modified when an acquisition is ongoing.*

Bit 4 **IODEF**: I/O Default mode

This bit is set and cleared by software. It defines the configuration of all the TSC I/Os when there is no ongoing acquisition. When there is an ongoing acquisition, it defines the configuration of all unused I/Os (not defined as sampling capacitor I/O or as channel I/O).

0: I/Os are forced to output push-pull low

1: I/Os are in input floating

*Note: This bit must not be modified when an acquisition is ongoing.*

Bit 3 **SYNCPOL**: Synchronization pin polarity

This bit is set and cleared by software to select the polarity of the synchronization input pin.

0: Falling edge only

1: Rising edge and high level

Bit 2 **AM**: Acquisition mode

This bit is set and cleared by software to select the acquisition mode.

0: Normal acquisition mode (acquisition starts as soon as START bit is set)

1: Synchronized acquisition mode (acquisition starts if START bit is set and when the selected signal is detected on the SYNC input pin)

*Note: This bit must not be modified when an acquisition is ongoing.*

Bit 1 **START**: Start a new acquisition

This bit is set by software to start a new acquisition. It is cleared by hardware as soon as the acquisition is complete or by software to cancel the ongoing acquisition.

0: Acquisition not started

1: Start a new acquisition

Bit 0 **TSC\_E**: Touch sensing controller enable

This bit is set and cleared by software to enable/disable the touch sensing controller.

0: Touch sensing controller disabled

1: Touch sensing controller enabled

*Note: When the touch sensing controller is disabled, TSC registers settings have no effect.*

**16.6.2 TSC interrupt enable register (TSC\_IER)**

Address offset: 0x04

Reset value: 0x0000 0000

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

Bits 31:2 Reserved, must be kept at reset value.

Bit 1 **MCEIE**: Max count error interrupt enable

This bit is set and cleared by software to enable/disable the max count error interrupt.

0: Max count error interrupt disabled

1: Max count error interrupt enabled

Bit 0 **EOAIE**: End of acquisition interrupt enable

This bit is set and cleared by software to enable/disable the end of acquisition interrupt.

0: End of acquisition interrupt disabled

1: End of acquisition interrupt enabled

### 16.6.3 TSC interrupt clear register (TSC\_ICR)

Address offset: 0x08

Reset value: 0x0000 0000

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

Bits 31:2 Reserved, must be kept at reset value.

Bit 1 **MCEIC**: Max count error interrupt clear

This bit is set by software to clear the max count error flag and it is cleared by hardware when the flag is reset. Writing a '0' has no effect.

0: No effect

1: Clears the corresponding MCEF of the TSC\_ISR register

Bit 0 **EOAIC**: End of acquisition interrupt clear

This bit is set by software to clear the end of acquisition flag and it is cleared by hardware when the flag is reset. Writing a '0' has no effect.

0: No effect

1: Clears the corresponding EOAF of the TSC\_ISR register

### 16.6.4 TSC interrupt status register (TSC\_ISR)

Address offset: 0x0C

Reset value: 0x0000 0000

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

Bits 31:2 Reserved, must be kept at reset value.

Bit 1 **MCEF**: Max count error flag

This bit is set by hardware as soon as an analog I/O group counter reaches the max count value specified. It is cleared by software writing 1 to the bit MCEIC of the TSC\_ICR register.

0: No max count error (MCE) detected

1: Max count error (MCE) detected

Bit 0 **EOAF**: End of acquisition flag

This bit is set by hardware when the acquisition of all enabled group is complete (all GxS bits of all enabled analog I/O groups are set or when a max count error is detected). It is cleared by software writing 1 to the bit EOAC of the TSC\_ICR register.

0: Acquisition is ongoing or not started

1: Acquisition is complete

### 16.6.5 TSC I/O hysteresis control register (TSC\_Iohcr)

Address offset: 0x10

Reset value: 0xFFFF FFFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
G8_IO4	G8_IO3	G8_IO2	G8_IO1	G7_IO4	G7_IO3	G7_IO2	G7_IO1	G6_IO4	G6_IO3	G6_IO2	G6_IO1	G5_IO4	G5_IO3	G5_IO2	G5_IO1
rw															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
G4_IO4	G4_IO3	G4_IO2	G4_IO1	G3_IO4	G3_IO3	G3_IO2	G3_IO1	G2_IO4	G2_IO3	G2_IO2	G2_IO1	G1_IO4	G1_IO3	G1_IO2	G1_IO1
rw															

Bits 31:0 **Gx\_IOy**: Gx\_IOy Schmitt trigger hysteresis mode, x = 8 to 1, y = 4 to 1.

These bits are set and cleared by software to enable/disable the Gx\_IOy Schmitt trigger hysteresis.

0: Gx\_IOy Schmitt trigger hysteresis disabled

1: Gx\_IOy Schmitt trigger hysteresis enabled

*Note: These bits control the I/O Schmitt trigger hysteresis whatever the I/O control mode is (even if controlled by standard GPIO registers).*

### 16.6.6 TSC I/O analog switch control register (TSC\_IOASCR)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
G8_IO4	G8_IO3	G8_IO2	G8_IO1	G7_IO4	G7_IO3	G7_IO2	G7_IO1	G6_IO4	G6_IO3	G6_IO2	G6_IO1	G5_IO4	G5_IO3	G5_IO2	G5_IO1
rw															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
G4_IO4	G4_IO3	G4_IO2	G4_IO1	G3_IO4	G3_IO3	G3_IO2	G3_IO1	G2_IO4	G2_IO3	G2_IO2	G2_IO1	G1_IO4	G1_IO3	G1_IO2	G1_IO1
rw															

Bits 31:0 **Gx\_IOy**: Gx\_IOy analog switch enable

These bits are set and cleared by software to enable/disable the Gx\_IOy analog switch.

0: Gx\_IOy analog switch disabled (opened)

1: Gx\_IOy analog switch enabled (closed)

*Note: These bits control the I/O analog switch whatever the I/O control mode is (even if controlled by standard GPIO registers).*

### 16.6.7 TSC I/O sampling control register (TSC\_IOSCR)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
G8_IO4	G8_IO3	G8_IO2	G8_IO1	G7_IO4	G7_IO3	G7_IO2	G7_IO1	G6_IO4	G6_IO3	G6_IO2	G6_IO1	G5_IO4	G5_IO3	G5_IO2	G5_IO1
rw															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
G4_IO4	G4_IO3	G4_IO2	G4_IO1	G3_IO4	G3_IO3	G3_IO2	G3_IO1	G2_IO4	G2_IO3	G2_IO2	G2_IO1	G1_IO4	G1_IO3	G1_IO2	G1_IO1
rw															

Bits 31:0 **Gx\_IOy**: Gx\_IOy sampling mode

These bits are set and cleared by software to configure the Gx\_IOy as a sampling capacitor I/O. Only one I/O per analog I/O group must be defined as sampling capacitor.

0: Gx\_IOy unused

1: Gx\_IOy used as sampling capacitor

*Note: These bits must not be modified when an acquisition is ongoing.*

*During the acquisition phase and even if the TSC peripheral alternate function is not enabled, as soon as the TSC\_IOSCR bit is set, the corresponding GPIO analog switch is automatically controlled by the touch sensing controller.*

### 16.6.8 TSC I/O channel control register (TSC\_IOCCR)

Address offset: 0x28

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
G8_IO4	G8_IO3	G8_IO2	G8_IO1	G7_IO4	G7_IO3	G7_IO2	G7_IO1	G6_IO4	G6_IO3	G6_IO2	G6_IO1	G5_IO4	G5_IO3	G5_IO2	G5_IO1
rw															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
G4_IO4	G4_IO3	G4_IO2	G4_IO1	G3_IO4	G3_IO3	G3_IO2	G3_IO1	G2_IO4	G2_IO3	G2_IO2	G2_IO1	G1_IO4	G1_IO3	G1_IO2	G1_IO1
rw															

Bits 31:0 **Gx\_IoY**: Gx\_IoY channel mode

These bits are set and cleared by software to configure the Gx\_IoY as a channel I/O.

0: Gx\_IoY unused

1: Gx\_IoY used as channel

*Note: These bits must not be modified when an acquisition is ongoing.*

*During the acquisition phase and even if the TSC peripheral alternate function is not enabled, as soon as the TSC\_IOCCR bit is set, the corresponding GPIO analog switch is automatically controlled by the touch sensing controller.*

### 16.6.9 TSC I/O group control status register (TSC\_IOGCSR)

Address offset: 0x30

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	G8S	G7S	G6S	G5S	G4S	G3S	G2S	G1S							
								r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	G8E	G7E	G6E	G5E	G4E	G3E	G2E	G1E							
								rw							

Bits 31:24 Reserved, must be kept at reset value.

Bits 23:16 **GxS**: Analog I/O group x status

These bits are set by hardware when the acquisition on the corresponding enabled analog I/O group x is complete. They are cleared by hardware when a new acquisition is started.

0: Acquisition on analog I/O group x is ongoing or not started

1: Acquisition on analog I/O group x is complete

*Note: When a max count error is detected the remaining GxS bits of the enabled analog I/O groups are not set.*

Bits 15:8 Reserved, must be kept at reset value.

Bits 7:0 **GxE**: Analog I/O group x enable

These bits are set and cleared by software to enable/disable the acquisition (counter is counting) on the corresponding analog I/O group x.

0: Acquisition on analog I/O group x disabled

1: Acquisition on analog I/O group x enabled

### 16.6.10 TSC I/O group x counter register (TSC\_IOGxCR)

x represents the analog I/O group number.

Address offset: 0x30 + 0x04 \* x, (x = 1 to 8)

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	CNT[13:0]													
		r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:14 Reserved, must be kept at reset value.

Bits 13:0 **CNT[13:0]**: Counter value

These bits represent the number of charge transfer cycles generated on the analog I/O group x to complete its acquisition (voltage across  $C_S$  has reached the threshold).

### 16.6.11 TSC register map

Table 60. TSC register map and reset values

Offset	Register								
0x0000	<b>TSC_CR</b>	CTPH[3:0]				CTPL[3:0]			
		Reset value	0	0	0	0	0	0	Res.
0x0004	<b>TSC_IER</b>								
		Reset value	0	0	0	0	0	0	Res.
0x0008	<b>TSC_ICR</b>								
		Reset value	0	0	0	0	0	0	Res.
0x000C	<b>TSC_ISR</b>								
		Reset value	0	0	0	0	0	0	Res.
0x0010	<b>TSC_IOHCR</b>								
		Reset value	1	1	1	1	1	1	G8_I04
0x0014									
0x0018	<b>TSC_IOASCR</b>								
		Reset value	0	0	0	0	0	0	G8_I04
0x001C									
0x0020	<b>TSC_IOSCR</b>								
		Reset value	0	0	0	0	0	0	G8_I04
0x0024									
0x0028	<b>TSC_IOCGR</b>								
		Reset value	0	0	0	0	0	0	G8_I04
0x002C									
0x0030	<b>TSC_IOGCSR</b>								
		Reset value	0	0	0	0	0	0	G8_I04
0x0034	<b>TSC_IOG1CR</b>								
		Reset value	0	0	0	0	0	0	G8_I04
0x0038	<b>TSC_IOG2CR</b>								
		Reset value	0	0	0	0	0	0	G8_I04

**Table 60. TSC register map and reset values (continued)**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x003C	TSC_IOG3CR	Res.	0	0	0	0	0	0	0	0	0	0	0																				
	Reset value	Res.	0	0	0	0	0	0	0	0	0	0	0																				
0x0040	TSC_IOG4CR	Res.	0	0	0	0	0	0	0	0	0	0	0																				
	Reset value	Res.	0	0	0	0	0	0	0	0	0	0	0																				
0x0044	TSC_IOG5CR	Res.	0	0	0	0	0	0	0	0	0	0	0																				
	Reset value	Res.	0	0	0	0	0	0	0	0	0	0	0																				
0x0048	TSC_IOG6CR	Res.	0	0	0	0	0	0	0	0	0	0	0																				
	Reset value	Res.	0	0	0	0	0	0	0	0	0	0	0																				
0x004C	TSC_IOG7CR	Res.	0	0	0	0	0	0	0	0	0	0	0																				
	Reset value	Res.	0	0	0	0	0	0	0	0	0	0	0																				
0x0050	TSC_IOG8CR	Res.	0	0	0	0	0	0	0	0	0	0	0																				
	Reset value	Res.	0	0	0	0	0	0	0	0	0	0	0																				

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 17 Advanced-control timers (TIM1)

### 17.1 TIM1 introduction

The advanced-control timers (TIM1) consist of a 16-bit auto-reload counter driven by a programmable prescaler.

It may be used for a variety of purposes, including measuring the pulse lengths of input signals (input capture) or generating output waveforms (output compare, PWM, complementary PWM with dead-time insertion).

Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the RCC clock controller prescalers.

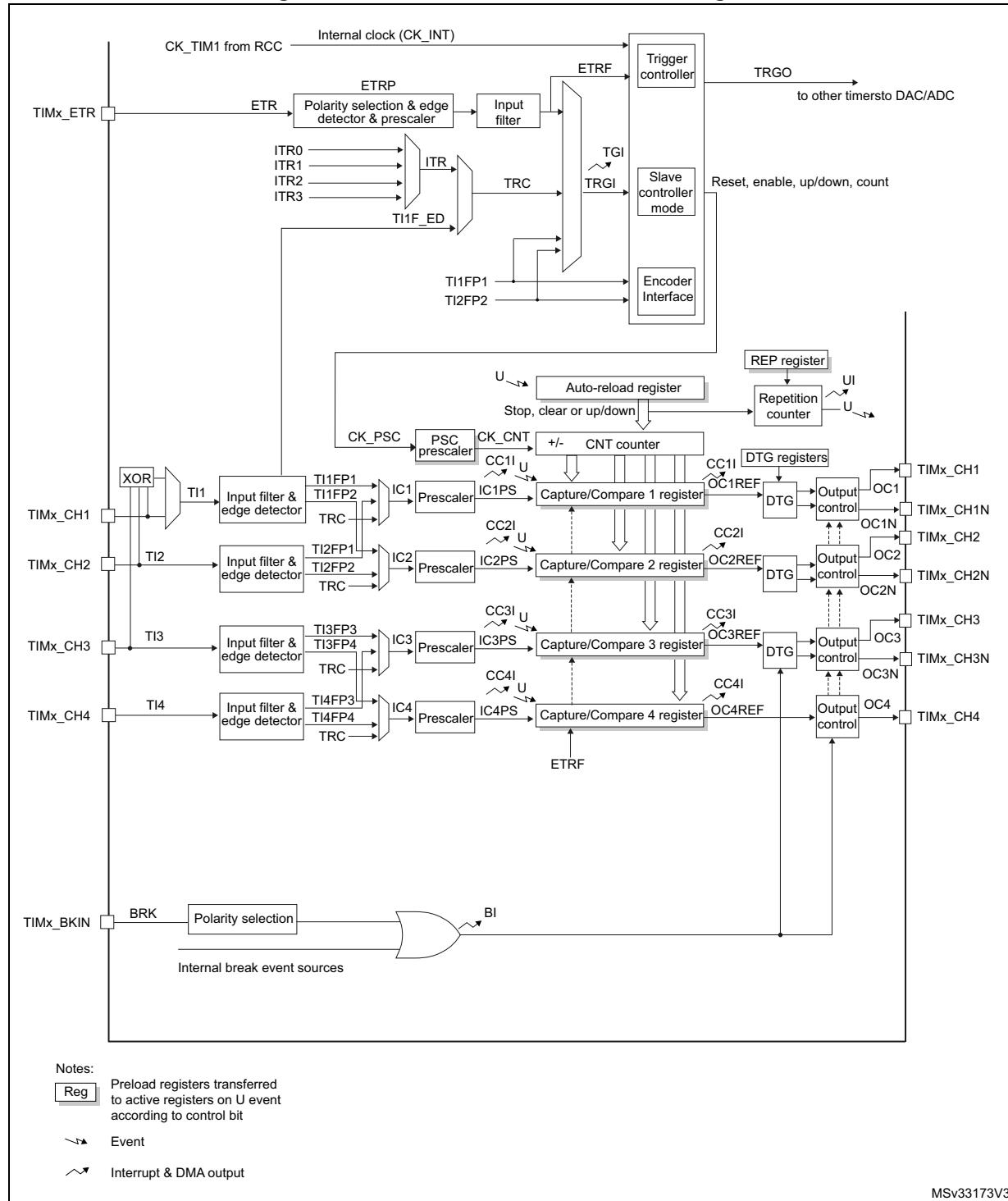
The advanced-control (TIM1) and general-purpose (TIMx) timers are completely independent, and do not share any resources. They can be synchronized together as described in [Section 17.3.20](#).

### 17.2 TIM1 main features

TIM1 timer features include:

- 16-bit up, down, up/down auto-reload counter.
- 16-bit programmable prescaler allowing dividing (also “on the fly”) the counter clock frequency either by any factor between 1 and 65535.
- Up to 4 independent channels for:
  - Input Capture
  - Output Compare
  - PWM generation (Edge- and Center-aligned modes)
  - One-pulse mode output
- Complementary outputs with programmable dead-time
- Synchronization circuit to control the timer with external signals and to interconnect several timers together.
- Repetition counter to update the timer registers only after a given number of cycles of the counter.
- Break input to put the timer’s output signals in reset state or in a known state.
- Interrupt/DMA generation on the following events:
  - Update: counter overflow/underflow, counter initialization (by software or internal/external trigger)
  - Trigger event (counter start, stop, initialization or count by internal/external trigger)
  - Input capture
  - Output compare
  - Break input
- Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes
- Trigger input for external clock or cycle-by-cycle current management

Figure 63. Advanced-control timer block diagram



MSv33173V3

## 17.3 TIM1 functional description

### 17.3.1 Time-base unit

The main block of the programmable advanced-control timer is a 16-bit counter with its related auto-reload register. The counter can count up, down or both up and down. The counter clock can be divided by a prescaler.

The counter, the auto-reload register and the prescaler register can be written or read by software. This is true even when the counter is running.

The time-base unit includes:

- Counter register (TIMx\_CNT)
- Prescaler register (TIMx\_PSC)
- Auto-reload register (TIMx\_ARR)
- Repetition counter register (TIMx\_RCR)

The auto-reload register is preloaded. Writing to or reading from the auto-reload register accesses the preload register. The content of the preload register are transferred into the shadow register permanently or at each update event (UEV), depending on the auto-reload preload enable bit (ARPE) in TIMx\_CR1 register. The update event is sent when the counter reaches the overflow (or underflow when downcounting) and if the UDIS bit equals 0 in the TIMx\_CR1 register. It can also be generated by software. The generation of the update event is described in detailed for each configuration.

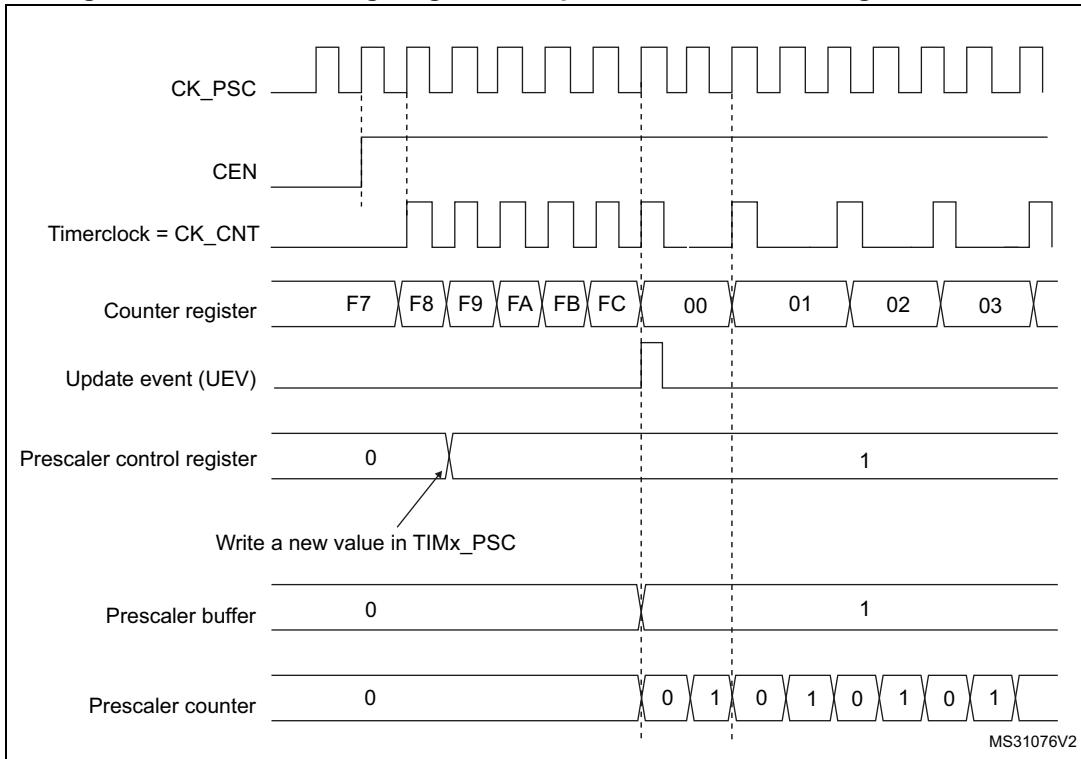
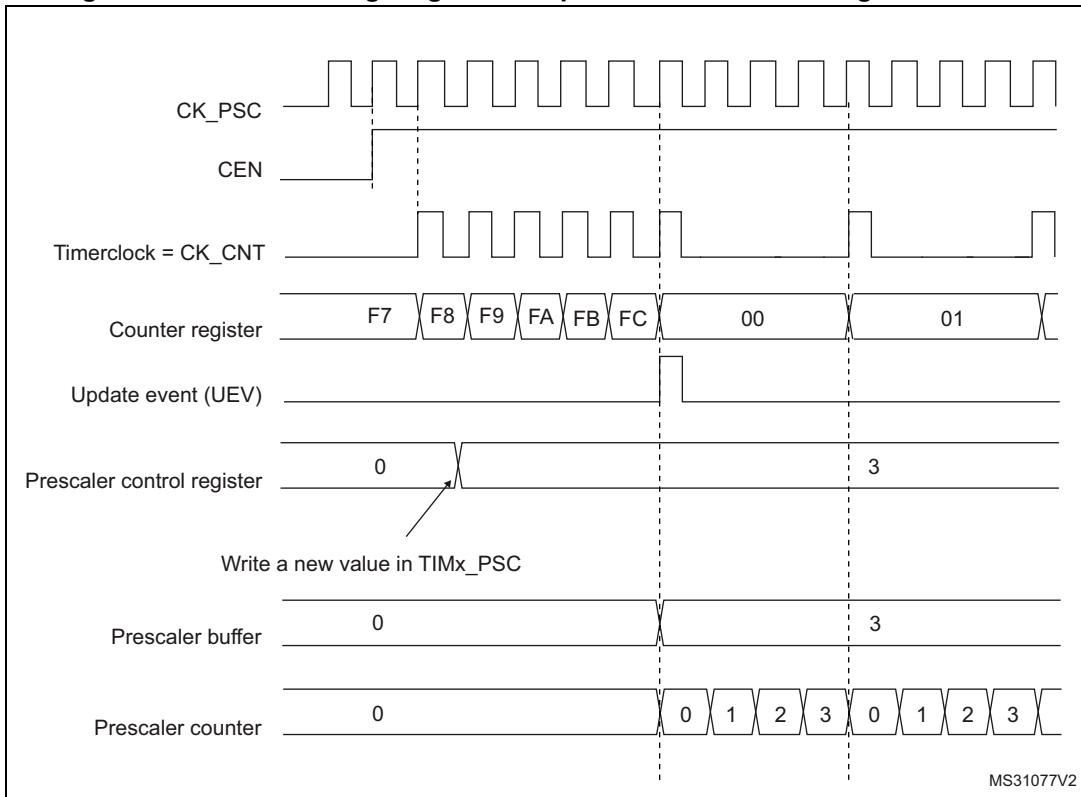
The counter is clocked by the prescaler output CK\_CNT, which is enabled only when the counter enable bit (CEN) in TIMx\_CR1 register is set (refer also to the slave mode controller description to get more details on counter enabling).

Note that the counter starts counting 1 clock cycle after setting the CEN bit in the TIMx\_CR1 register.

#### Prescaler description

The prescaler can divide the counter clock frequency by any factor between 1 and 65536. It is based on a 16-bit counter controlled through a 16-bit register (in the TIMx\_PSC register). It can be changed on the fly as this control register is buffered. The new prescaler ratio is taken into account at the next update event.

*Figure 65* and *Figure 66* give some examples of the counter behavior when the prescaler ratio is changed on the fly:

**Figure 64. Counter timing diagram with prescaler division change from 1 to 2****Figure 65. Counter timing diagram with prescaler division change from 1 to 4**

### 17.3.2 Counter modes

#### Upcounting mode

In upcounting mode, the counter counts from 0 to the auto-reload value (content of the TIMx\_ARR register), then restarts from 0 and generates a counter overflow event.

If the repetition counter is used, the update event (UEV) is generated after upcounting is repeated for the number of times programmed in the repetition counter register (TIMx\_RCR). Else the update event is generated at each counter overflow.

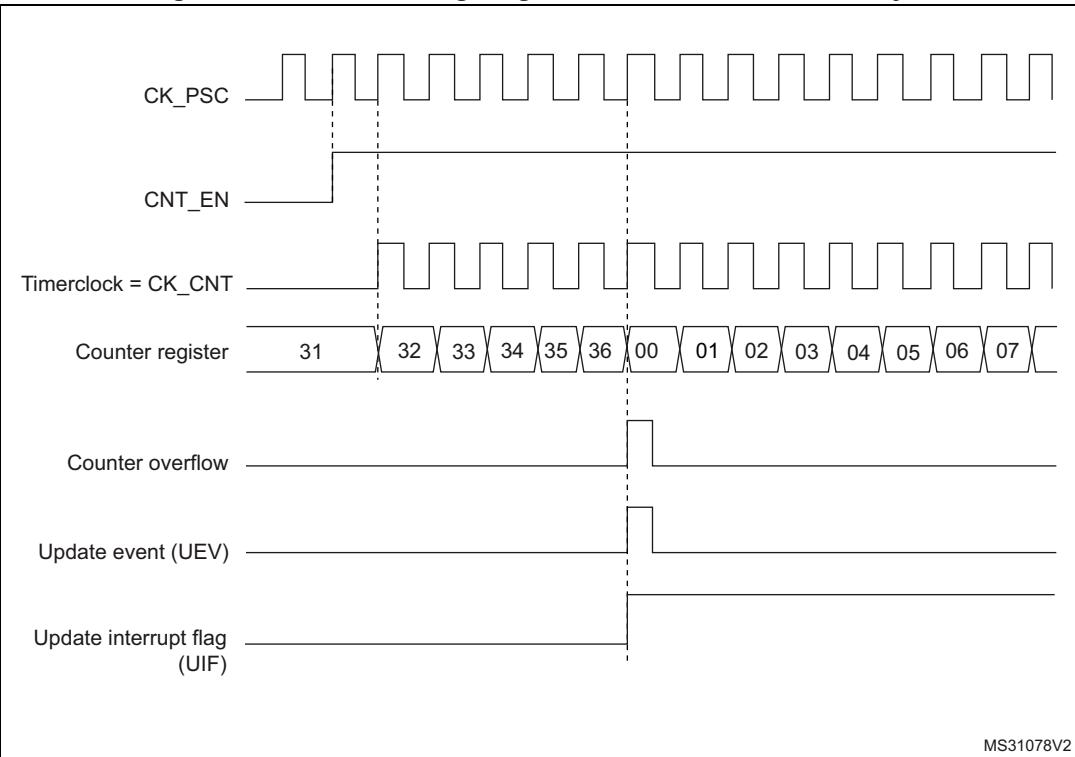
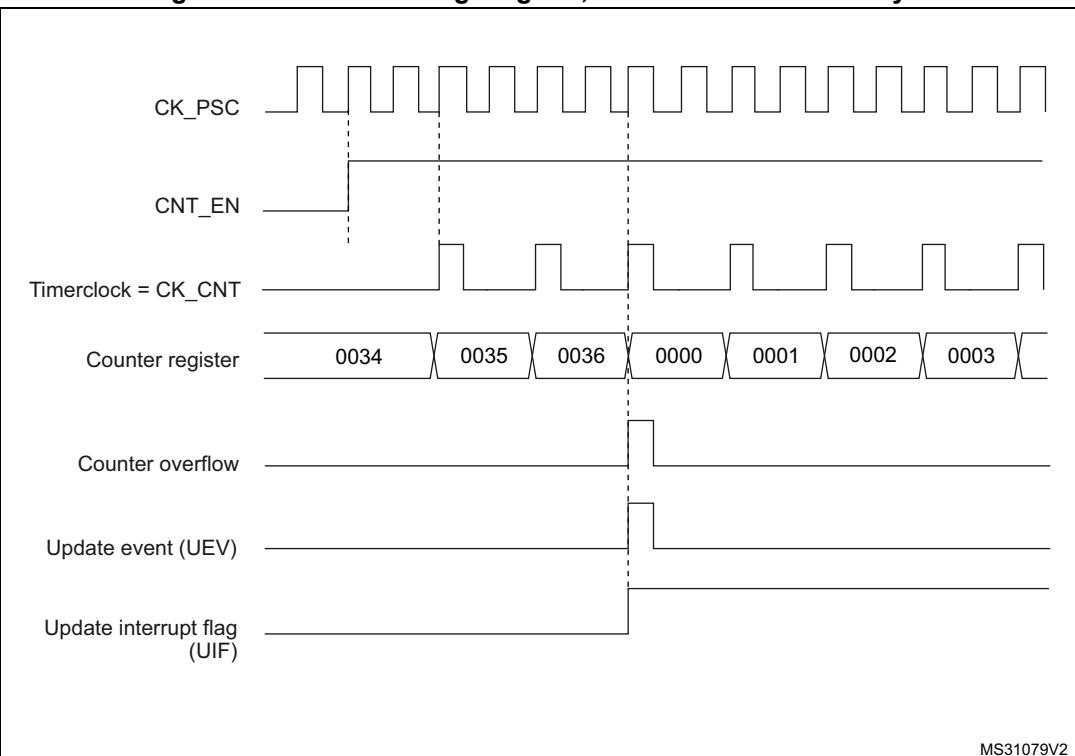
Setting the UG bit in the TIMx\_EGR register (by software or by using the slave mode controller) also generates an update event.

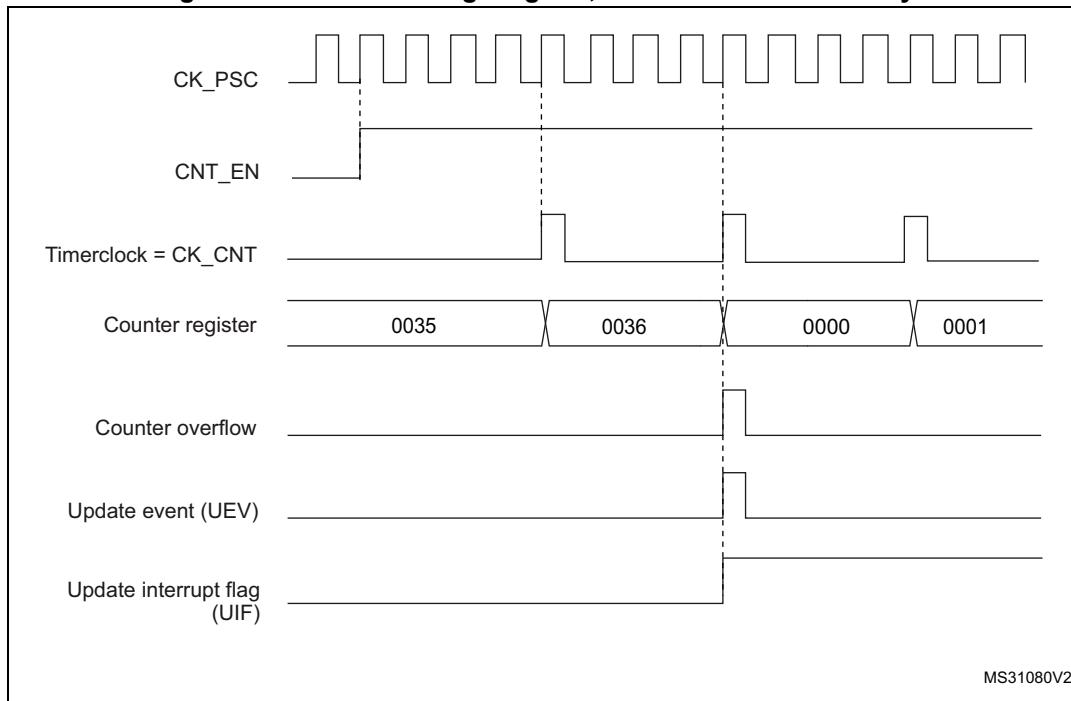
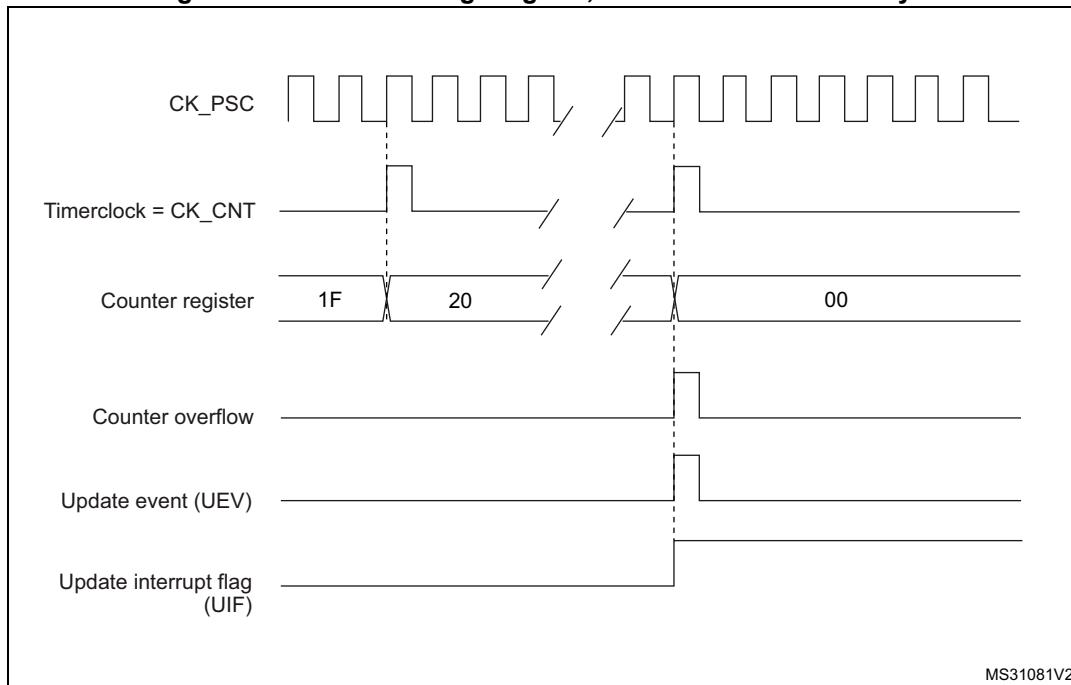
The UEV event can be disabled by software by setting the UDIS bit in the TIMx\_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until the UDIS bit has been written to 0. However, the counter restarts from 0, as well as the counter of the prescaler (but the prescale rate does not change). In addition, if the URS bit (update request selection) in TIMx\_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

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

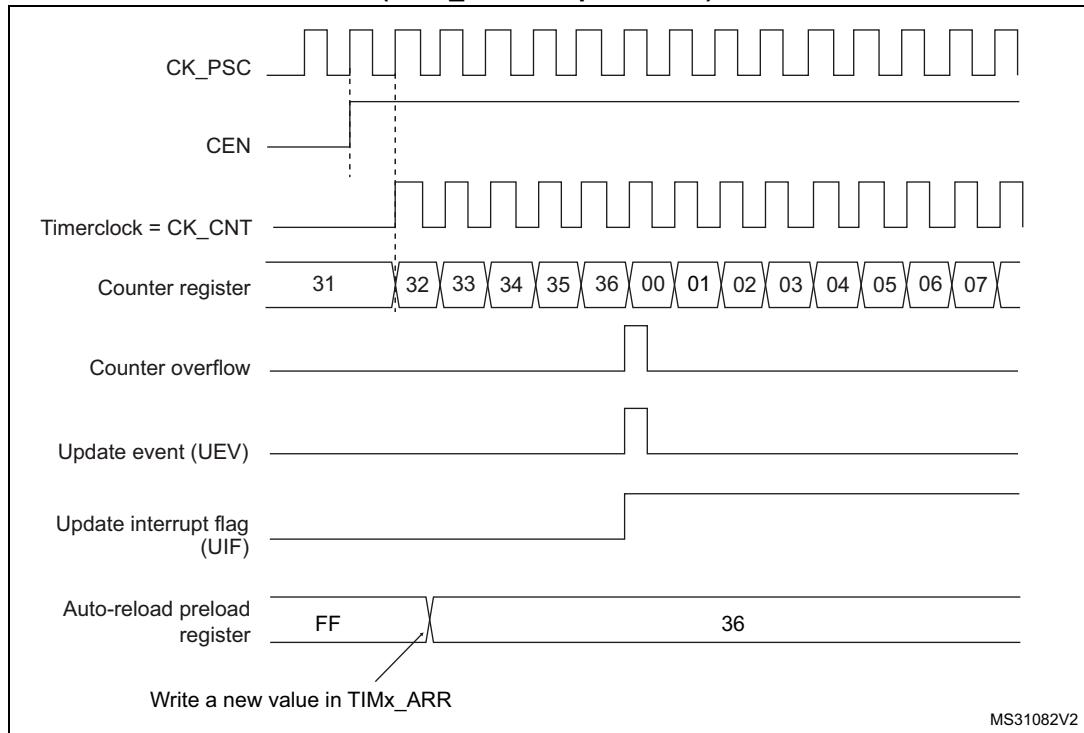
- The repetition counter is reloaded with the content of TIMx\_RCR register,
- The auto-reload shadow register is updated with the preload value (TIMx\_ARR),
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx\_PSC register).

The following figures show some examples of the counter behavior for different clock frequencies when TIMx\_ARR=0x36.

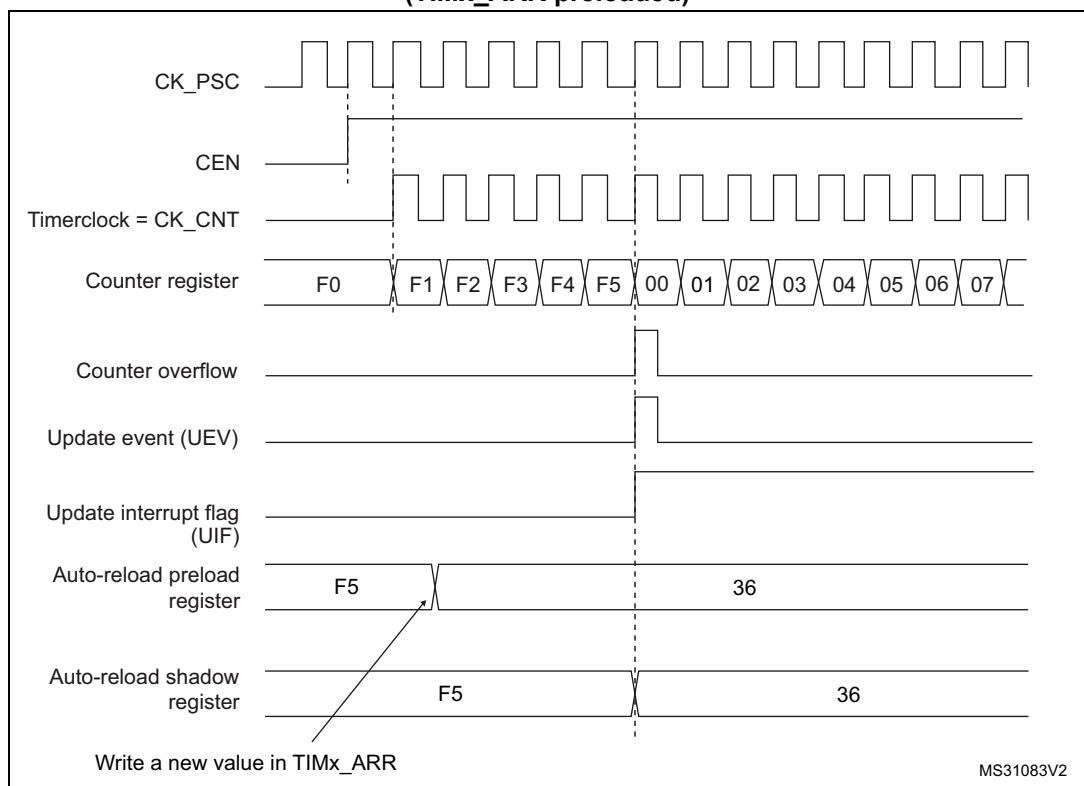
**Figure 66. Counter timing diagram, internal clock divided by 1****Figure 67. Counter timing diagram, internal clock divided by 2**

**Figure 68. Counter timing diagram, internal clock divided by 4****Figure 69. Counter timing diagram, internal clock divided by N**

**Figure 70. Counter timing diagram, update event when ARPE=0  
(TIMx\_ARR not preloaded)**



**Figure 71. Counter timing diagram, update event when ARPE=1  
(TIMx\_ARR preloaded)**



## Downcounting mode

In downcounting mode, the counter counts from the auto-reload value (content of the TIMx\_ARR register) down to 0, then restarts from the auto-reload value and generates a counter underflow event.

If the repetition counter is used, the update event (UEV) is generated after downcounting is repeated for the number of times programmed in the repetition counter register (TIMx\_RCR). Else the update event is generated at each counter underflow.

Setting the UG bit in the TIMx\_EGR register (by software or by using the slave mode controller) also generates an update event.

The UEV update event can be disabled by software by setting the UDIS bit in TIMx\_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until UDIS bit has been written to 0. However, the counter restarts from the current auto-reload value, whereas the counter of the prescaler restarts from 0 (but the prescale rate doesn't change).

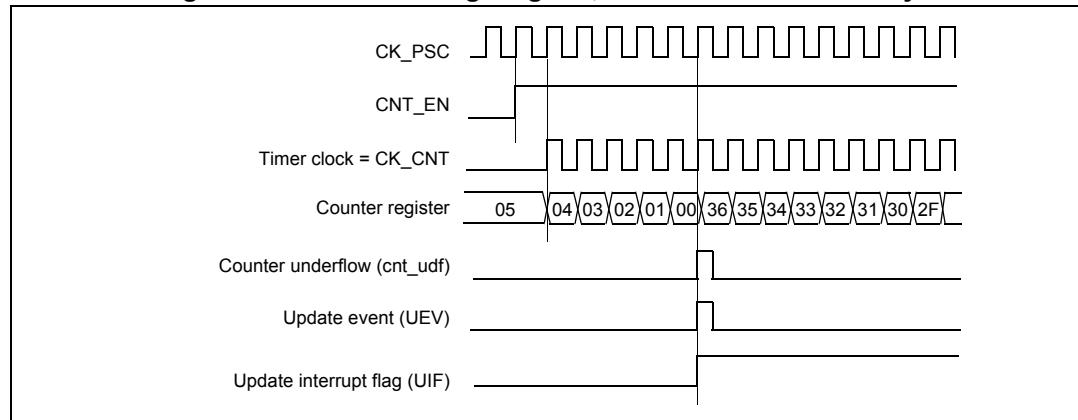
In addition, if the URS bit (update request selection) in TIMx\_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on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

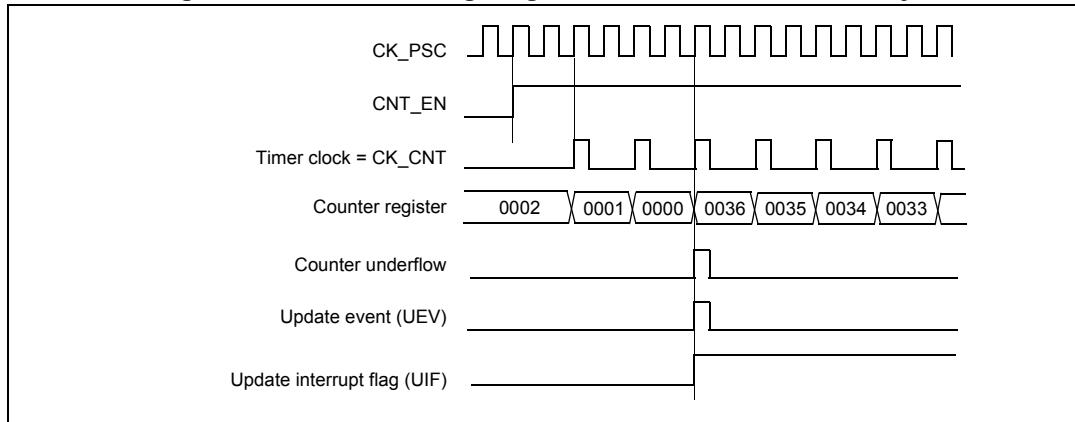
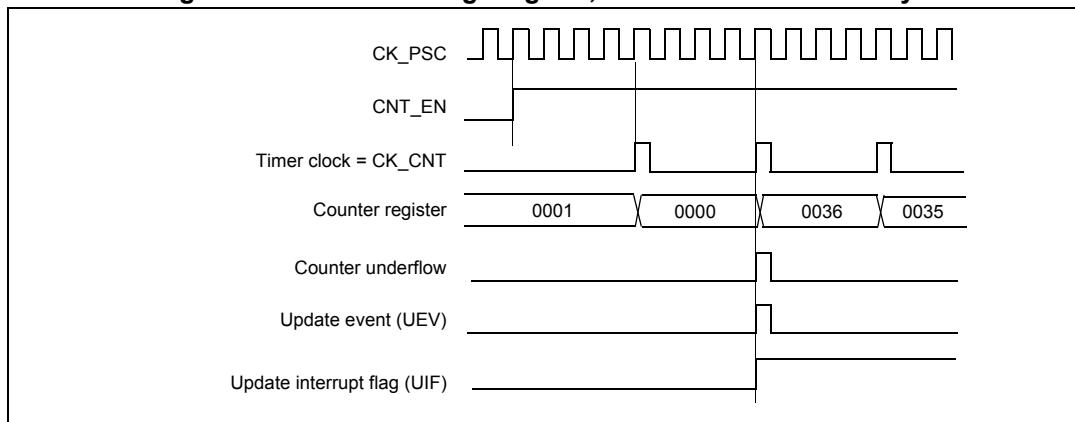
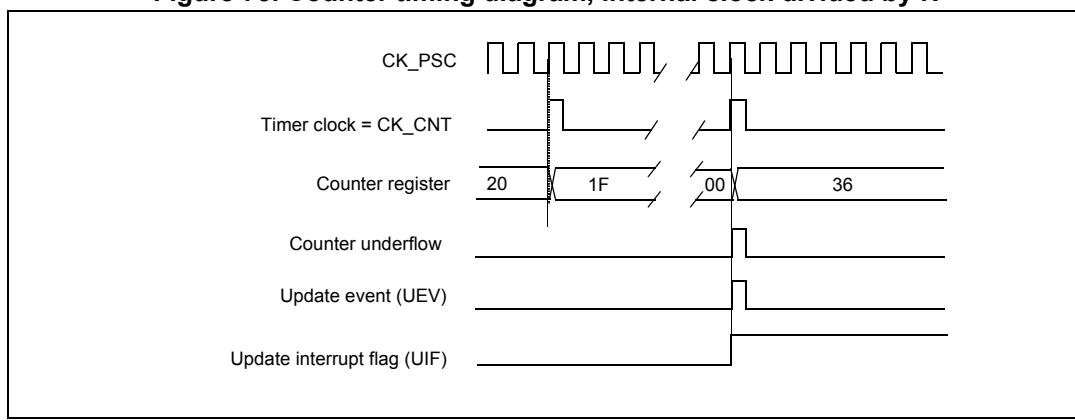
When an update event occurs, all the registers are updated and the update flag (UIF bit in TIMx\_SR register) is set (depending on the URS bit):

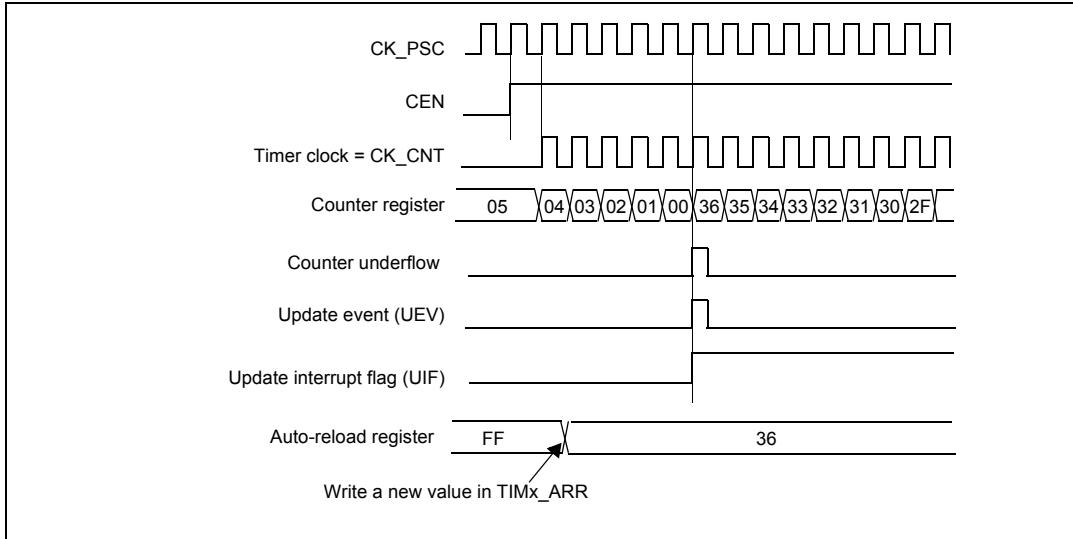
- The repetition counter is reloaded with the content of TIMx\_RCR register
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx\_PSC register)
- The auto-reload active register is updated with the preload value (content of the TIMx\_ARR register). Note that the auto-reload is updated before the counter is reloaded, so that the next period is the expected one

The following figures show some examples of the counter behavior for different clock frequencies when TIMx\_ARR=0x36.

**Figure 72. Counter timing diagram, internal clock divided by 1**



**Figure 73. Counter timing diagram, internal clock divided by 2****Figure 74. Counter timing diagram, internal clock divided by 4****Figure 75. Counter timing diagram, internal clock divided by N**

**Figure 76. Counter timing diagram, update event when repetition counter is not used**

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

In center-aligned mode, the counter counts from 0 to the auto-reload value (content of the TIMx\_ARR register) – 1, generates a counter overflow event, then counts from the auto-reload value down to 1 and generates a counter underflow event. Then it restarts counting from 0.

Center-aligned mode is active when the CMS bits in TIMx\_CR1 register are not equal to '00'. The Output compare interrupt flag of channels configured in output is set when: the counter counts down (Center aligned mode 1, CMS = "01"), the counter counts up (Center aligned mode 2, CMS = "10") the counter counts up and down (Center aligned mode 3, CMS = "11").

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

The update event can be generated at each counter overflow and at each counter underflow or by setting the UG bit in the TIMx\_EGR register (by software or by using the slave mode controller) also generates an update event. In this case, the counter restarts counting from 0, as well as the counter of the prescaler.

The UEV update event can be disabled by software by setting the UDIS bit in the TIMx\_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until UDIS bit has been written to 0. However, the counter continues counting up and down, based on the current auto-reload value.

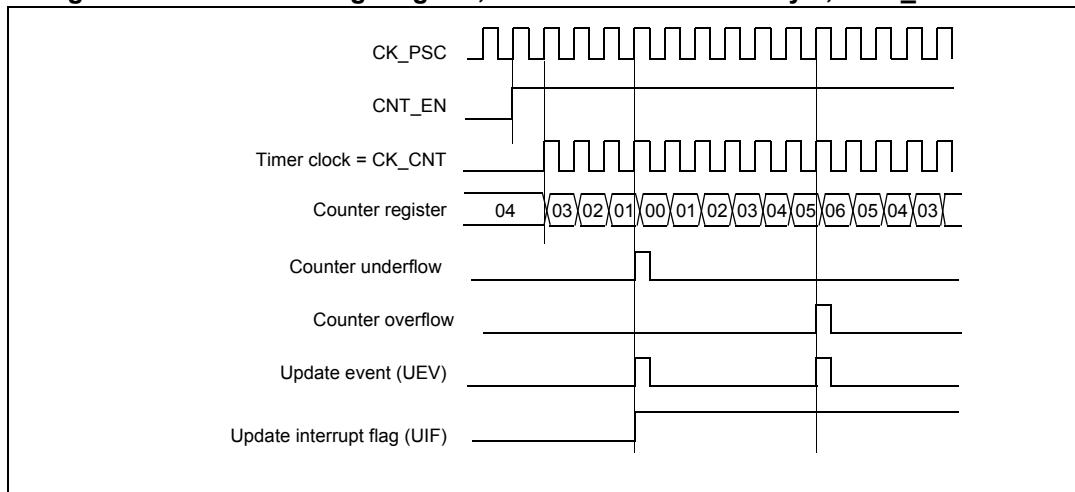
In addition, if the URS bit (update request selection) in TIMx\_CR1 register is set, setting the UG bit generates an UEV update event but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on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

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

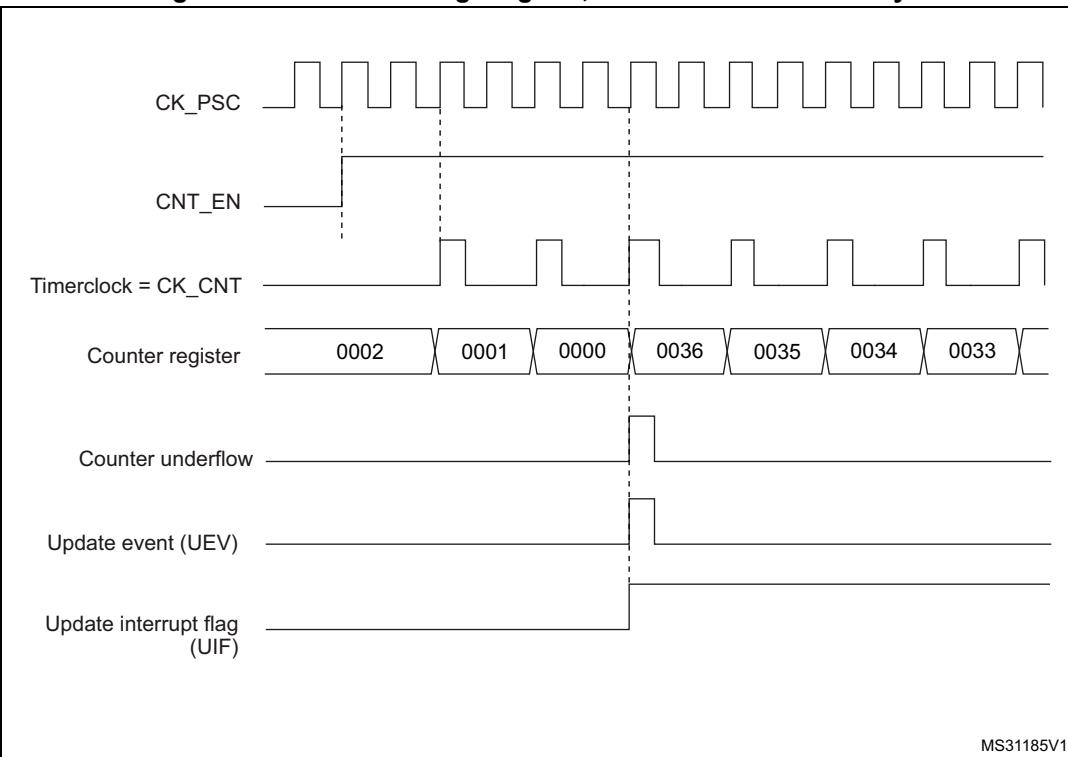
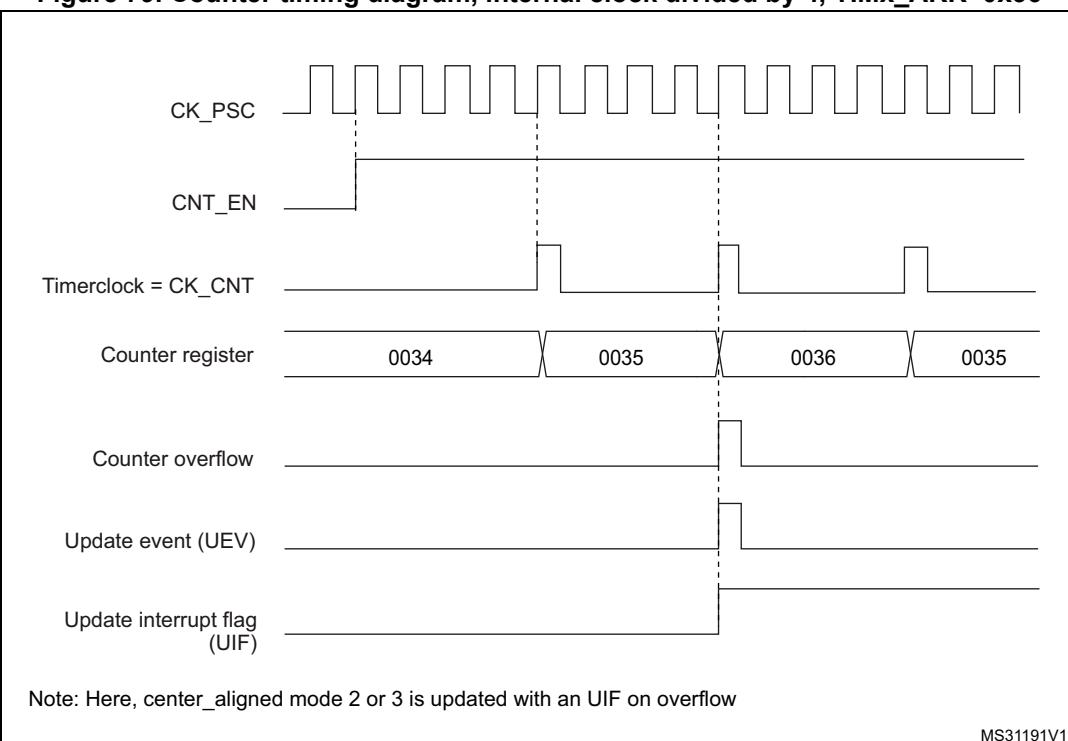
- The repetition counter is reloaded with the content of TIMx\_RCR register
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx\_PSC register)
- The auto-reload active register is updated with the preload value (content of the TIMx\_ARR register). Note that if the update source is a counter overflow, the auto-reload is updated before the counter is reloaded, so that the next period is the expected one (the counter is loaded with the new value).

The following figures show some examples of the counter behavior for different clock frequencies.

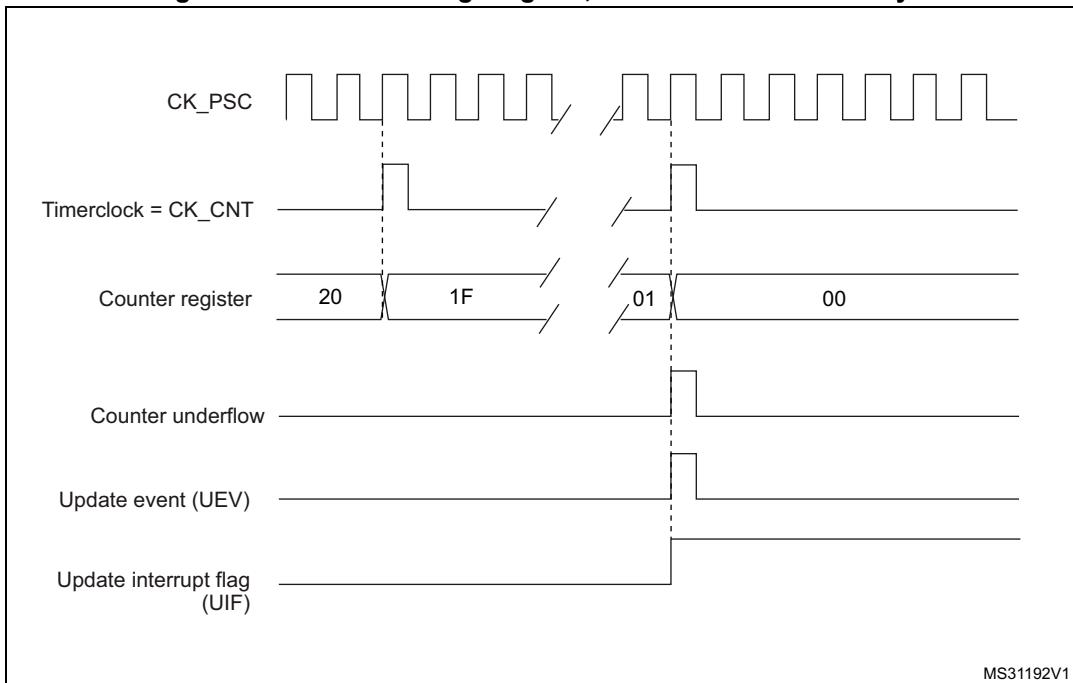
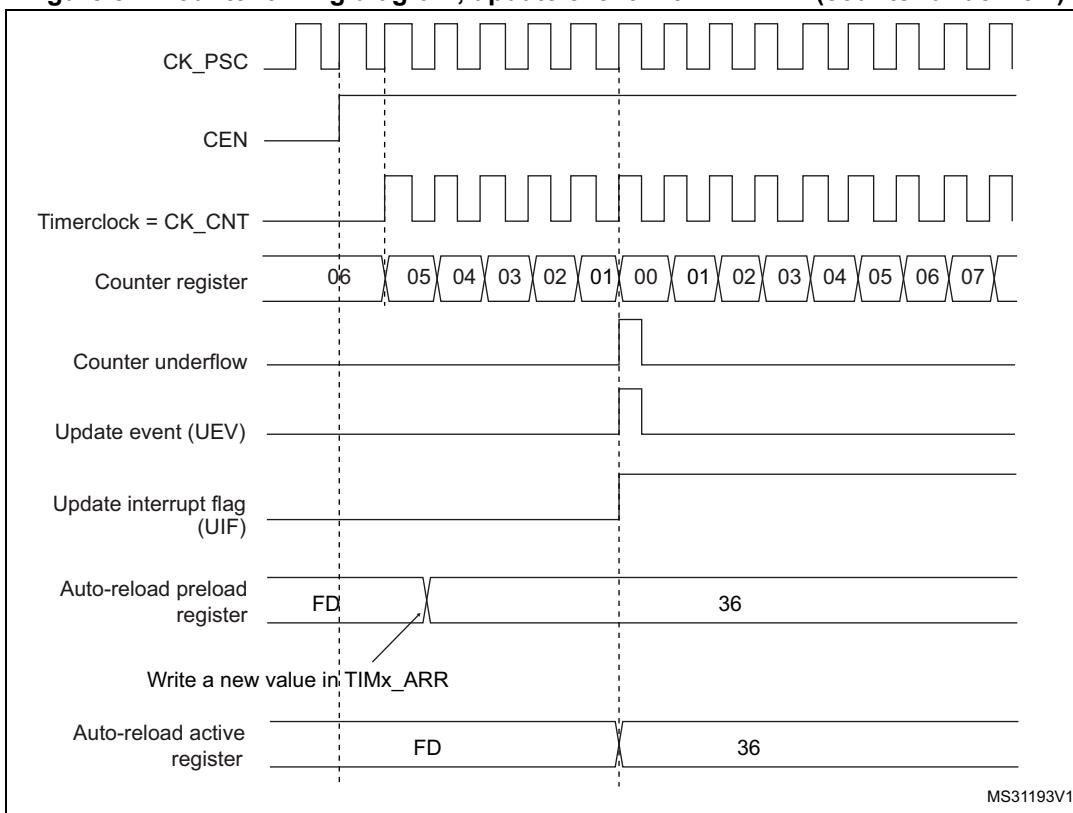
**Figure 77. Counter timing diagram, internal clock divided by 1, TIMx\_ARR = 0x6**

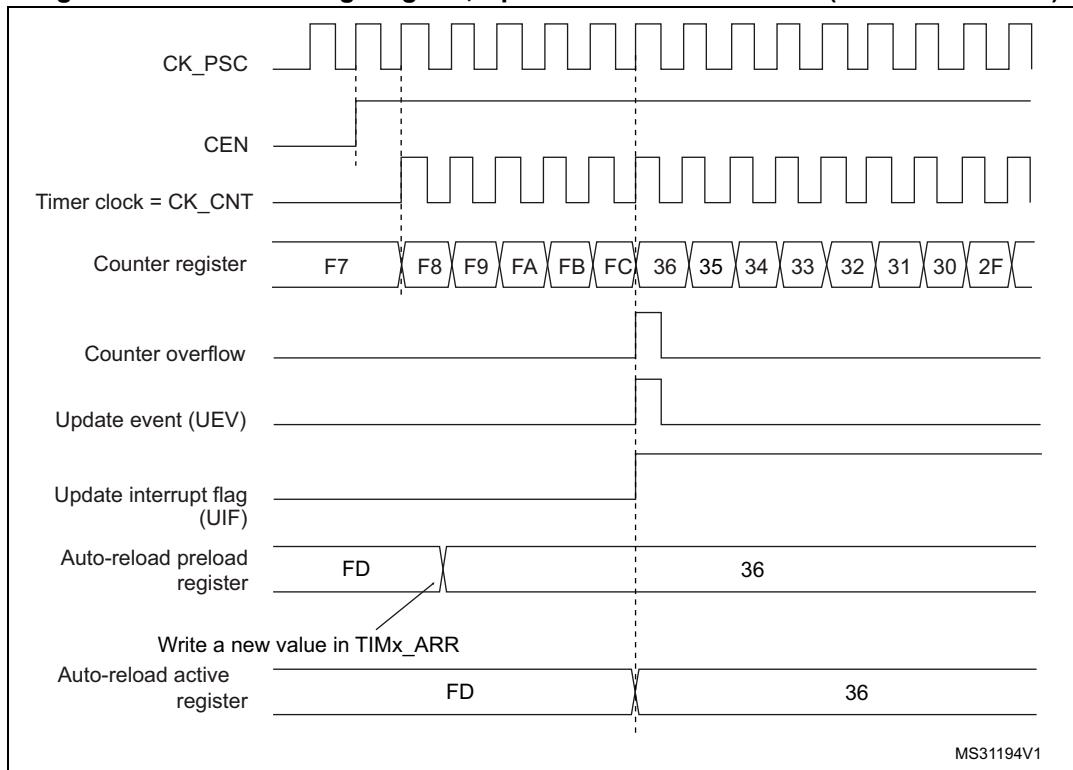


1. Here, center-aligned mode 1 is used (for more details refer to [Section 17.4: TIM1 registers on page 375](#)).

**Figure 78. Counter timing diagram, internal clock divided by 2****Figure 79. Counter timing diagram, internal clock divided by 4, TIMx\_ARR=0x36**

1. Center-aligned mode 2 or 3 is used with an UIF on overflow.

**Figure 80. Counter timing diagram, internal clock divided by N****Figure 81. Counter timing diagram, update event with ARPE=1 (counter underflow)**

**Figure 82. Counter timing diagram, Update event with ARPE=1 (counter overflow)**

### 17.3.3 Repetition counter

[Section 17.3.1: Time-base unit](#) describes how the update event (UEV) is generated with respect to the counter overflows/underflows. It is actually generated only when the repetition counter has reached zero. This can be useful when generating PWM signals.

This means that data are transferred from the preload registers to the shadow registers (TIMx\_ARR auto-reload register, TIMx\_PSC prescaler register, but also TIMx\_CCRx capture/compare registers in compare mode) every N counter overflows or underflows, where N is the value in the TIMx\_RCR repetition counter register.

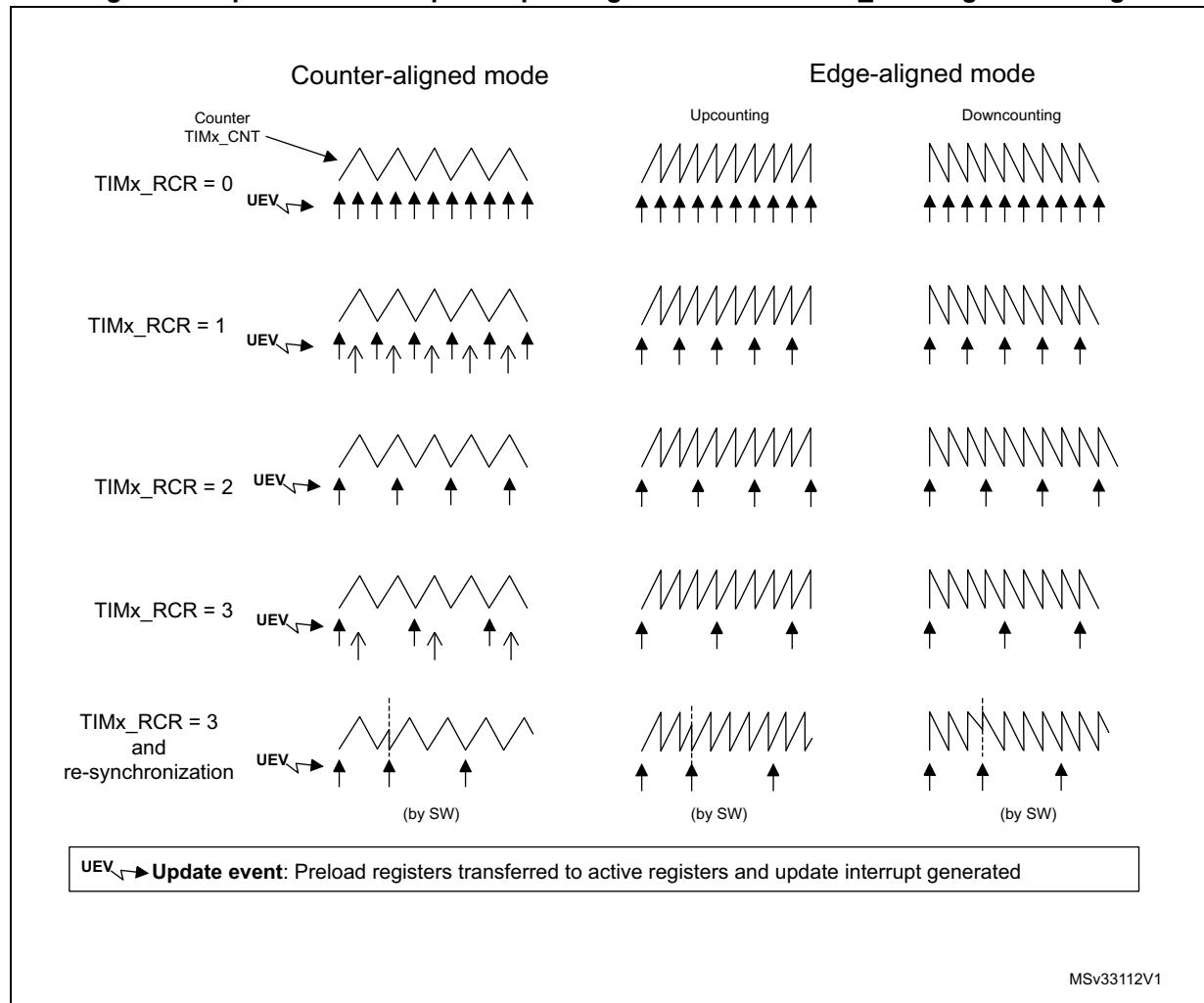
The repetition counter is decremented:

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

The repetition counter is an auto-reload type; the repetition rate is maintained as defined by the TIMx\_RCR register value (refer to [Figure 83](#)). When the update event is generated by software (by setting the UG bit in TIMx\_EGR register) or by hardware through the slave mode controller, it occurs immediately whatever the value of the repetition counter is and the repetition counter is reloaded with the content of the TIMx\_RCR register.

In center-aligned mode, for odd values of RCR, the update event occurs either on the overflow or on the underflow depending on when the RCR register was written and when the counter was started. If the RCR was written before starting the counter, the UEV occurs on the overflow. If the RCR was written after starting the counter, the UEV occurs on the underflow. For example for RCR = 3, the UEV is generated on each 4th overflow or underflow event depending on when RCR was written.

**Figure 83. Update rate examples depending on mode and TIMx\_RCR register settings**



MSv33112V1

### 17.3.4 Clock sources

The counter clock can be provided by the following clock sources:

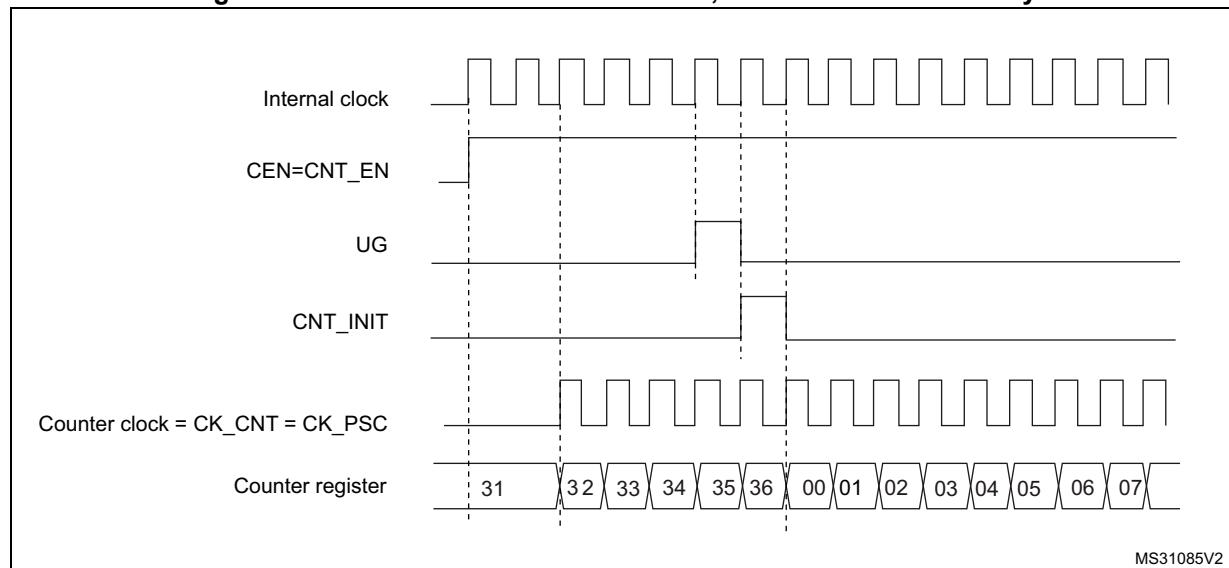
- Internal clock (CK\_INT)
- External clock mode1: external input pin
- External clock mode2: external trigger input ETR
- Internal trigger inputs (ITRx): using one timer as prescaler for another timer, for example, Timer 1 can be configured to act as a prescaler for Timer 2. Refer to [Using one timer as prescaler for another on page 440](#) for more details.

#### Internal clock source (CK\_INT)

If the slave mode controller is disabled (SMS=000), then the CEN, DIR (in the TIMx\_CR1 register) and UG bits (in the TIMx\_EGR register) are actual control bits and can be changed only by software (except UG which remains cleared automatically). As soon as the CEN bit is written to 1, the prescaler is clocked by the internal clock CK\_INT.

[Figure 84](#) shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

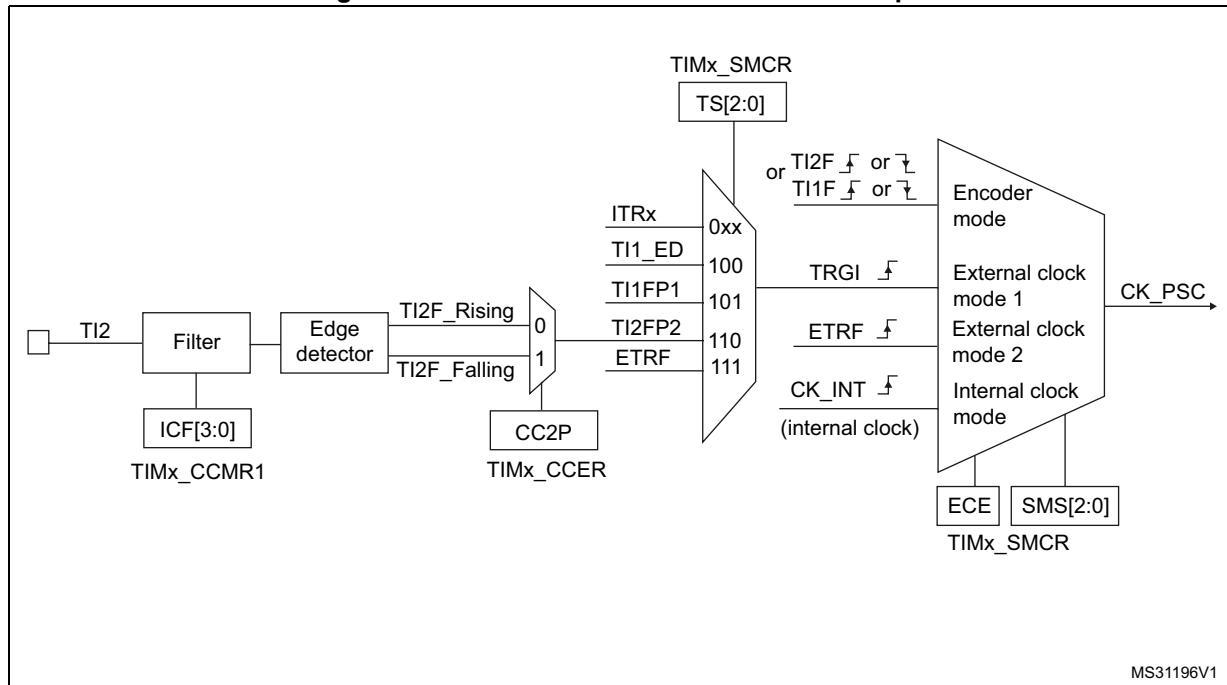
**Figure 84. Control circuit in normal mode, internal clock divided by 1**



### External clock source mode 1

This mode is selected when SMS=111 in the TIMx\_SMCR register. The counter can count at each rising or falling edge on a selected input.

**Figure 85. TI2 external clock connection example**



For example, to configure the upcounter to count in response to a rising edge on the TI2 input, use the following procedure:

1. Configure channel 2 to detect rising edges on the TI2 input by writing CC2S = '01' in the TIMx\_CCMR1 register.
2. Configure the input filter duration by writing the IC2F[3:0] bits in the TIMx\_CCMR1 register (if no filter is needed, keep IC2F=0000).
3. Select rising edge polarity by writing CC2P=0 in the TIMx\_CCER register.
4. Configure the timer in external clock mode 1 by writing SMS=111 in the TIMx\_SMCR register.
5. Select TI2 as the trigger input source by writing TS=110 in the TIMx\_SMCR register.
6. Enable the counter by writing CEN=1 in the TIMx\_CR1 register.

*Note:*

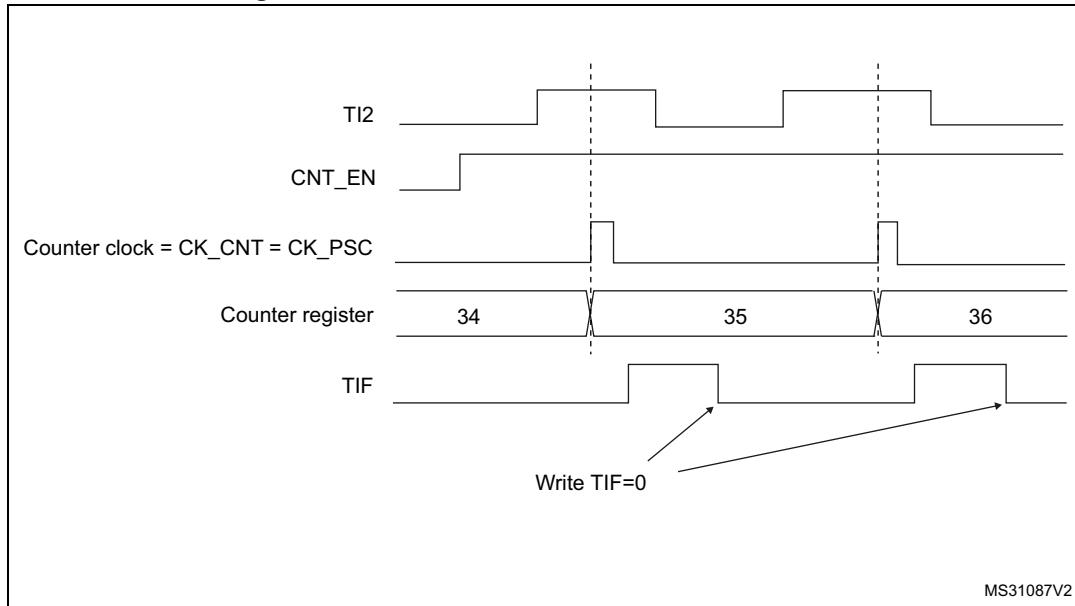
*The capture prescaler is not used for triggering, so it does not need to be configured.*

For code examples refer to the Appendix section [A.9.1: Upcounter on TI2 rising edge code example](#).

When a rising edge occurs on TI2, the counter counts once and the TIF flag is set.

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

Figure 86. Control circuit in external clock mode 1



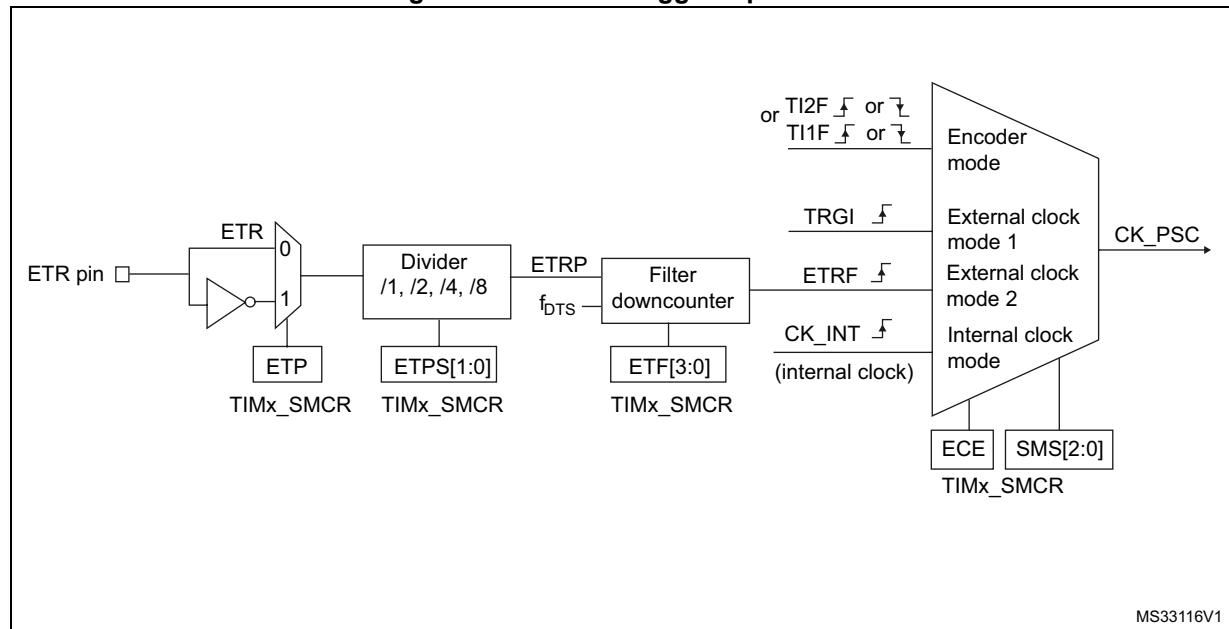
### External clock source mode 2

This mode is selected by writing ECE=1 in the TIMx\_SMCR register.

The counter can count at each rising or falling edge on the external trigger input ETR.

The [Figure 87](#) gives an overview of the external trigger input block.

Figure 87. External trigger input block



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

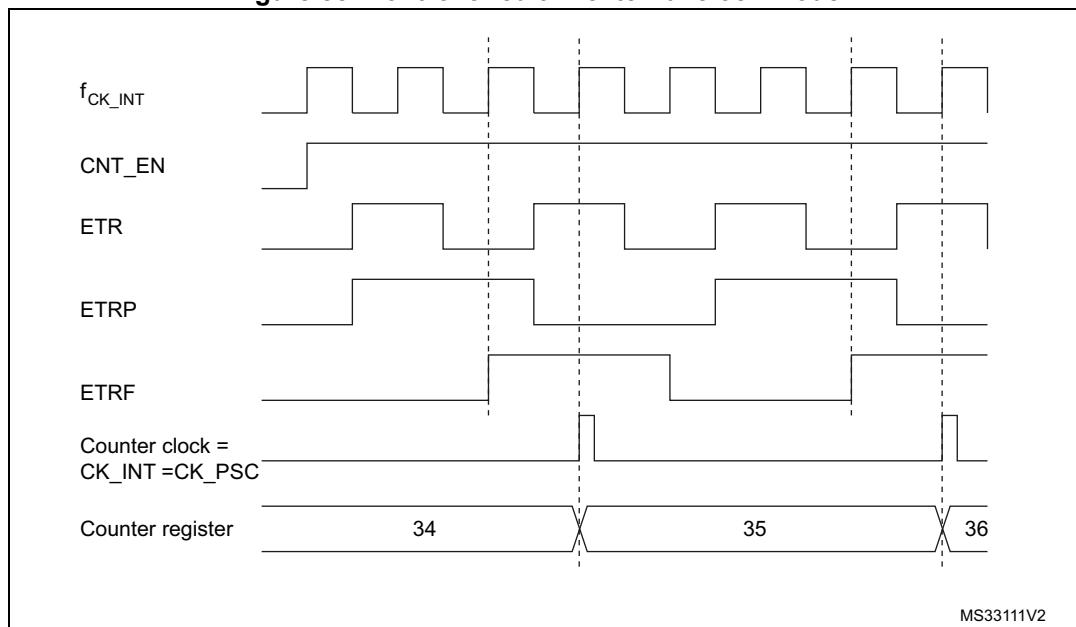
1. As no filter is needed in this example, write ETF[3:0]=0000 in the TIMx\_SMCR register.
2. Set the prescaler by writing ETPS[1:0]=01 in the TIMx\_SMCR register
3. Select rising edge detection on the ETR pin by writing ETP=0 in the TIMx\_SMCR register
4. Enable external clock mode 2 by writing ECE=1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

The counter counts once each 2 ETR rising edges.

For code example refer to the Appendix section [A.9.2: Up counter on each 2 ETR rising edges code example](#).

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

**Figure 88. Control circuit in external clock mode 2**



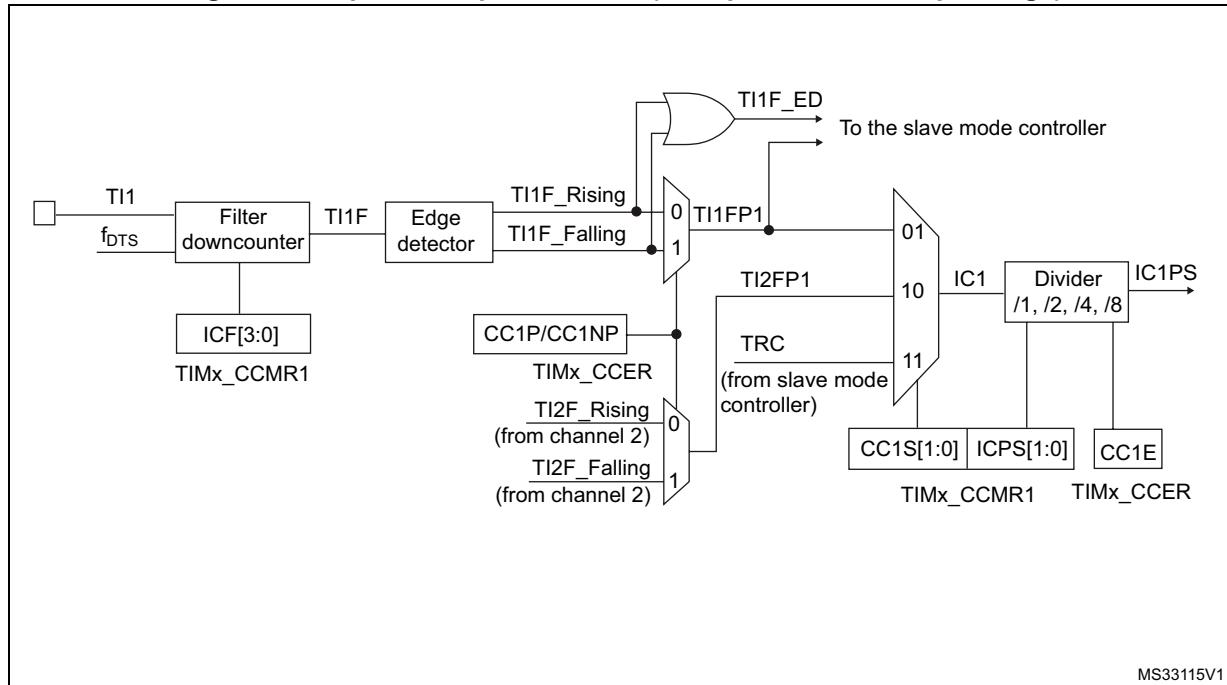
### 17.3.5 Capture/compare channels

Each Capture/Compare channel is built around a capture/compare register (including a shadow register), a input stage for capture (with digital filter, multiplexing and prescaler) and an output stage (with comparator and output control).

[Figure 89](#) to [Figure 92](#) give an overview of one Capture/Compare channel.

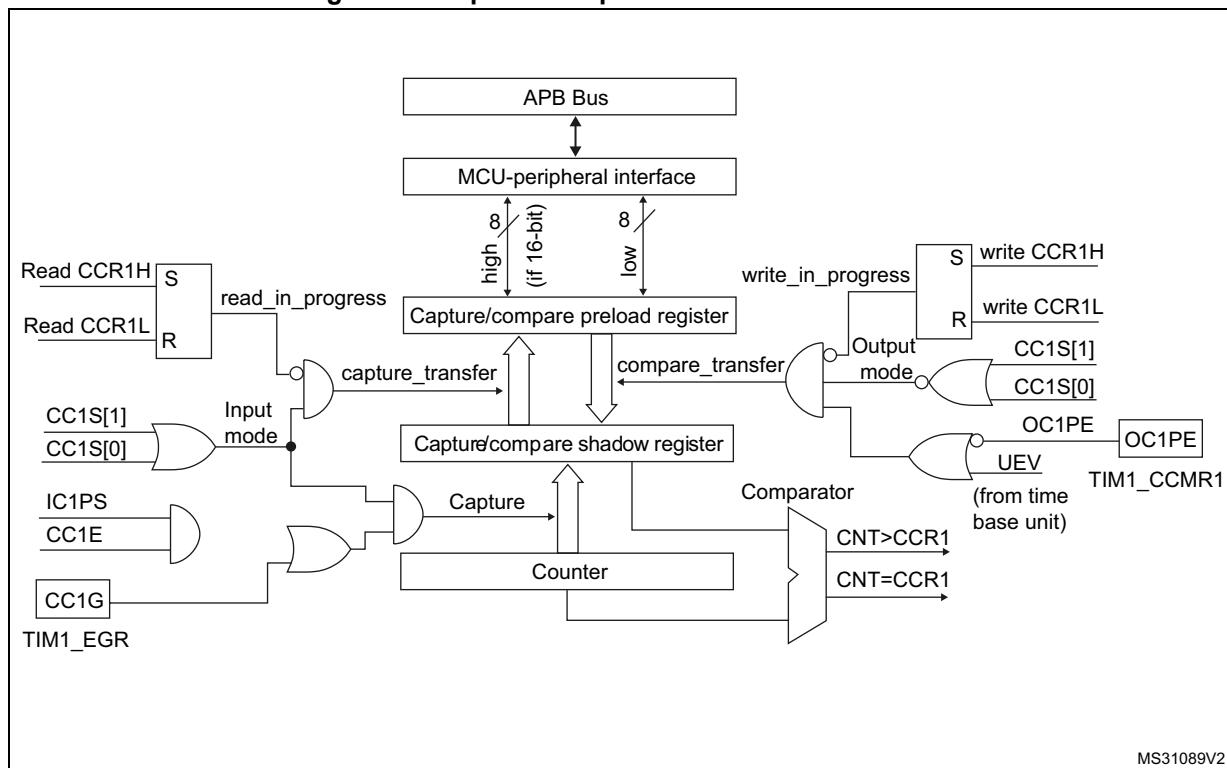
The input stage samples the corresponding TIx input to generate a filtered signal TIxF. Then, an edge detector with polarity selection generates a signal (TIxFPx) which can be used as trigger input by the slave mode controller or as the capture command. It is prescaled before the capture register (ICxPS).

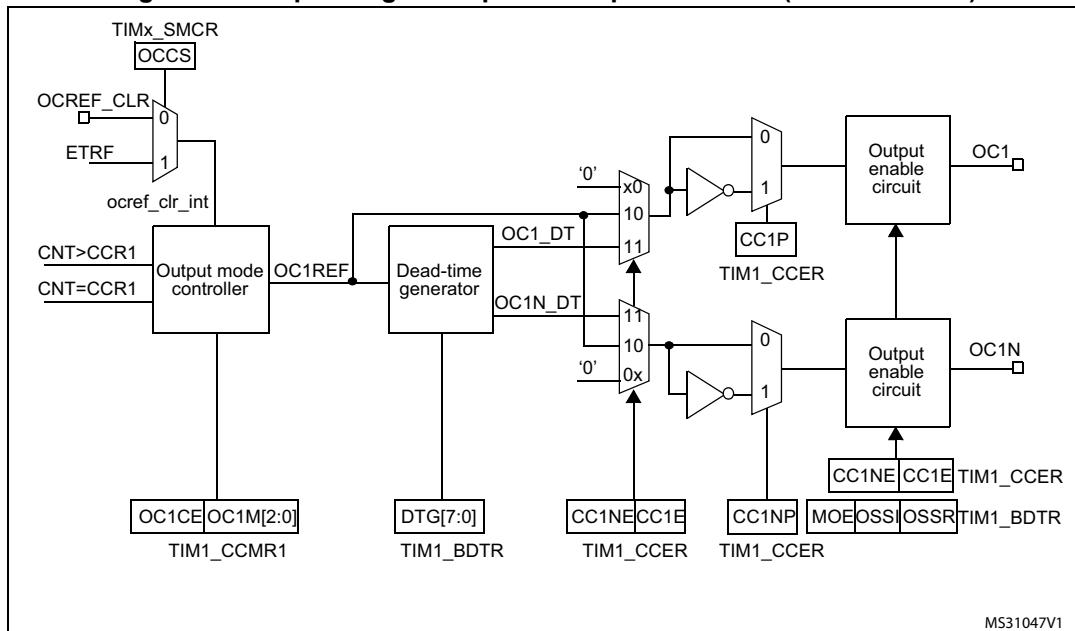
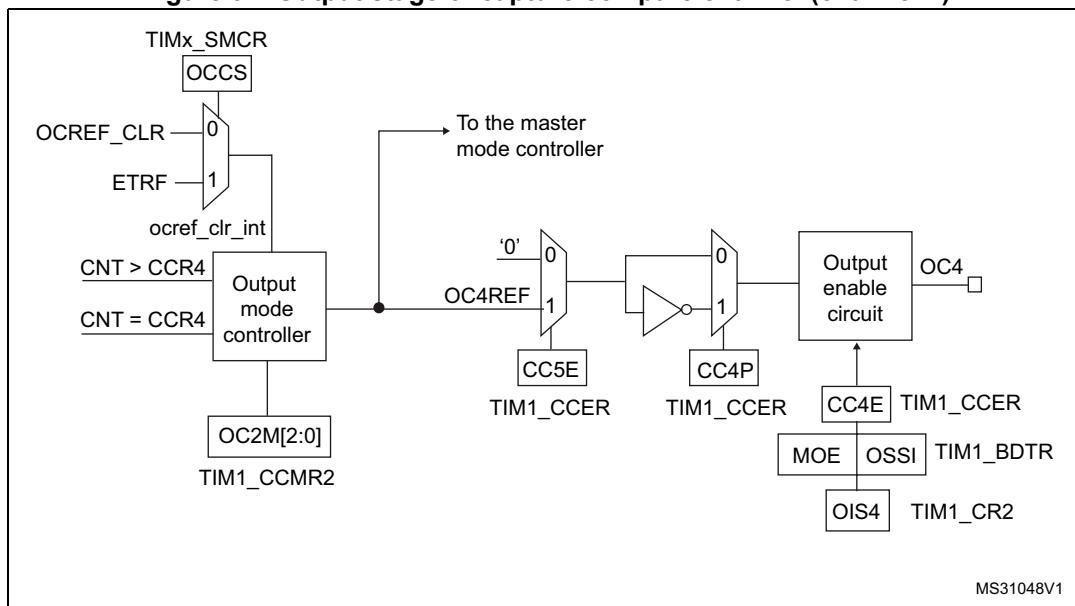
Figure 89. Capture/compare channel (example: channel 1 input stage)



The output stage generates an intermediate waveform which is then used for reference: OCxRef (active high). The polarity acts at the end of the chain.

Figure 90. Capture/compare channel 1 main circuit



**Figure 91. Output stage of capture/compare channel (channel 1 to 3)****Figure 92. Output stage of capture/compare channel (channel 4)**

The capture/compare block is made of one preload register and one shadow register. Write and read always access the preload register.

In capture mode, captures are actually done in the shadow register, which is copied into the preload register.

In compare mode, the content of the preload register is copied into the shadow register which is compared to the counter.

### 17.3.6 Input capture mode

In Input capture mode, the Capture/Compare Registers (TIMx\_CCRx) are used to latch the value of the counter after a transition detected by the corresponding ICx signal. When a capture occurs, the corresponding CCxIF flag (TIMx\_SR register) is set and an interrupt or a DMA request can be sent if they are enabled. If a capture occurs while the CCxIF flag was already high, then the over-capture flag CCxOF (TIMx\_SR register) is set. CCxIF can be cleared by software by writing it to '0' or by reading the captured data stored in the TIMx\_CCRx register. CCxOF is cleared when it is written with 0.

The following example shows how to capture the counter value in TIMx\_CCR1 when TI1 input rises. To do this, use the following procedure:

- Select the active input: TIMx\_CCR1 must be linked to the TI1 input, so write the CC1S bits to 01 in the TIMx\_CCMR1 register. As soon as CC1S becomes different from 00, the channel is configured in input and the TIMx\_CCR1 register becomes read-only.
- Program the appropriate input filter duration in relation with the signal connected to the timer (when the input is one of the TIx (ICxF bits in the TIMx\_CCMRx register). Let's imagine that, when toggling, the input signal is not stable during at most 5 internal clock cycles. We must program a filter duration longer than these 5 clock cycles. We can validate a transition on TI1 when 8 consecutive samples with the new level have been detected (sampled at  $f_{DTS}$  frequency). Then write IC1F bits to 0011 in the TIMx\_CCMR1 register.
- Select the edge of the active transition on the TI1 channel by writing CC1P and CC1NP bits to 0 in the TIMx\_CCER register (rising edge in this case).
- Program the input prescaler. In our example, we wish the capture to be performed at each valid transition, so the prescaler is disabled (write IC1PS bits to '00' in the TIMx\_CCMR1 register).
- Enable capture from the counter into the capture register by setting the CC1E bit in the TIMx\_CCER register.
- If needed, enable the related interrupt request by setting the CC1IE bit in the TIMx\_DIER register, and/or the DMA request by setting the CC1DE bit in the TIMx\_DIER register.

For code example refer to the Appendix section [A.9.3: Input capture configuration code example](#).

When an input capture occurs:

- The TIMx\_CCR1 register gets the value of the counter on the active transition.
- CC1IF flag is set (interrupt flag). CC1OF is also set if at least two consecutive captures occurred whereas the flag was not cleared.
- An interrupt is generated depending on the CC1IE bit.
- A DMA request is generated depending on the CC1DE bit.

For code example refer to the Appendix section [A.9.4: Input capture data management code example](#).

In order to handle the overcapture, it is recommended to read the data before the overcapture flag. This is to avoid missing an overcapture which could happen after reading the flag and before reading the data.

**Note:** *IC interrupt and/or DMA requests can be generated by software by setting the corresponding CCxG bit in the TIMx\_EGR register.*

### 17.3.7 PWM input mode

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

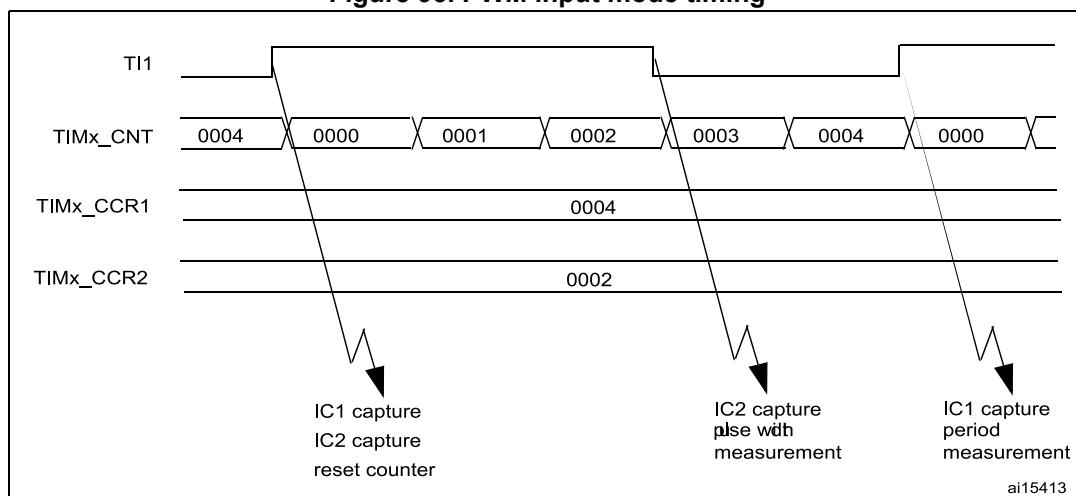
- Two ICx signals are mapped on the same TIx input.
- These 2 ICx signals are active on edges with opposite polarity.
- One of the two TIxFP signals is selected as trigger input and the slave mode controller is configured in reset mode.

For example, one can measure the period (in TIMx\_CCR1 register) and the duty cycle (in TIMx\_CCR2 register) of the PWM applied on TI1 using the following procedure (depending on CK\_INT frequency and prescaler value):

- Select the active input for TIMx\_CCR1: write the CC1S bits to 01 in the TIMx\_CCMR1 register (TI1 selected).
- Select the active polarity for TI1FP1 (used both for capture in TIMx\_CCR1 and counter clear): write the CC1P and CC1NP bits to '0' (active on rising edge).
- Select the active input for TIMx\_CCR2: write the CC2S bits to 10 in the TIMx\_CCMR1 register (TI1 selected).
- Select the active polarity for TI1FP2 (used for capture in TIMx\_CCR2): write the CC2P bit to '1' and the CC2NP bit to '0' (active on falling edge).
- Select the valid trigger input: write the TS bits to 101 in the TIMx\_SMCR register (TI1FP1 selected).
- Configure the slave mode controller in reset mode: write the SMS bits to 100 in the TIMx\_SMCR register.
- Enable the captures: write the CC1E and CC2E bits to '1' in the TIMx\_CCER register.

For code example refer to the Appendix section [A.9.5: PWM input configuration code example](#).

**Figure 93. PWM input mode timing**



### 17.3.8 Forced output mode

In output mode (CCxS bits = 00 in the TIMx\_CCMRx register), each output compare signal (OCxREF and then OCx/OCxN) can be forced to active or inactive level directly by software, independently of any comparison between the output compare register and the counter.

To force an output compare signal (OCXREF/OCx) to its active level, one just needs to write 101 in the OCxM bits in the corresponding TIMx\_CCMRx register. Thus OCXREF is forced high (OCxREF is always active high) and OCx get opposite value to CCxP polarity bit.

For example: CCxP=0 (OCx active high) => OCx is forced to high level.

The OCxREF signal can be forced low by writing the OCxM bits to 100 in the TIMx\_CCMRx register.

Anyway, the comparison between the TIMx\_CCRx shadow register and the counter is still performed and allows the flag to be set. Interrupt and DMA requests can be sent accordingly. This is described in the output compare mode section below.

### 17.3.9 Output compare mode

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

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

- Assigns the corresponding output pin to a programmable value defined by the output compare mode (OCxM bits in the TIMx\_CCMRx register) and the output polarity (CCxP bit in the TIMx\_CCER register). The output pin can keep its level (OCXM=000), be set active (OCXM=001), be set inactive (OCXM=010) or can toggle (OCXM=011) on match.
- Sets a flag in the interrupt status register (CCxIF bit in the TIMx\_SR register).
- Generates an interrupt if the corresponding interrupt mask is set (CCXIE bit in the TIMx\_DIER register).
- Sends a DMA request if the corresponding enable bit is set (CCxDE bit in the TIMx\_DIER register, CCDS bit in the TIMx\_CR2 register for the DMA request selection).

The TIMx\_CCRx registers can be programmed with or without preload registers using the OCxPE bit in the TIMx\_CCMRx register.

In output compare mode, the update event UEV has no effect on OCxREF and OCx output. The timing resolution is one count of the counter. Output compare mode can also be used to output a single pulse (in One Pulse mode).

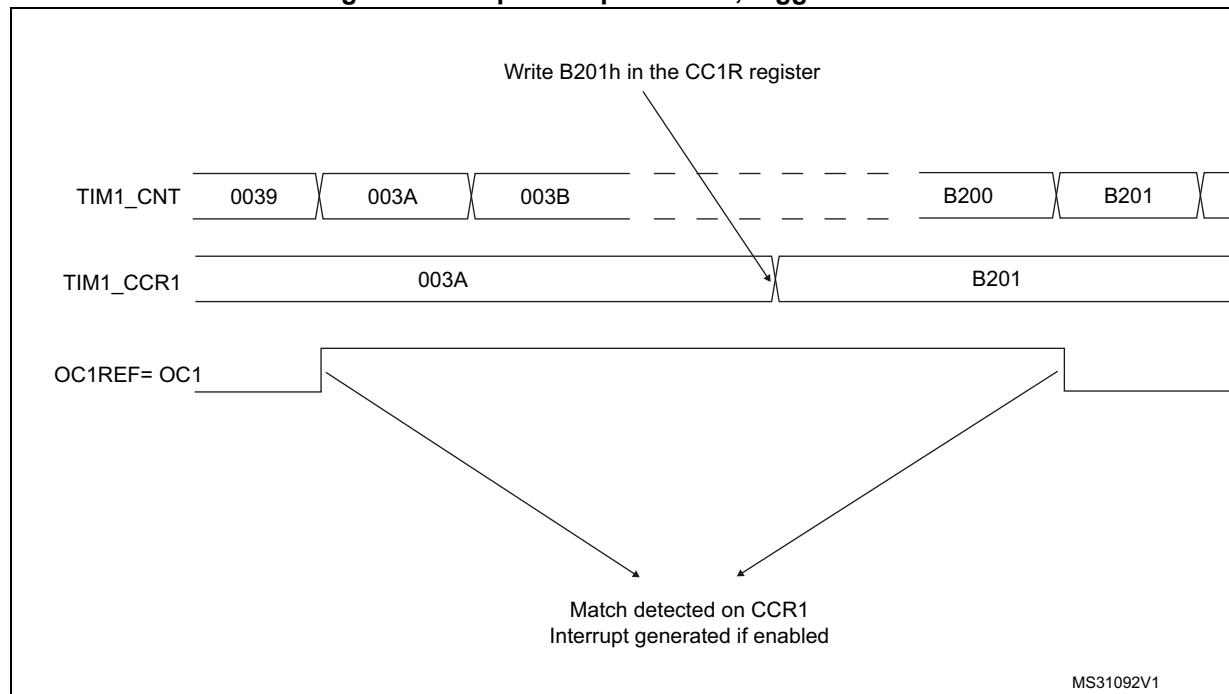
**Procedure:**

1. Select the counter clock (internal, external, prescaler).
2. Write the desired data in the TIMx\_ARR and TIMx\_CCRx registers.
3. Set the CCxIE bit if an interrupt request is to be generated.
4. Select the output mode. For example:
  - Write OCxM = 011 to toggle OCx output pin when CNT matches CCRx
  - Write OCxPE = 0 to disable preload register
  - Write CCxP = 0 to select active high polarity
  - Write CCxE = 1 to enable the output
5. Enable the counter by setting the CEN bit in the TIMx\_CR1 register.

For code example refer to the Appendix section [A.9.7: Output compare configuration code example](#).

The TIMx\_CCRx register can be updated at any time by software to control the output waveform, provided that the preload register is not enabled (OCxPE='0', else TIMx\_CCRx shadow register is updated only at the next update event UEV). An example is given in [Figure 94](#).

**Figure 94. Output compare mode, toggle on OC1**



### 17.3.10 PWM mode

Pulse Width Modulation mode allows to generate a signal with a frequency determined by the value of the TIMx\_ARR register and a duty cycle determined by the value of the TIMx\_CCRx register.

The PWM mode can be selected independently on each channel (one PWM per OCx output) by writing '110' (PWM mode 1) or '111' (PWM mode 2) in the OCxM bits in the TIMx\_CCMRx register. The corresponding preload register must be enabled by setting the

OCxPE bit in the TIMx\_CCMRx register, and eventually the auto-reload preload register (in upcounting or center-aligned modes) by setting the ARPE bit in the TIMx\_CR1 register.

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

OCx polarity is software programmable using the CCxP bit in the TIMx\_CCER register. It can be programmed as active high or active low. OCx output is enabled by a combination of the CCxE, CCxNE, MOE, OSS1 and OSSR bits (TIMx\_CCER and TIMx\_BDTR registers). Refer to the TIMx\_CCER register description for more details.

In PWM mode (1 or 2), TIMx\_CNT and TIMx\_CCRx are always compared to determine whether  $\text{TIMx\_CCR}_x \leq \text{TIMx\_CNT}$  or  $\text{TIMx\_CNT} \leq \text{TIMx\_CCR}_x$  (depending on the direction of the counter).

The timer is able to generate PWM in edge-aligned mode or center-aligned mode depending on the CMS bits in the TIMx\_CR1 register.

### PWM edge-aligned mode

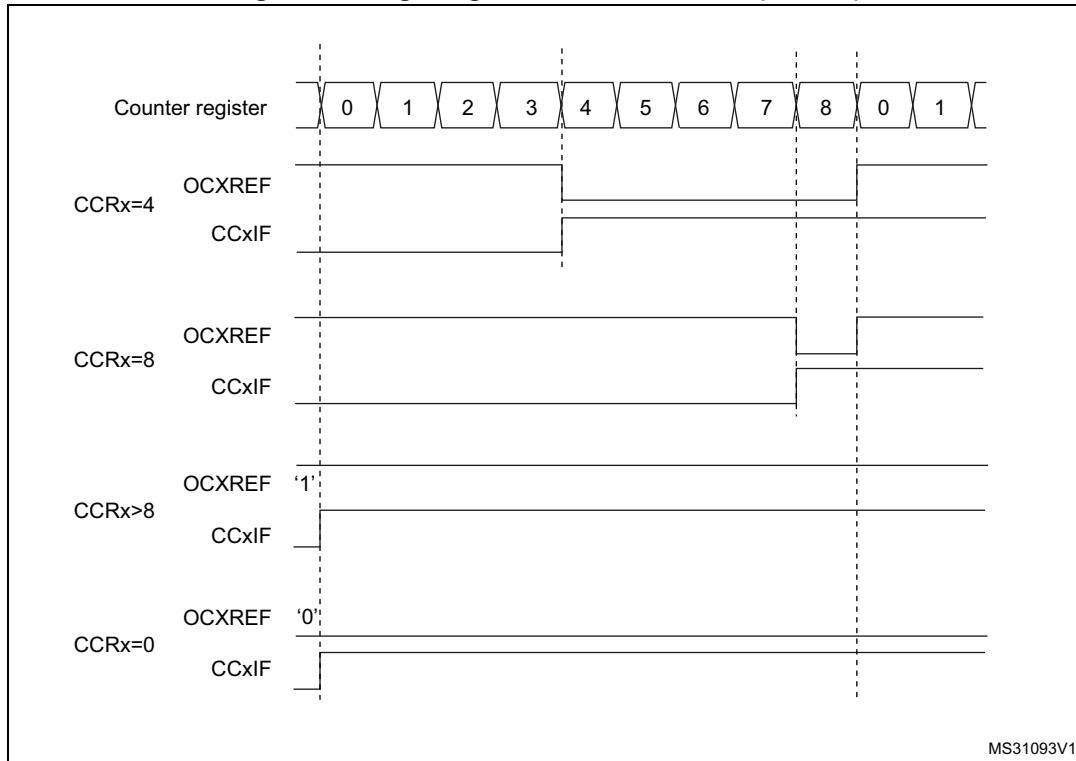
- Upcounting configuration

Upcounting is active when the DIR bit in the TIMx\_CR1 register is low. Refer to the [Upcounting mode on page 332](#).

In the following example, we consider PWM mode 1. The reference PWM signal OCxREF is high as long as  $\text{TIMx\_CNT} < \text{TIMx\_CCR}_x$  else it becomes low. If the compare value in TIMx\_CCRx is greater than the auto-reload value (in TIMx\_ARR) then OCxREF is held at '1'. If the compare value is 0 then OCxRef is held at '0'.

[Figure 95](#) shows some edge-aligned PWM waveforms in an example where TIMx\_ARR=8.

Figure 95. Edge-aligned PWM waveforms (ARR=8)



For code example refer to the Appendix section [A.9.8: Edge-aligned PWM configuration example](#).

- Downcounting configuration

Downcounting is active when DIR bit in TIMx\_CR1 register is high. Refer to the [Downcounting mode on page 336](#)

In PWM mode 1, the reference signal OCxRef is low as long as  $\text{TIMx\_CNT} > \text{TIMx\_CCRx}$  else it becomes high. If the compare value in TIMx\_CCRx is greater than the auto-reload value in TIMx\_ARR, then OCxREF is held at '1'. 0% PWM is not possible in this mode.

### PWM center-aligned mode

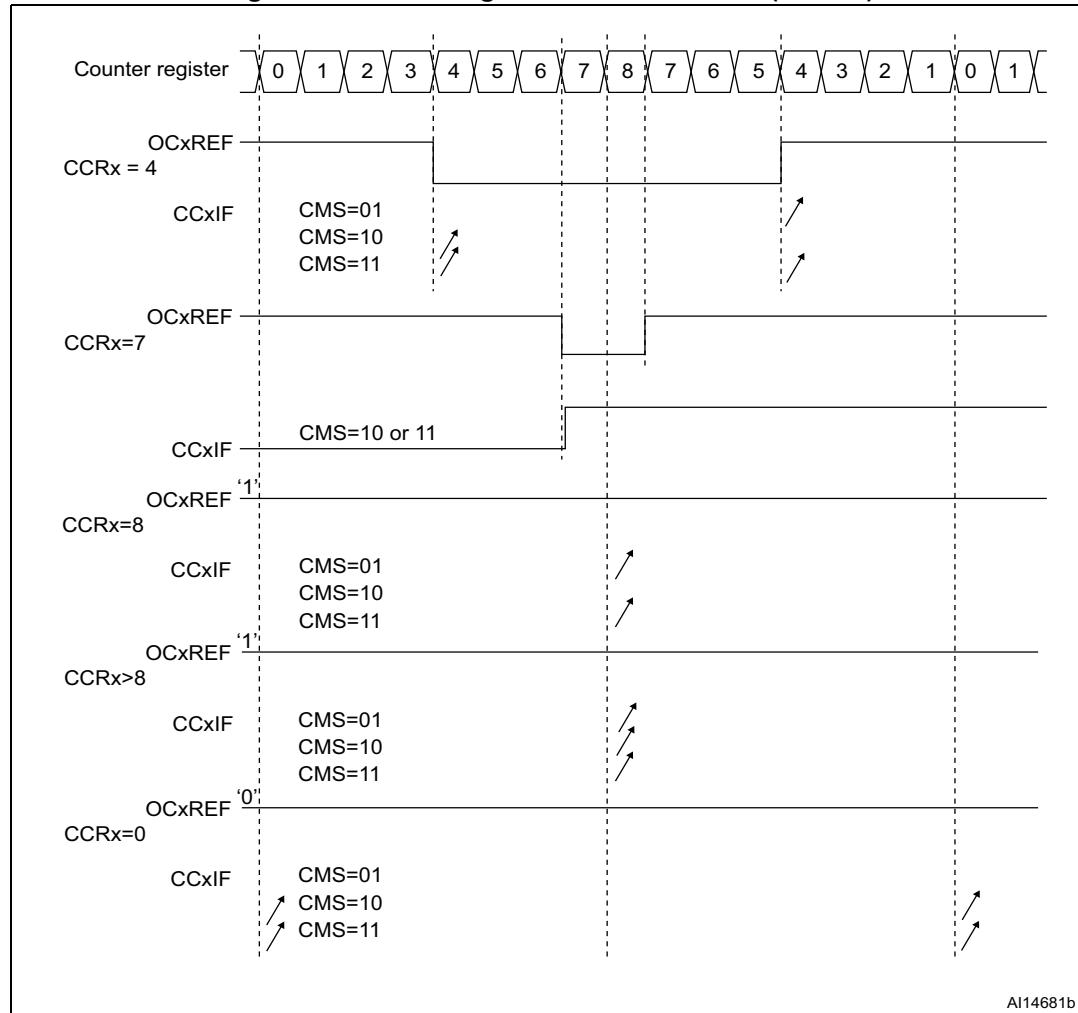
Center-aligned mode is active when the CMS bits in TIMx\_CR1 register are different from '00' (all the remaining configurations having the same effect on the OCxRef/OCx signals). The compare flag is set when the counter counts up, when it counts down or both when it counts up and down depending on the CMS bits configuration. The direction bit (DIR) in the TIMx\_CR1 register is updated by hardware and must not be changed by software. Refer to the [Center-aligned mode \(up/down counting\) on page 338](#).

*Figure 96* shows some center-aligned PWM waveforms in an example where:

- TIMx\_ARR=8,
- PWM mode is the PWM mode 1,
- The flag is set when the counter counts down corresponding to the center-aligned mode 1 selected for CMS=01 in TIMx\_CR1 register.

For code example refer to the Appendix section [A.9.9: Center-aligned PWM configuration example](#).

**Figure 96. Center-aligned PWM waveforms (ARR=8)**



Hints on using center-aligned mode:

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

### 17.3.11 Complementary outputs and dead-time insertion

The advanced-control timers (TIM1) can output two complementary signals and manage the switching-off and the switching-on instants of the outputs.

This time is generally known as dead-time and it has to be adjusted depending on the devices that are connected to the outputs and their characteristics (intrinsic delays of level-shifters, delays due to power switches...)

The polarity of the outputs (main output OC<sub>x</sub> or complementary OC<sub>xN</sub>) can be selected independently for each output. This is done by writing to the CC<sub>xP</sub> and CC<sub>xNP</sub> bits in the TIMx\_CCER register.

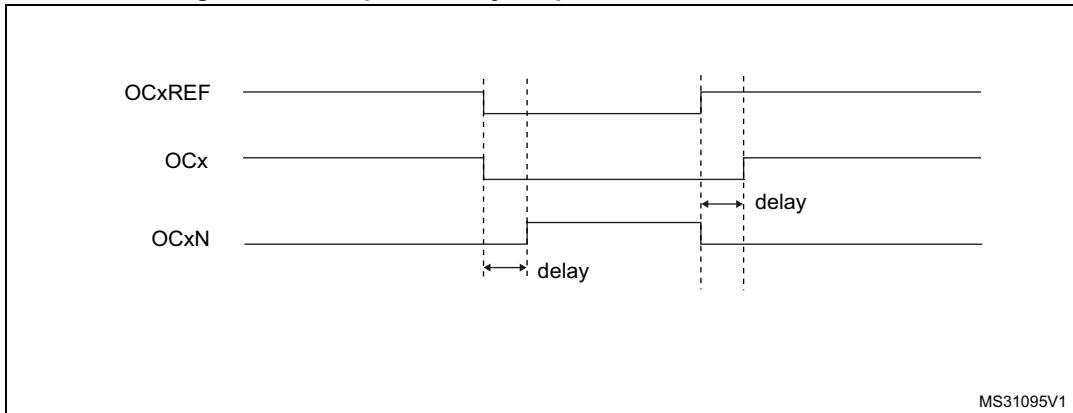
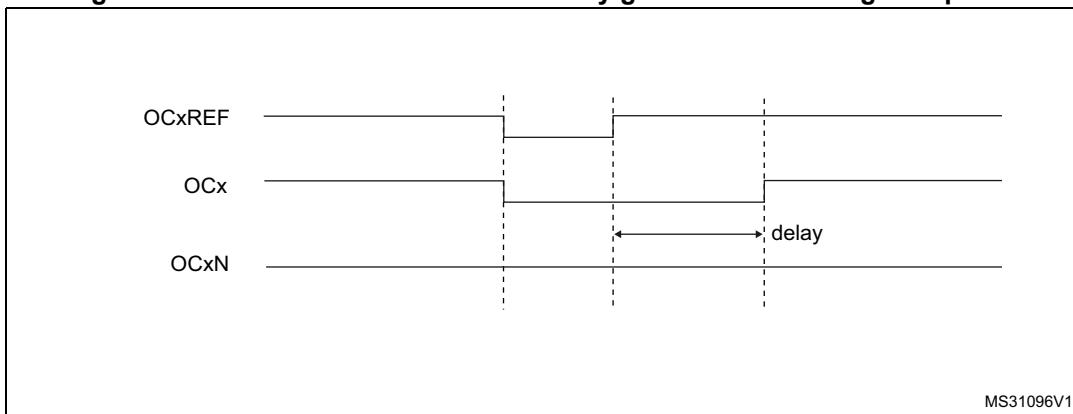
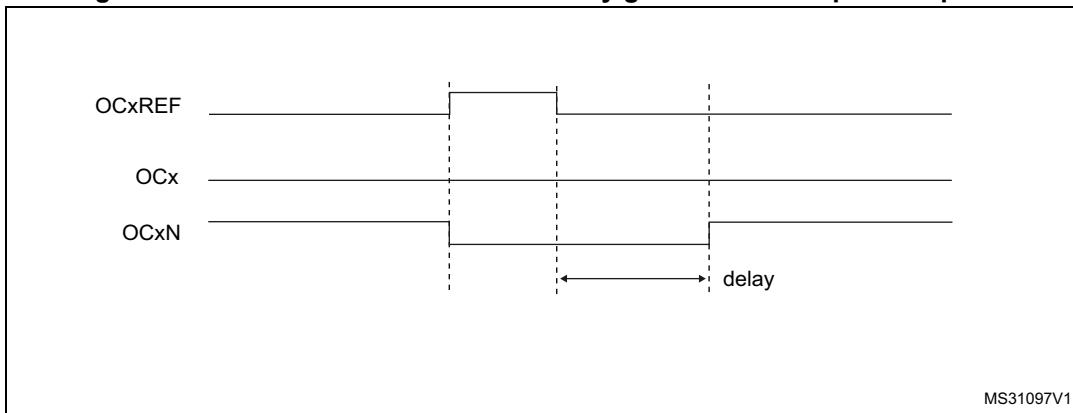
The complementary signals OC<sub>x</sub> and OC<sub>xN</sub> are activated by a combination of several control bits: the CC<sub>xE</sub> and CC<sub>xNE</sub> bits in the TIMx\_CCER register and the MOE, OIS<sub>x</sub>, OIS<sub>xN</sub>, OSS<sub>I</sub> and OSS<sub>R</sub> bits in the TIMx\_BDTR and TIMx\_CR2 registers. Refer to [Table 63: Output control bits for complementary OC<sub>x</sub> and OC<sub>xN</sub> channels with break feature on page 393](#) for more details. In particular, the dead-time is activated when switching to the IDLE state (MOE falling down to 0).

Dead-time insertion is enabled by setting both CC<sub>xE</sub> and CC<sub>xNE</sub> bits, and the MOE bit if the break circuit is present. There is one 10-bit dead-time generator for each channel. From a reference waveform OC<sub>xREF</sub>, it generates 2 outputs OC<sub>x</sub> and OC<sub>xN</sub>. If OC<sub>x</sub> and OC<sub>xN</sub> are active high:

- The OC<sub>x</sub> 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<sub>xN</sub> 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<sub>x</sub> or OC<sub>xN</sub>) then the corresponding pulse is not generated.

The following figures show the relationships between the output signals of the dead-time generator and the reference signal OC<sub>xREF</sub>. (we suppose CC<sub>xP</sub>=0, CC<sub>xNP</sub>=0, MOE=1, CC<sub>xE</sub>=1 and CC<sub>xNE</sub>=1 in these examples)

**Figure 97. Complementary output with dead-time insertion.****Figure 98. Dead-time waveforms with delay greater than the negative pulse.****Figure 99. Dead-time waveforms with delay greater than the positive pulse.**

The dead-time delay is the same for each of the channels and is programmable with the DTG bits in the TIMx\_BDTR register. Refer to [Section 17.4.18: TIM1 break and dead-time register \(TIM1\\_BDTR\) on page 398](#) for delay calculation.

### Re-directing OCxREF to OCx or OCxN

In output mode (forced, output compare or PWM), OCxREF can be re-directed to the OCx output or to OCxN output by configuring the CCxE and CCxNE bits in the TIMx\_CCER register.

This allows to send a specific waveform (such as PWM or static active level) on one output while the complementary remains at its inactive level. Other alternative possibilities are to have both outputs at inactive level or both outputs active and complementary with dead-time.

**Note:**

*When only OCxN is enabled (CCxE=0, CCxNE=1), it is not complemented and becomes active as soon as OCxREF is high. For example, if CCxNP=0 then OCxN=OCxRef. On the other hand, when both OCx and OCxN are enabled (CCxE=CCxNE=1) OCx becomes active when OCxREF is high whereas OCxN is complemented and becomes active when OCxREF is low.*

### 17.3.12 Using the break function

When using the break function, the output enable signals and inactive levels are modified according to additional control bits (MOE, OSS1 and OSSR bits in the TIMx\_BDTR register, OISx and OISxN bits in the TIMx\_CR2 register). In any case, the OCx and OCxN outputs cannot be set both to active level at a given time. Refer to [Table 63: Output control bits for complementary OCx and OCxN channels with break feature on page 393](#) for more details.

The source for break (BRK) channel can be an external source connected to the BKIN pin or one of the following internal sources:

- the core LOCKUP output
- the PVD output
- the SRAM parity error signal
- a clock failure event generated by the CSS detector
- the output from a comparator

When exiting from reset, the break circuit is disabled and the MOE bit is low. The break function can be enabled by setting the BKE bit in the TIMx\_BDTR register. The break input polarity can be selected by configuring the BKP bit in the same register. BKE and BKP can be modified at the same time. When the BKE and BKP bits are written, a delay of 1 APB clock cycle is applied before the writing is effective. Consequently, it is necessary to wait 1 APB clock period to correctly read back the bit after the write operation.

Because MOE falling edge can be asynchronous, a resynchronization circuit has been inserted between the actual signal (acting on the outputs) and the synchronous control bit (accessed in the TIMx\_BDTR register). It results in some delays between the asynchronous and the synchronous signals. In particular, if MOE is set to 1 whereas it was low, a delay must be inserted (dummy instruction) before reading it correctly. This is because the write acts on the asynchronous signal whereas the read reflects the synchronous signal.

When a break occurs (selected level on the break input):

- The MOE bit is cleared asynchronously, putting the outputs in inactive state, idle state or in reset state (selected by the OSS1 bit). This feature functions even if the MCU oscillator is off.
- Each output channel is driven with the level programmed in the OISx bit in the TIMx\_CR2 register as soon as MOE=0. If OSS1=0 then the timer releases the enable output else the enable output remains high.
- When complementary outputs are used:
  - The outputs are first put in reset state inactive state (depending on the polarity). This is done asynchronously so that it works even if no clock is provided to the timer.
  - If the timer clock is still present, then the dead-time generator is reactivated in order to drive the outputs with the level programmed in the OISx and OISxN bits after a dead-time. Even in this case, OCx and OCxN cannot be driven to their active level together. Note that because of the resynchronization on MOE, the dead-time duration is a bit longer than usual (around 2 ck\_tim clock cycles).
  - If OSS1=0 then the timer releases the enable outputs else the enable outputs remain or become high as soon as one of the CCxE or CCxNE bits is high.
- The break status flag (BIF bit in the TIMx\_SR register) is set. An interrupt can be generated if the BIE bit in the TIMx\_DIER register is set.
- If the AOE bit in the TIMx\_BDTR register is set, the MOE bit is automatically set again at the next update event UEV. This can be used to perform a regulation, for instance. Else, MOE remains low until it is written with 1 again. In this case, it can be used for security and the break input can be connected to an alarm from power drivers, thermal sensors or any security components.

Note:

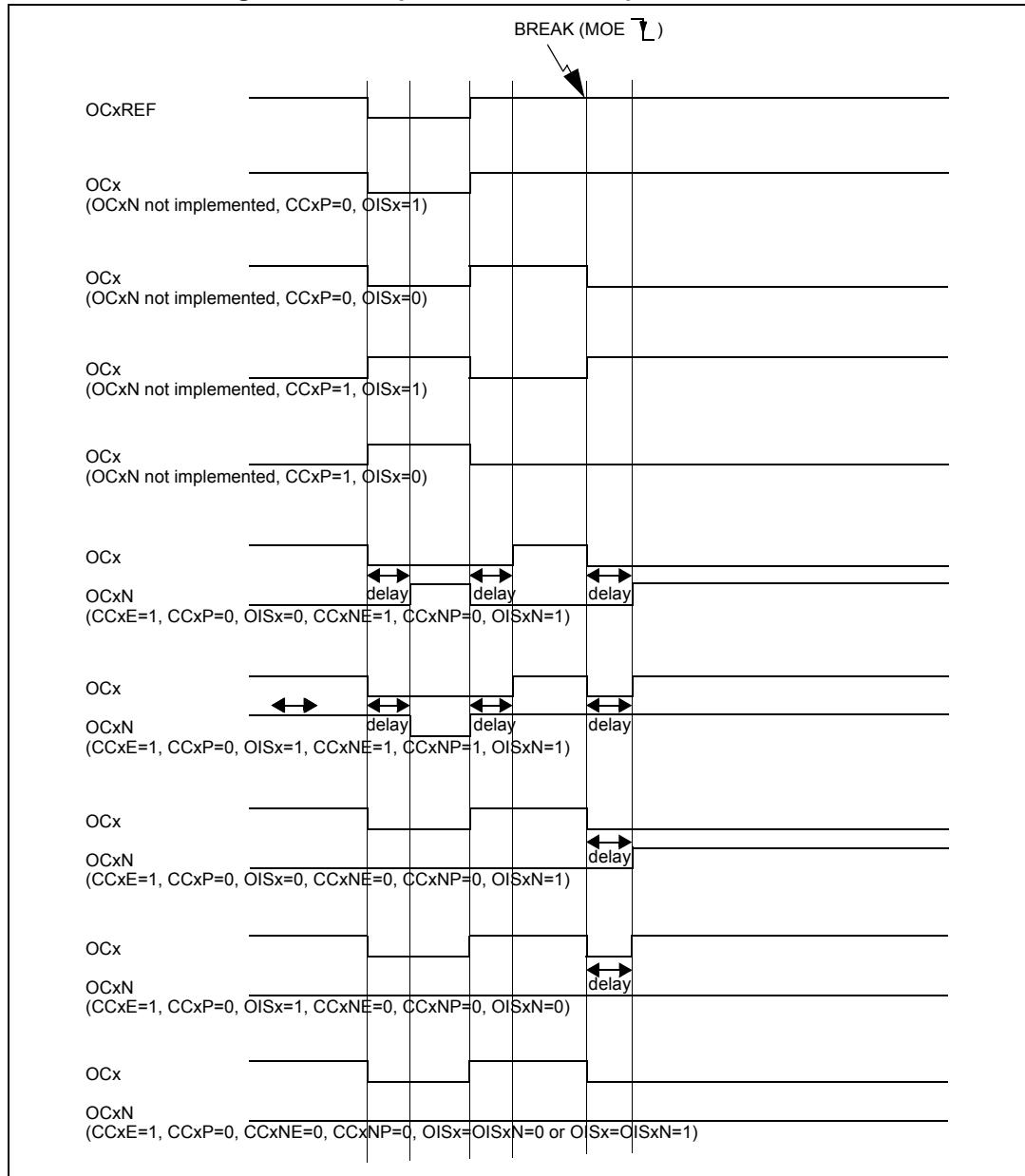
*The break inputs is acting on level. Thus, the MOE cannot be set while the break input is active (neither automatically nor by software). In the meantime, the status flag BIF cannot be cleared.*

The break can be generated by the BRK input which has a programmable polarity and an enable bit BKE in the TIMx\_BDTR Register.

In addition to the break input and the output management, a write protection has been implemented inside the break circuit to safeguard the application. It allows to freeze the configuration of several parameters (dead-time duration, OCx/OCxN polarities and state when disabled, OCxM configurations, break enable and polarity). The protection can be selected among 3 levels with the LOCK bits in the TIMx\_BDTR register. Refer to [Section 17.4.18: TIM1 break and dead-time register \(TIM1\\_BDTR\) on page 398](#). The LOCK bits can be written only once after an MCU reset.

The [Figure 100](#) shows an example of behavior of the outputs in response to a break.

Figure 100. Output behavior in response to a break



### 17.3.13 Clearing the OCxREF signal on an external event

The OCxREF signal of a given channel can be cleared when a high level is applied on the OCREF\_CLR\_INPUT (OCxCE enable bit in the corresponding TIMx\_CCMRx register set to '1'). OCxREF remains low until the next update event (UEV) occurs. This function can only be used in Output compare and PWM modes. It does not work in Forced mode.

OCREF\_CLR\_INPUT can be selected between the OCREF\_CLR input and ETRF (ETR after the filter) by configuring the OCCS bit in the TIMx\_SMCR register.

When ETRF is chosen, ETR must be configured as follows:

The OCxREF signal for a given channel can be driven Low by applying a High level to the ETRF input (OCxCE enable bit of the corresponding TIMx\_CCMRx register set to '1'). The OCxREF signal remains Low until the next update event, UEV, occurs.

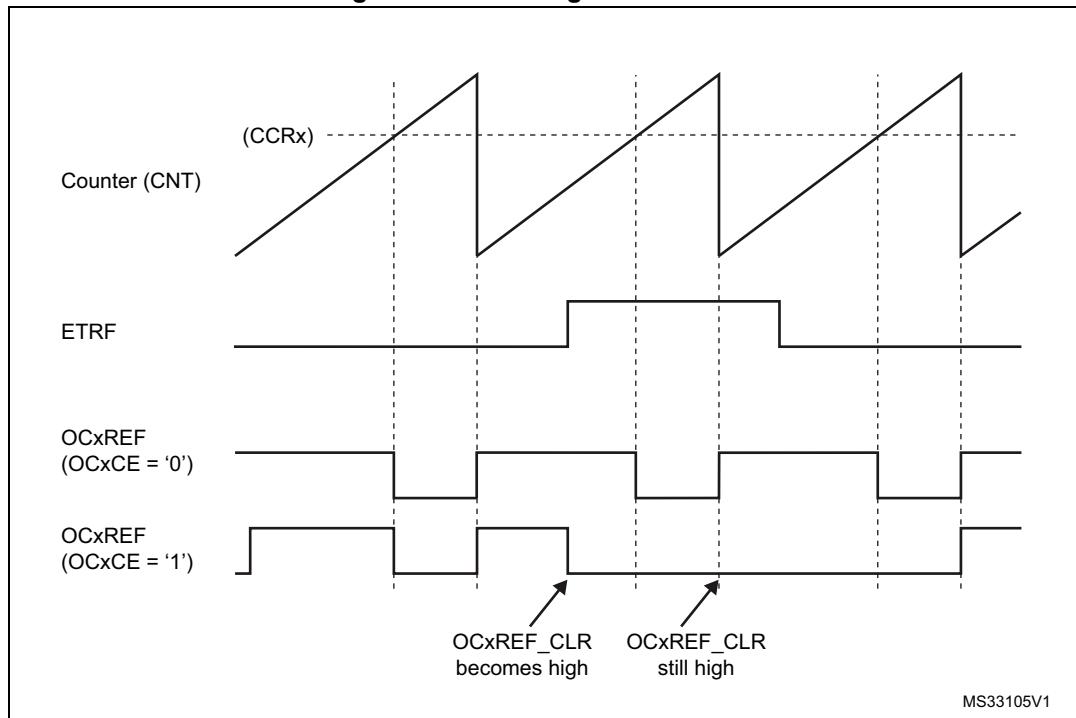
This function can only be used in output compare and PWM modes, and does not work in forced mode.

For example, the OCxREF signal can be connected to the output of a comparator to be used for current handling. In this case, the ETR must be configured as follow:

1. The External Trigger Prescaler should be kept off: bits ETPS[1:0] of the TIMx\_SMCR register set to '00'.
2. The external clock mode 2 must be disabled: bit ECE of the TIMx\_SMCR register set to '0'.
3. The External Trigger Polarity (ETP) and the External Trigger Filter (ETF) can be configured according to the user needs.

For code example refer to the Appendix section [A.9.10: ETR configuration to clear OCxREF code example](#).

*Figure 101* shows the behavior of the OCxREF signal when the ETRF Input becomes High, for both values of the enable bit OCxCE. In this example, the timer TIMx is programmed in PWM mode.

**Figure 101. Clearing TIMx OCxREF**

MS33105V1

**Note:** In case of a PWM with a 100% duty cycle (if CCR<sub>x</sub>>ARR), then OCxREF is enabled again at the next counter overflow.

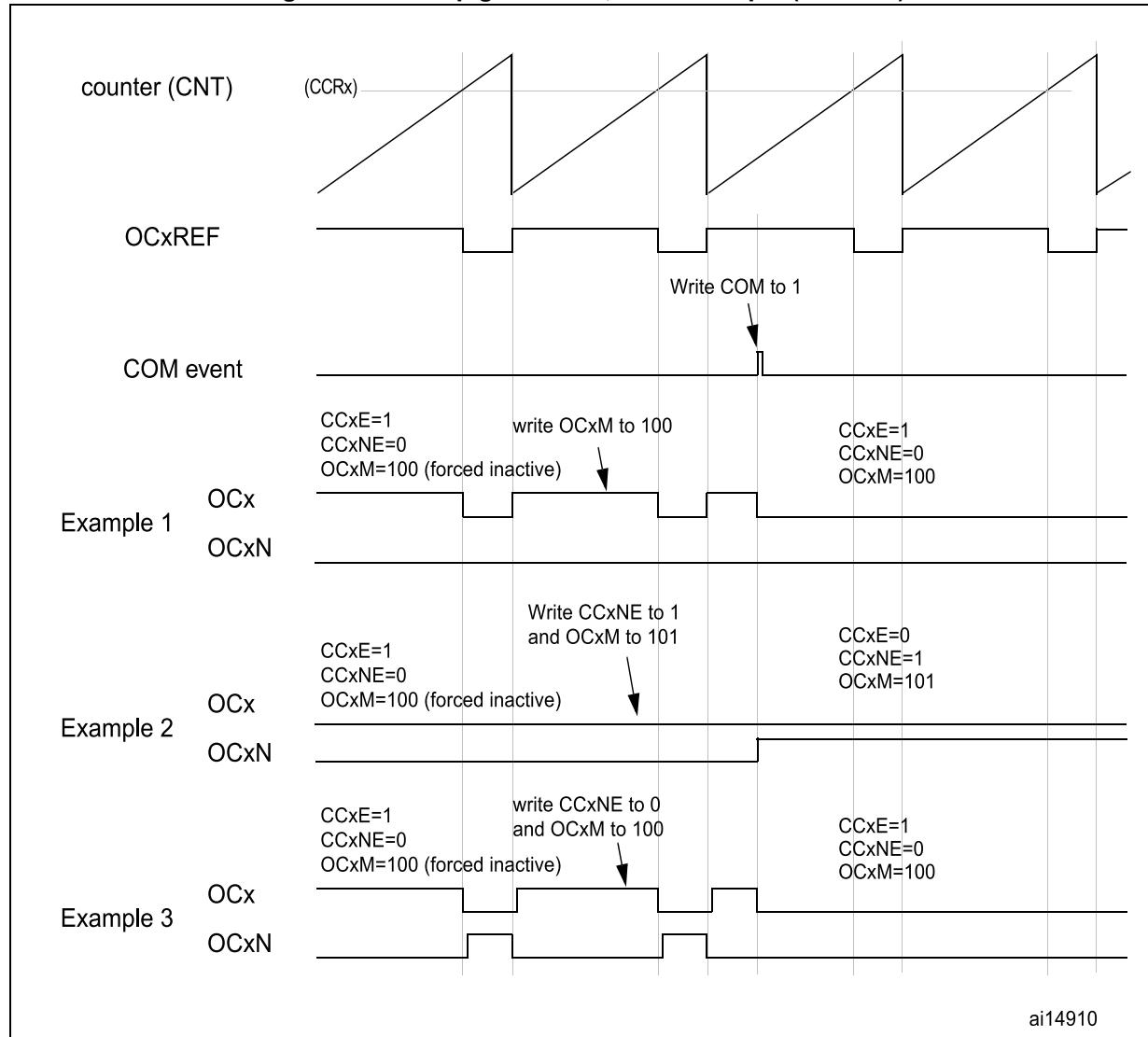
### 17.3.14 6-step PWM generation

When complementary outputs are used on a channel, preload bits are available on the OCxM, CCxE and CCxNE bits. The preload bits are transferred to the shadow bits at the COM commutation event. Thus one can program in advance the configuration for the next step and change the configuration of all the channels at the same time. COM can be generated by software by setting the COM bit in the TIMx\_EGR register or by hardware (on TRGI rising edge).

A flag is set when the COM event occurs (COMIF bit in the TIMx\_SR register), which can generate an interrupt (if the COMIE bit is set in the TIMx\_DIER register) or a DMA request (if the COMDE bit is set in the TIMx\_DIER register).

The [Figure 102](#) describes the behavior of the OCx and OCxN outputs when a COM event occurs, in 3 different examples of programmed configurations.

**Figure 102. 6-step generation, COM example (OSSR=1)**



ai14910

### 17.3.15 One-pulse mode

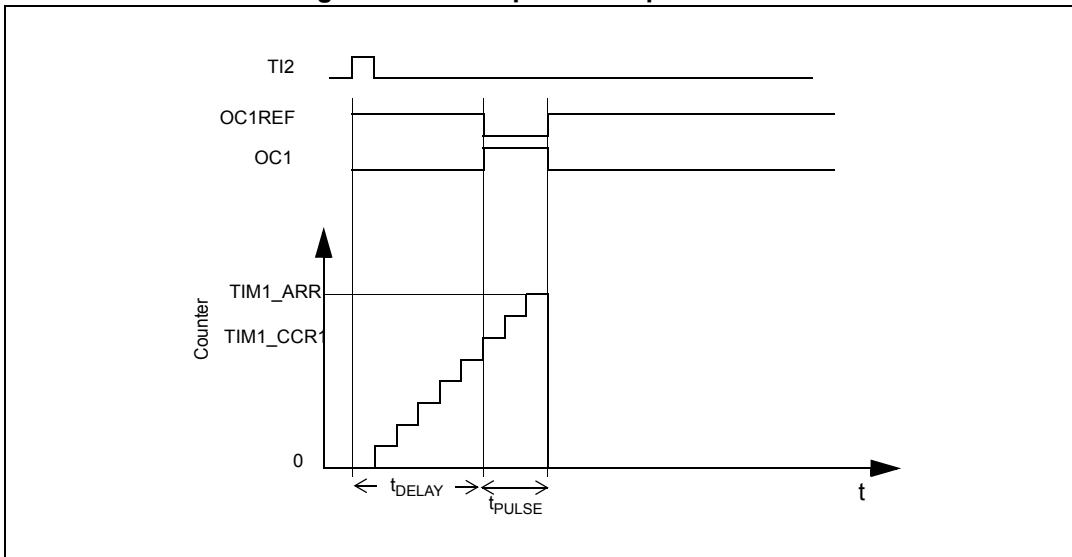
One-pulse mode (OPM) is a particular case of the previous modes. It allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length after a programmable delay.

Starting the counter can be controlled through the slave mode controller. Generating the waveform can be done in output compare mode or PWM mode. One-pulse mode is selected by setting the OPM bit in the TIMx\_CR1 register. This makes the counter stop automatically at the next update event UEV.

A pulse can be correctly generated only if the compare value is different from the counter initial value. Before starting (when the timer is waiting for the trigger), the configuration must be:

- In upcounting: CNT < CCRx  $\leq$  ARR (in particular, 0 < CCRx)
- In downcounting: CNT > CCRx

**Figure 103. Example of one pulse mode**



For example one may want to generate a positive pulse on OC1 with a length of  $t_{PULSE}$  and after a delay of  $t_{DELAY}$  as soon as a positive edge is detected on the TI2 input pin.

Let's use TI2FP2 as trigger 1:

- Map TI2FP2 to TI2 by writing CC2S='01' in the TIMx\_CCMR1 register.
- TI2FP2 must detect a rising edge, write CC2P='0' and CC2NP='0' in the TIMx\_CCER register.
- Configure TI2FP2 as trigger for the slave mode controller (TRGI) by writing TS='110' in the TIMx\_SMCR register.
- TI2FP2 is used to start the counter by writing SMS to '110' in the TIMx\_SMCR register (trigger mode).

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

- The  $t_{DELAY}$  is defined by the value written in the TIMx\_CCR1 register.
- The  $t_{PULSE}$  is defined by the difference between the auto-reload value and the compare value ( $\text{TIMx\_ARR} - \text{TIMx\_CCR1} + 1$ ).
- Let's say one want to build a waveform with a transition from '0' to '1' when a compare match occurs and a transition from '1' to '0' when the counter reaches the auto-reload value. To do this PWM mode 2 must be enabled by writing  $\text{OC1M}=111$  in the TIMx\_CCMR1 register. Optionally the preload registers can be enabled by writing  $\text{OC1PE}'1'$  in the TIMx\_CCMR1 register and ARPE in the TIMx\_CR1 register. In this case one has to write the compare value in the TIMx\_CCR1 register, the auto-reload value in the TIMx\_ARR register, generate an update by setting the UG bit and wait for external trigger event on TI2. CC1P is written to '0' in this example.

In our example, the DIR and CMS bits in the TIMx\_CR1 register should be low.

For code example refer to the Appendix section [A.9.16: One-Pulse mode code example](#).

Since only 1 pulse (Single mode) is needed, a 1 must be written in the OPM bit in the TIMx\_CR1 register to stop the counter at the next update event (when the counter rolls over from the auto-reload value back to 0). When OPM bit in the TIMx\_CR1 register is set to '0', so the Repetitive Mode is selected.

#### Particular case: OCx fast enable

In One-pulse mode, the edge detection on  $\text{TI}_x$  input set the CEN bit which enables the counter. Then the comparison between the counter and the compare value makes the output toggle. But several clock cycles are needed for these operations and it limits the minimum delay  $t_{DELAY}$  min we can get.

If one wants to output a waveform with the minimum delay, the OCxFE bit can be set in the TIMx\_CCMRx register. Then OCxRef (and OCx) are forced in response to the stimulus, without taking in account the comparison. Its new level is the same as if a compare match had occurred. OCxFE acts only if the channel is configured in PWM1 or PWM2 mode.

For code example refer to the part of code, conditioned by  $\text{PULSE\_WITHOUT\_DELAY} > 0$  in the Appendix section [A.9.16: One-Pulse mode code example](#).

### 17.3.16 Encoder interface mode

To select Encoder Interface mode write SMS='001' in the TIMx\_SMCR register if the counter is counting on  $\text{TI}_2$  edges only, SMS='010' if it is counting on  $\text{TI}_1$  edges only and SMS='011' if it is counting on both  $\text{TI}_1$  and  $\text{TI}_2$  edges.

Select the  $\text{TI}_1$  and  $\text{TI}_2$  polarity by programming the CC1P and CC2P bits in the TIMx\_CCER register. When needed, the input filter can be programmed as well. CC1NP and CC2NP must be kept low.

The two inputs  $\text{TI}_1$  and  $\text{TI}_2$  are used to interface to an incremental encoder. Refer to [Table 61](#). The counter is clocked by each valid transition on  $\text{TI}_1\text{FP}_1$  or  $\text{TI}_2\text{FP}_2$  ( $\text{TI}_1$  and  $\text{TI}_2$  after input filter and polarity selection,  $\text{TI}_1\text{FP}_1=\text{TI}_1$  if not filtered and not inverted,  $\text{TI}_2\text{FP}_2=\text{TI}_2$  if not filtered and not inverted) assuming that it is enabled (CEN bit in TIMx\_CR1 register written to '1'). The sequence of transitions of the two inputs is evaluated and generates count pulses as well as the direction signal. Depending on the sequence the counter counts up or down, the DIR bit in the TIMx\_CR1 register is modified by hardware

accordingly. The DIR bit is calculated at each transition on any input (TI1 or TI2), whatever the counter is counting on TI1 only, TI2 only or both TI1 and TI2.

Encoder interface mode acts simply as an external clock with direction selection. This means that the counter just counts continuously between 0 and the auto-reload value in the TIMx\_ARR register (0 to ARR or ARR down to 0 depending on the direction). So the TIMx\_ARR must be configured before starting. In the same way, the capture, compare, prescaler, repetition counter, trigger output features continue to work as normal. Encoder mode and External clock mode 2 are not compatible and must not be selected together.

In this mode, the counter is modified automatically following the speed and the direction of the incremental encoder and its content, therefore, always represents the encoder's position. The count direction correspond to the rotation direction of the connected sensor. The table summarizes the possible combinations, assuming TI1 and TI2 do not switch at the same time.

**Table 61. Counting direction versus encoder signals**

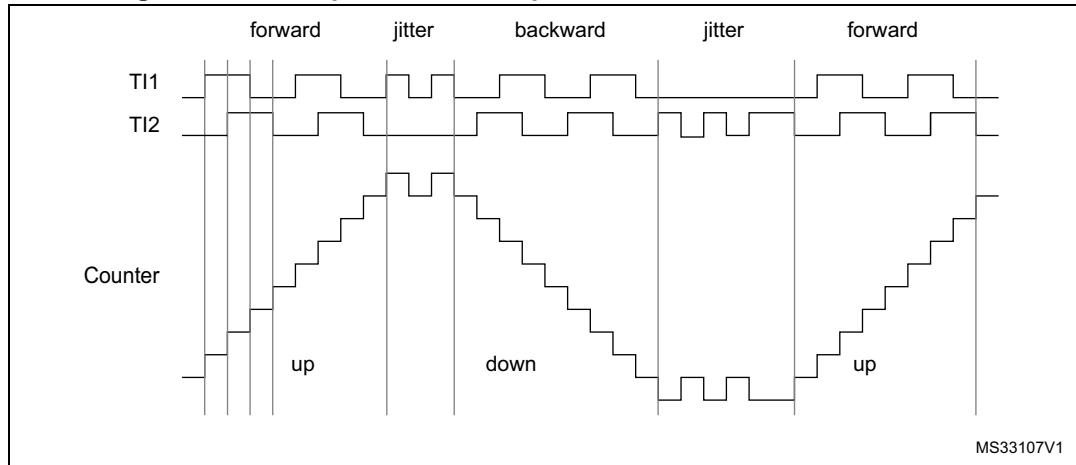
Active edge	Level on opposite signal (TI1FP1 for TI2, TI2FP2 for TI1)	TI1FP1 signal		TI2FP2 signal	
		Rising	Falling	Rising	Falling
Counting on TI1 only	High	Down	Up	No Count	No Count
	Low	Up	Down	No Count	No Count
Counting on TI2 only	High	No Count	No Count	Up	Down
	Low	No Count	No Count	Down	Up
Counting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

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

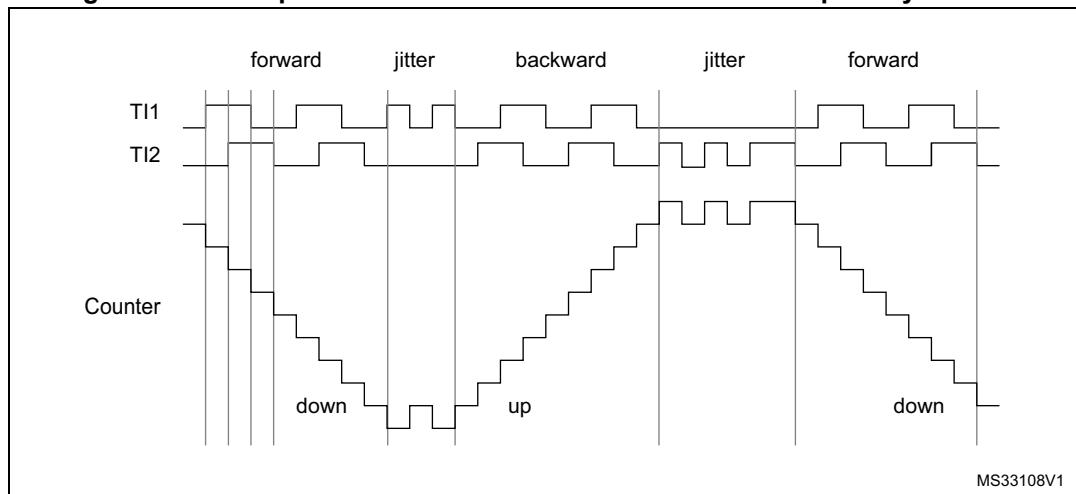
*Figure 104* gives an example of counter operation, showing count signal generation and direction control. It also shows how input jitter is compensated where both edges are selected. This might occur if the sensor is positioned near to one of the switching points. For this example we assume that the configuration is the following:

- CC1S='01' (TIMx\_CCMR1 register, TI1FP1 mapped on TI1).
- CC2S='01' (TIMx\_CCMR2 register, TI1FP2 mapped on TI2).
- CC1P='0' (TIMx\_CCER register, TI1FP1 non-inverted, TI1FP1=TI1).
- CC2P='0' (TIMx\_CCER register, TI1FP2 non-inverted, TI1FP2= TI2).
- SMS='011' (TIMx\_SMCR register, both inputs are active on both rising and falling edges).
- CEN='1' (TIMx\_CR1 register, Counter enabled).

For code example refer to the Appendix section [A.9.11: Encoder interface code example](#).

**Figure 104. Example of counter operation in encoder interface mode.**

*Figure 105* gives an example of counter behavior when TI1FP1 polarity is inverted (same configuration as above except CC1P='1').

**Figure 105. Example of encoder interface mode with TI1FP1 polarity inverted.**

The timer, when configured in Encoder Interface mode provides information on the sensor's current position. Dynamic information can be obtained (speed, acceleration, deceleration) by measuring the period between two encoder events using a second timer configured in capture mode. The output of the encoder which indicates the mechanical zero can be used for this purpose. Depending on the time between two events, the counter can also be read at regular times. This can be done by latching the counter value into a third input capture register if available (then the capture signal must be periodic and can be generated by another timer). When available, it is also possible to read its value through a DMA request generated by a real-time clock.

### 17.3.17 Timer input XOR function

The TI1S bit in the TIMx\_CR2 register, allows the input filter of channel 1 to be connected to the output of a XOR gate, combining the three input pins TIMx\_CH1, TIMx\_CH2 and TIMx\_CH3.

The XOR output can be used with all the timer input functions such as trigger or input capture. An example of this feature used to interface Hall sensors is given in [Section 17.3.18](#) below.

### 17.3.18 Interfacing with Hall sensors

This is done using the advanced-control timers (TIM1) to generate PWM signals to drive the motor and another timer TIMx (TIM2 or TIM3) referred to as “interfacing timer” in [Figure 106](#). The “interfacing timer” captures the 3 timer input pins (CC1, CC2, CC3) connected through a XOR to the TI1 input channel (selected by setting the TI1S bit in the TIMx\_CR2 register).

The slave mode controller is configured in reset mode; the slave input is TI1F\_ED. Thus, each time one of the 3 inputs toggles, the counter restarts counting from 0. This creates a time base triggered by any change on the Hall inputs.

On the “interfacing timer”, capture/compare channel 1 is configured in capture mode, capture signal is TRC (See [Figure 89: Capture/compare channel \(example: channel 1 input stage\) on page 348](#)). The captured value, which corresponds to the time elapsed between 2 changes on the inputs, gives information about motor speed.

The “interfacing timer” can be used in output mode to generate a pulse which changes the configuration of the channels of the advanced-control timer (TIM1) (by triggering a COM event). The TIM1 timer is used to generate PWM signals to drive the motor. To do this, the interfacing timer channel must be programmed so that a positive pulse is generated after a programmed delay (in output compare or PWM mode). This pulse is sent to the advanced-control timer (TIM1) through the TRGO output.

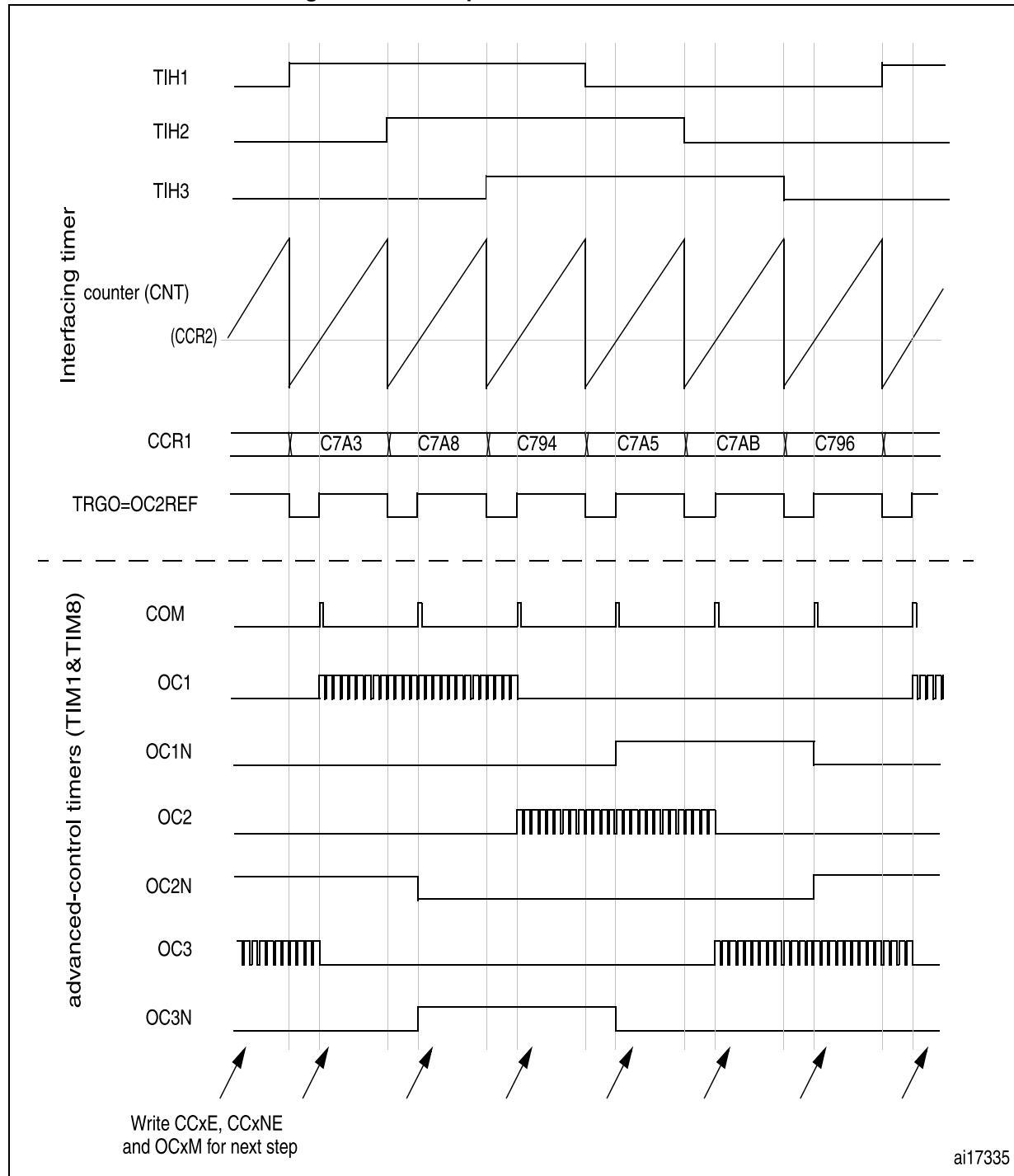
Example: one wants to change the PWM configuration of the advanced-control timer TIM1 after a programmed delay each time a change occurs on the Hall inputs connected to one of the TIMx timers.

- Configure 3 timer inputs XORed to the TI1 input channel by writing the TI1S bit in the TIMx\_CR2 register to ‘1’,
- Program the time base: write the TIMx\_ARR to the max value (the counter must be cleared by the TI1 change. Set the prescaler to get a maximum counter period longer than the time between 2 changes on the sensors,
- Program channel 1 in capture mode (TRC selected): write the CC1S bits in the TIMx\_CCMR1 register to ‘01’. The digital filter can also be programmed if needed,
- Program channel 2 in PWM 2 mode with the desired delay: write the OC2M bits to ‘111’ and the CC2S bits to ‘00’ in the TIMx\_CCMR1 register,
- Select OC2REF as trigger output on TRGO: write the MMS bits in the TIMx\_CR2 register to ‘101’,

In the advanced-control timer TIM1, the right ITR input must be selected as trigger input, the timer is programmed to generate PWM signals, the capture/compare control signals are preloaded (CCPC=1 in the TIMx\_CR2 register) and the COM event is controlled by the trigger input (CCUS=1 in the TIMx\_CR2 register). The PWM control bits (CCxE, OCxM) are written after a COM event for the next step (this can be done in an interrupt subroutine generated by the rising edge of OC2REF).

[Figure 106](#) describes this example.

**Figure 106. Example of hall sensor interface**



### 17.3.19 TIMx and external trigger synchronization

The TIMx timer can be synchronized with an external trigger in several modes: Reset mode, Gated mode and Trigger mode.

#### Slave mode: Reset mode

The counter and its prescaler can be reinitialized in response to an event on a trigger input. Moreover, if the URS bit from the TIMx\_CR1 register is low, an update event UEV is generated. Then all the preloaded registers (TIMx\_ARR, TIMx\_CCRx) are updated.

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

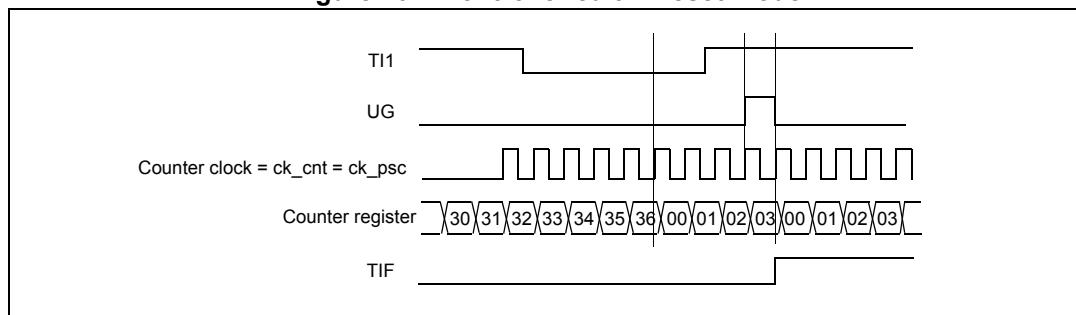
- Configure the channel 1 to detect rising edges on TI1. Configure the input filter duration (in this example, we do not need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. The CC1S bits select the input capture source only, CC1S = 01 in the TIMx\_CCMR1 register. Write CC1P=0 and CC1NP='0' in TIMx\_CCER register to validate the polarity (and detect rising edges only).
- Configure the timer in reset mode by writing SMS=100 in TIMx\_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx\_SMCR register.
- Start the counter by writing CEN=1 in the TIMx\_CR1 register.

For code example refer to the Appendix section [A.9.12: Reset mode code example](#).

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

The following figure shows this behavior when the auto-reload register TIMx\_ARR=0x36. The delay between the rising edge on TI1 and the actual reset of the counter is due to the resynchronization circuit on TI1 input.

**Figure 107. Control circuit in reset mode**



### Slave mode: Gated mode

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

In the following example, the upcounter counts only when TI1 input is low:

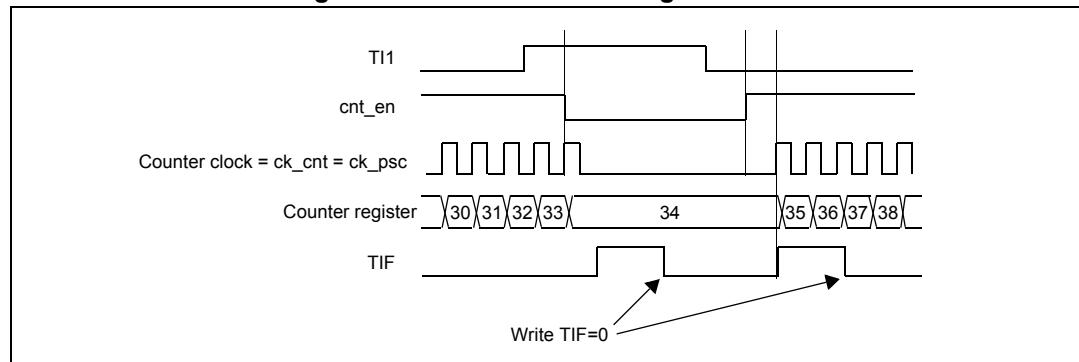
- Configure the channel 1 to detect low levels on TI1. Configure the input filter duration (in this example, we do not need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. The CC1S bits select the input capture source only, CC1S=01 in TIMx\_CCMR1 register. Write CC1P=1 and CC1NP='0' in TIMx\_CCER register to validate the polarity (and detect low level only).
- Configure the timer in gated mode by writing SMS=101 in TIMx\_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx\_SMCR register.
- Enable the counter by writing CEN=1 in the TIMx\_CR1 register (in gated mode, the counter doesn't start if CEN=0, whatever is the trigger input level).

For code example refer to the Appendix section [A.9.13: Gated mode code example](#).

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

The delay between the rising edge on TI1 and the actual stop of the counter is due to the resynchronization circuit on TI1 input.

**Figure 108. Control circuit in gated mode**



### Slave mode: Trigger mode

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

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

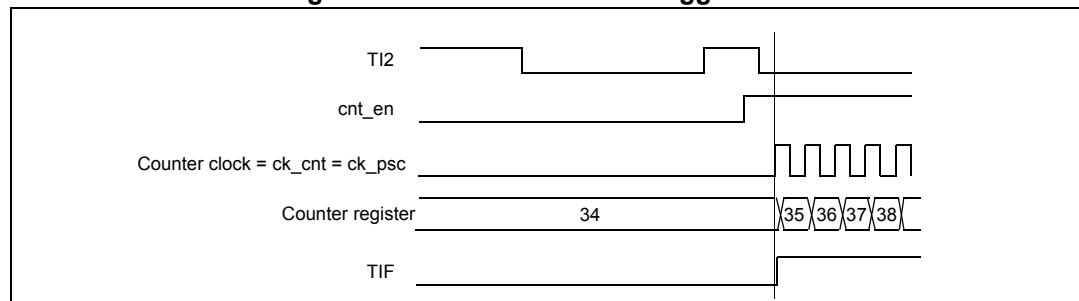
- Configure the channel 2 to detect rising edges on TI2. Configure the input filter duration (in this example, we do not need any filter, so we keep IC2F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. The CC2S bits are configured to select the input capture source only, CC2S=01 in TIMx\_CCMR1 register. Write CC2P=1 and CC2NP=0 in TIMx\_CCER register to validate the polarity (and detect low level only).
- Configure the timer in trigger mode by writing SMS=110 in TIMx\_SMCR register. Select TI2 as the input source by writing TS=110 in TIMx\_SMCR register.

For code example refer to the Appendix section [A.9.14: Trigger mode code example](#).

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

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

**Figure 109. Control circuit in trigger mode**



### Slave mode: external clock mode 2 + trigger mode

The external clock mode 2 can be used in addition to another slave mode (except external clock mode 1 and encoder mode). In this case, the ETR signal is used as external clock input, and another input can be selected as trigger input (in reset mode, gated mode or trigger mode). It is recommended not to select ETR as TRGI through the TS bits of TIMx\_SMCR register.

In the following example, the upcounter is incremented at each rising edge of the ETR signal as soon as a rising edge of TI1 occurs:

- Configure the external trigger input circuit by programming the TIMx\_SMCR register as follows:
  - ETF = 0000: no filter
  - ETPS=00: prescaler disabled
  - ETP=0: detection of rising edges on ETR and ECE=1 to enable the external clock mode 2.

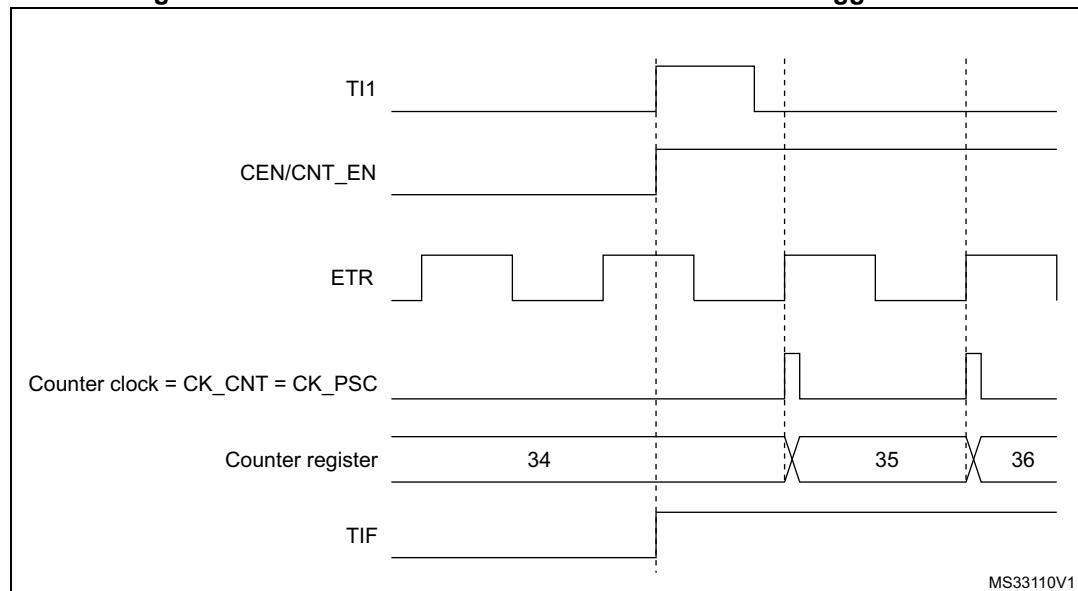
2. Configure the channel 1 as follows, to detect rising edges on TI:
  - IC1F=0000: no filter.
  - The capture prescaler is not used for triggering and does not need to be configured.
  - CC1S=01in TIMx\_CCMR1 register to select only the input capture source
  - CC1P=0 and CC1NP='0' in TIMx\_CCER register to validate the polarity (and detect rising edge only).
3. Configure the timer in trigger mode by writing SMS=110 in TIMx\_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx\_SMCR register.

For code example refer to the Appendix section [A.9.15: External clock mode 2 + trigger mode code example](#).

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

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

**Figure 110. Control circuit in external clock mode 2 + trigger mode**



### 17.3.20 Timer synchronization

The TIM timers are linked together internally for timer synchronization or chaining. Refer to [Section 18.3.15: Timer synchronization on page 439](#) for details.

### 17.3.21 Debug mode

When the microcontroller enters debug mode (Cortex™-M0 core halted), the TIMx counter either continues to work normally or stops, depending on DBG\_TIMx\_STOP configuration bit in DBG module.

## 17.4 TIM1 registers

Refer to [Section 1.2 on page 42](#) for a list of abbreviations used in register descriptions.

### 17.4.1 TIM1 control register 1 (TIM1\_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	CKD[1:0]		ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN

Bits 15:10 Reserved, must be kept at reset value.

Bits 9:8 **CKD[1:0]**: Clock division

This bit-field indicates the division ratio between the timer clock (CK\_INT) frequency and the dead-time and sampling clock ( $t_{DTS}$ ) used by the dead-time generators and the digital filters (ETR, TIx),

00:  $t_{DTS}=t_{CK\_INT}$

01:  $t_{DTS}=2*t_{CK\_INT}$

10:  $t_{DTS}=4*t_{CK\_INT}$

11: Reserved, do not program this value

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

0: TIMx\_ARR register is not buffered

1: TIMx\_ARR register is buffered

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

00: Edge-aligned mode. The counter counts up or down depending on the direction bit (DIR).

01: Center-aligned mode 1. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx\_CCMRx register) are set only when the counter is counting down.

10: Center-aligned mode 2. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx\_CCMRx register) are set only when the counter is counting up.

11: Center-aligned mode 3. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx\_CCMRx register) are set both when the counter is counting up or down.

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

Bit 4 **DIR**: Direction

0: Counter used as upcounter

1: Counter used as downcounter

*Note:* This bit is read only when the timer is configured in Center-aligned mode or Encoder mode.

Bit 3 **OPM**: One pulse mode

0: Counter is not stopped at update event

1: Counter stops counting at the next update event (clearing the bit CEN)

**Bit 2 URS:** Update request source

This bit is set and cleared by software to select the UEV event sources.

0: Any of the following events generate an update interrupt or DMA request if enabled.

These events can be:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

1: Only counter overflow/underflow generates an update interrupt or DMA request if enabled.

**Bit 1 UDIS:** Update disable

This bit is set and cleared by software to enable/disable UEV event generation.

0: UEV enabled. The Update (UEV) event is generated by one of the following events:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

Buffered registers are then loaded with their preload values.

1: UEV disabled. The Update event is not generated, shadow registers keep their value (ARR, PSC, CCRx). However the counter and the prescaler are reinitialized if the UG bit is set or if a hardware reset is received from the slave mode controller.

**Bit 0 CEN:** Counter enable

0: Counter disabled

1: Counter enabled

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

## 17.4.2 TIM1 control register 2 (TIM1\_CR2)

Address offset: 0x04

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1	TI1S	MMS[2:0]	CCDS	CCUS	Res.	CCPC		
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

Bit 15 Reserved, must be kept at reset value.

Bit 14 **OIS4:** Output Idle state 4 (OC4 output)

refer to OIS1 bit

Bit 13 **OIS3N:** Output Idle state 3 (OC3N output)

refer to OIS1N bit

Bit 12 **OIS3:** Output Idle state 3 (OC3 output)

refer to OIS1 bit

Bit 11 **OIS2N:** Output Idle state 2 (OC2N output)

refer to OIS1N bit

Bit 10 **OIS2:** Output Idle state 2 (OC2 output)

refer to OIS1 bit

Bit 9 **OIS1N**: Output Idle state 1 (OC1N output)

- 0: OC1N=0 after a dead-time when MOE=0
- 1: OC1N=1 after a dead-time when MOE=0

*Note: This bit cannot be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx\_BDTR register).*

Bit 8 **OIS1**: Output Idle state 1 (OC1 output)

- 0: OC1=0 (after a dead-time if OC1N is implemented) when MOE=0
- 1: OC1=1 (after a dead-time if OC1N is implemented) when MOE=0

*Note: This bit cannot be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx\_BDTR register).*

Bit 7 **TI1S**: TI1 selection

- 0: The TIMx\_CH1 pin is connected to TI1 input
- 1: The TIMx\_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

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

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

000: **Reset** - the UG bit from the TIMx\_EGR register is used as trigger output (TRGO). If the reset is generated by the trigger input (slave mode controller configured in reset mode) then the signal on TRGO is delayed compared to the actual reset.

001: **Enable** - the Counter Enable signal CNT\_EN is used as trigger output (TRGO). It is useful to start several timers at the same time or to control a window in which a slave timer is enable. The Counter Enable signal is generated by a logic OR between CEN control bit and the trigger input when configured in gated mode. When the Counter Enable signal is controlled by the trigger input, there is a delay on TRGO, except if the master/slave mode is selected (see the MSM bit description in TIMx\_SMCR register).

010: **Update** - The update event is selected as trigger output (TRGO). For instance a master timer can then be used as a prescaler for a slave timer.

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

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

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

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

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

Bit 3 **CCDS**: Capture/compare DMA selection

- 0: CCx DMA request sent when CCx event occurs
- 1: CCx DMA requests sent when update event occurs

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

- 0: When capture/compare control bits are preloaded (CCPC=1), they are updated by setting the COMG bit only
- 1: When capture/compare control bits are preloaded (CCPC=1), they are updated by setting the COMG bit or when an rising edge occurs on TRGI

*Note: This bit acts only on channels that have a complementary output.*

Bit 1 Reserved, must be kept at reset value.

Bit 0 **CCPC**: Capture/compare preloaded control

- 0: CCxE, CCxNE and OCxM bits are not preloaded
- 1: CCxE, CCxNE and OCxM bits are preloaded, after having been written, they are updated only when a communication event (COM) occurs (COMG bit set or rising edge detected on TRGI, depending on the CCUS bit).

*Note: This bit acts only on channels that have a complementary output.*

### 17.4.3 TIM1 slave mode control register (TIM1\_SMCR)

Address offset: 0x08

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ETP	ECE	ETPS[1:0]		ETF[3:0]				MSM	TS[2:0]			OCCS	SMS[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 **ETP**: External trigger polarity

This bit selects whether ETR or  $\overline{ETR}$  is used for trigger operations

0: ETR is non-inverted, active at high level or rising edge.

1: ETR is inverted, active at low level or falling edge.

Bit 14 **ECE**: External clock enable

This bit enables External clock mode 2.

0: External clock mode 2 disabled

1: External clock mode 2 enabled. The counter is clocked by any active edge on the ETRF signal.

*Note:* 1: Setting the ECE bit has the same effect as selecting external clock mode 1 with TRGI connected to ETRF (SMS=111 and TS=111).

2: It is possible to simultaneously use external clock mode 2 with the following slave modes: reset mode, gated mode and trigger mode. Nevertheless, TRGI must not be connected to ETRF in this case (TS bits must not be 111).

3: If external clock mode 1 and external clock mode 2 are enabled at the same time, the external clock input is ETRF.

Bits 13:12 **ETPS[1:0]**: External trigger prescaler

External trigger signal ETRP frequency must be at most 1/4 of TIMxCLK frequency. A prescaler can be enabled to reduce ETRP frequency. It is useful when inputting fast external clocks.

00: Prescaler OFF

01: ETRP frequency divided by 2

10: ETRP frequency divided by 4

11: ETRP frequency divided by 8

Bits 11:8 **ETF[3:0]**: External trigger filter

This bit-field then defines the frequency used to sample ETRP signal and the length of the digital filter applied to ETRP. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output:

- 0000: No filter, sampling is done at  $f_{DTS}$
- 0001:  $f_{SAMPLING} = f_{CK\_INT}$ , N = 2
- 0010:  $f_{SAMPLING} = f_{CK\_INT}$ , N = 4
- 0011:  $f_{SAMPLING} = f_{CK\_INT}$ , N = 8
- 0100:  $f_{SAMPLING} = f_{DTS} / 2$ , N = 6
- 0101:  $f_{SAMPLING} = f_{DTS} / 2$ , N = 8
- 0110:  $f_{SAMPLING} = f_{DTS} / 4$ , N = 6
- 0111:  $f_{SAMPLING} = f_{DTS} / 4$ , N = 8
- 1000:  $f_{SAMPLING} = f_{DTS} / 8$ , N = 6
- 1001:  $f_{SAMPLING} = f_{DTS} / 8$ , N = 8
- 1010:  $f_{SAMPLING} = f_{DTS} / 16$ , N = 5
- 1011:  $f_{SAMPLING} = f_{DTS} / 16$ , N = 6
- 1100:  $f_{SAMPLING} = f_{DTS} / 16$ , N = 8
- 1101:  $f_{SAMPLING} = f_{DTS} / 32$ , N = 5
- 1110:  $f_{SAMPLING} = f_{DTS} / 32$ , N = 6
- 1111:  $f_{SAMPLING} = f_{DTS} / 32$ , N = 8

*Note: Care must be taken that  $f_{DTS}$  is replaced in the formula by CK\_INT when ETF[3:0] = 1, 2 or 3.*

Bit 7 **MSM**: Master/slave mode

- 0: No action
- 1: The effect of an event on the trigger input (TRGI) is delayed to allow a perfect synchronization between the current timer and its slaves (through TRGO). It is useful if we want to synchronize several timers on a single external event.

Bits 6:4 **TS[2:0]**: Trigger selection

This bit-field selects the trigger input to be used to synchronize the counter.

- 000: Internal Trigger 0 (ITR0)
- 001: Internal Trigger 1 (ITR1)
- 010: Internal Trigger 2 (ITR2)
- 011: Internal Trigger 3 (ITR3)
- 100: TI1 Edge Detector (TI1F\_ED)
- 101: Filtered Timer Input 1 (TI1FP1)
- 110: Filtered Timer Input 2 (TI2FP2)
- 111: External Trigger input (ETRF)

See [Table 62: TIMx Internal trigger connection on page 380](#) for more details on ITRx meaning for each Timer.

*Note: These bits must be changed only when they are not used (e.g. when SMS=000) to avoid wrong edge detections at the transition.*

Bit 3 **OCCS**: OCREF clear selection.

This bit is used to select the OCREF clear source.

- 0:OCREF\_CLR\_INT is connected to the OCREF\_CLR input
- 1: OCREF\_CLR\_INT is connected to ETRF

Bits 2:0 **SMS[2:0]**: Slave mode selection

When external signals are selected the active edge of the trigger signal (TRGI) is linked to the polarity selected on the external input (see Input Control register and Control Register description).

- 000: Slave mode disabled - if CEN = '1' then the prescaler is clocked directly by the internal clock.
- 001: Encoder mode 1 - Counter counts up/down on TI1FP1 edge depending on TI2FP2 level.
- 010: Encoder mode 2 - Counter counts up/down on TI2FP2 edge depending on TI1FP1 level.
- 011: Encoder mode 3 - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input.
- 100: Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter and generates an update of the registers.
- 101: Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high. The counter stops (but is not reset) as soon as the trigger becomes low. Both start and stop of the counter are controlled.
- 110: Trigger Mode - The counter starts at a rising edge of the trigger TRGI (but it is not reset). Only the start of the counter is controlled.
- 111: External Clock Mode 1 - Rising edges of the selected trigger (TRGI) clock the counter.

*Note: The gated mode must not be used if TI1F\_ED is selected as the trigger input (TS='100'). Indeed, TI1F\_ED outputs 1 pulse for each transition on TI1F, whereas the gated mode checks the level of the trigger signal.*

*Note: The clock of the slave timer must be enabled prior to receive events from the master timer, and must not be changed on-the-fly while triggers are received from the master timer.*

**Table 62. TIMx Internal trigger connection**

Slave TIM	ITR0 (TS = 000)	ITR1 (TS = 001)	ITR2 (TS = 010)	ITR3 (TS = 011)
TIM1	TIM15	TIM2	TIM3	TIM17

#### 17.4.4 TIM1 DMA/interrupt enable register (TIM1\_DIER)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	COM DE	CC4DE	CC3DE	CC2DE	CC1DE	UDE	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 Reserved, must be kept at reset value.

Bit 14 **TDE**: Trigger DMA request enable

- 0: Trigger DMA request disabled
- 1: Trigger DMA request enabled

Bit 13 **COMDE**: COM DMA request enable

- 0: COM DMA request disabled
- 1: COM DMA request enabled

Bit 12 **CC4DE**: Capture/Compare 4 DMA request enable

- 0: CC4 DMA request disabled
- 1: CC4 DMA request enabled

Bit 11 **CC3DE**: Capture/Compare 3 DMA request enable

- 0: CC3 DMA request disabled
- 1: CC3 DMA request enabled

Bit 10 **CC2DE**: Capture/Compare 2 DMA request enable

- 0: CC2 DMA request disabled
- 1: CC2 DMA request enabled

Bit 9 **CC1DE**: Capture/Compare 1 DMA request enable

- 0: CC1 DMA request disabled
- 1: CC1 DMA request enabled

Bit 8 **UDE**: Update DMA request enable

- 0: Update DMA request disabled
- 1: Update DMA request enabled

Bit 7 **BIE**: Break interrupt enable

- 0: Break interrupt disabled
- 1: Break interrupt enabled

Bit 6 **TIE**: Trigger interrupt enable

- 0: Trigger interrupt disabled
- 1: Trigger interrupt enabled

Bit 5 **COMIE**: COM interrupt enable

- 0: COM interrupt disabled
- 1: COM interrupt enabled

Bit 4 **CC4IE**: Capture/Compare 4 interrupt enable

- 0: CC4 interrupt disabled
- 1: CC4 interrupt enabled

Bit 3 **CC3IE**: Capture/Compare 3 interrupt enable

- 0: CC3 interrupt disabled
- 1: CC3 interrupt enabled

Bit 2 **CC2IE**: Capture/Compare 2 interrupt enable

- 0: CC2 interrupt disabled
- 1: CC2 interrupt enabled

Bit 1 **CC1IE**: Capture/Compare 1 interrupt enable

- 0: CC1 interrupt disabled
- 1: CC1 interrupt enabled

Bit 0 **UIE**: Update interrupt enable

- 0: Update interrupt disabled
- 1: Update interrupt enabled

### 17.4.5 TIM1 status register (TIM1\_SR)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	CC4OF	CC3OF	CC2OF	CC1OF	Res.	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF
			rc_w0	rc_w0	rc_w0	rc_w0		rc_w0							

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

Bit 12 **CC4OF**: Capture/Compare 4 overcapture flag

refer to CC1OF description

Bit 11 **CC3OF**: Capture/Compare 3 overcapture flag

refer to CC1OF description

Bit 10 **CC2OF**: Capture/Compare 2 overcapture flag

refer to CC1OF description

Bit 9 **CC1OF**: Capture/Compare 1 overcapture flag

This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to '0'.

0: No overcapture has been detected.

1: The counter value has been captured in TIMx\_CCR1 register while CC1IF flag was already set

Bit 8 Reserved, must be kept at reset value.

Bit 7 **BIF**: Break interrupt flag

This flag is set by hardware as soon as the break input goes active. It can be cleared by software if the break input is not active.

0: No break event occurred.

1: An active level has been detected on the break input.

Bit 6 **TIF**: Trigger interrupt flag

This flag is set by hardware on trigger event (active edge detected on TRGI input when the slave mode controller is enabled in all modes but gated mode). It is cleared by software.

0: No trigger event occurred.

1: Trigger interrupt pending.

Bit 5 **COMIF**: COM interrupt flag

This flag is set by hardware on COM event (when Capture/compare Control bits - CCxE, CCxNE, OCxM - have been updated). It is cleared by software by writing it to '0'.

0: No COM event occurred.

1: COM interrupt pending.

Bit 4 **CC4IF**: Capture/Compare 4 interrupt flag

refer to CC1IF description

Bit 3 **CC3IF**: Capture/Compare 3 interrupt flag

refer to CC1IF description

Bit 2 **CC2IF**: Capture/Compare 2 interrupt flag  
refer to CC1IF description

Bit 1 **CC1IF**: Capture/Compare 1 interrupt flag

**If channel CC1 is configured as output:**

This flag is set by hardware when the counter matches the compare value, with some exception in center-aligned mode (refer to the CMS bits in the TIMx\_CR1 register description). It is cleared by software.

0: No match.

1: The content of the counter TIMx\_CNT matches the content of the TIMx\_CCR1 register. When the contents of TIMx\_CCR1 are greater than the contents of TIMx\_ARR, the CC1IF bit goes high on the counter overflow (in upcounting and up/down-counting modes) or underflow (in downcounting mode)

**If channel CC1 is configured as input:**

This bit is set by hardware on a capture. It is cleared by software or by reading the TIMx\_CCR1 register.

0: No input capture occurred

1: The counter value has been captured in TIMx\_CCR1 register (An edge has been detected on IC1 which matches the selected polarity)

Bit 0 **UIF**: Update interrupt flag

This bit is set by hardware on an update event. It is cleared by software.

0: No update occurred.

1: Update interrupt pending. This bit is set by hardware when the registers are updated:

- At overflow or underflow regarding the repetition counter value (update if repetition counter = 0) and if the UDIS=0 in the TIMx\_CR1 register.
- When CNT is reinitialized by software using the UG bit in TIMx\_EGR register, if URS=0 and UDIS=0 in the TIMx\_CR1 register.
- When CNT is reinitialized by a trigger event (refer to [Section 17.4.3: TIM1 slave mode control register \(TIM1\\_SMCR\)](#)), if URS=0 and UDIS=0 in the TIMx\_CR1 register.

#### 17.4.6 TIM1 event generation register (TIM1\_EGR)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	BG	TG	COMG	CC4G	CC3G	CC2G	CC1G	UG							

Bits 15:8 Reserved, must be kept at reset value.

**Bit 7 BG:** Break generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: A break event is generated. MOE bit is cleared and BIF flag is set. Related interrupt or DMA transfer can occur if enabled.

**Bit 6 TG:** Trigger generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: The TIF flag is set in TIMx\_SR register. Related interrupt or DMA transfer can occur if enabled.

**Bit 5 COMG:** Capture/Compare control update generation

This bit can be set by software, it is automatically cleared by hardware

0: No action

1: When CCPC bit is set, it allows to update CCxE, CCxNE and OCxM bits

*Note: This bit acts only on channels having a complementary output.*

**Bit 4 CC4G:** Capture/Compare 4 generation

Refer to CC1G description

**Bit 3 CC3G:** Capture/Compare 3 generation

Refer to CC1G description

**Bit 2 CC2G:** Capture/Compare 2 generation

Refer to CC1G description

**Bit 1 CC1G:** Capture/Compare 1 generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: A capture/compare event is generated on channel 1:

**If channel CC1 is configured as output:**

CC1IF flag is set, Corresponding interrupt or DMA request is sent if enabled.

**If channel CC1 is configured as input:**

The current value of the counter is captured in TIMx\_CCR1 register. The CC1IF flag is set, the corresponding interrupt or DMA request is sent if enabled. The CC1OF flag is set if the CC1IF flag was already high.

**Bit 0 UG:** Update generation

This bit can be set by software, it is automatically cleared by hardware.

0: No action

1: Reinitialize the counter and generates an update of the registers. Note that the prescaler counter is cleared too (anyway the prescaler ratio is not affected). The counter is cleared if the center-aligned mode is selected or if DIR=0 (upcounting), else it takes the auto-reload value (TIMx\_ARR) if DIR=1 (downcounting).

### 17.4.7 TIM1 capture/compare mode register 1 (TIM1\_CCMR1)

Address offset: 0x18

Reset value: 0x0000

The channels can be used in input (capture mode) or in output (compare mode). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OCxx describes its function when the channel is configured in output, ICxx describes its function when the channel is configured in input. So one must take care that the same bit can have a different meaning for the input stage and for the output stage.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2 CE	OC2M[2:0]			OC2 PE	OC2 FE	CC2S[1:0]	OC1 CE	OC1M[2:0]			OC1 PE	OC1 FE	CC1S[1:0]		
IC2F[3:0]			IC2PSC[1:0]		IC1F[3:0]			IC1PSC[1:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

#### Output compare mode

Bit 15 **OC2CE**: Output Compare 2 clear enable

Bits 14:12 **OC2M[2:0]**: Output Compare 2 mode

Bit 11 **OC2PE**: Output Compare 2 preload enable

Bit 10 **OC2FE**: Output Compare 2 fast enable

Bits 9:8 **CC2S[1:0]**: Capture/Compare 2 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output

01: CC2 channel is configured as input, IC2 is mapped on TI2

10: CC2 channel is configured as input, IC2 is mapped on TI1

11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode is working only if an internal trigger input is selected through the TS bit (TIMx\_SMCR register)

*Note: CC2S bits are writable only when the channel is OFF (CC2E = '0' in TIMx\_CCER).*

Bit 7 **OC1CE**: Output Compare 1 clear enable

OC1CE: Output Compare 1 Clear Enable

0: OC1Ref is not affected by the ETRF Input

1: OC1Ref is cleared as soon as a High level is detected on ETRF input

**Bits 6:4 OC1M:** Output Compare 1 mode

These bits define the behavior of the output reference signal OC1REF from which OC1 and OC1N are derived. OC1REF is active high whereas OC1 and OC1N active level depends on CC1P and CC1NP bits.

- 000: Frozen - The comparison between the output compare register TIMx\_CCR1 and the counter TIMx\_CNT has no effect on the outputs (this mode is used to generate a timing base).
- 001: Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).
- 010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).
- 011: Toggle - OC1REF toggles when TIMx\_CNT=TIMx\_CCR1.
- 100: Force inactive level - OC1REF is forced low.
- 101: Force active level - OC1REF is forced high.
- 110: PWM mode 1 - In upcounting, channel 1 is active as long as TIMx\_CNT<TIMx\_CCR1 else inactive. In downcounting, channel 1 is inactive (OC1REF='0') as long as TIMx\_CNT>TIMx\_CCR1 else active (OC1REF='1').
- 111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIMx\_CNT<TIMx\_CCR1 else active. In downcounting, channel 1 is active as long as TIMx\_CNT>TIMx\_CCR1 else inactive.

*Note: 1: These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx\_BDTR register) and CC1S='00' (the channel is configured in output).*

*2: In PWM mode 1 or 2, the OCREF level changes only when the result of the comparison changes or when the output compare mode switches from “frozen” mode to “PWM” mode.*

*3: On channels having a complementary output, this bit field is preloaded. If the CCPC bit is set in the TIMx\_CR2 register then the OC1M active bits take the new value from the preloaded bits only when a COM event is generated.*

Bit 3 **OC1PE**: Output Compare 1 preload enable

0: Preload register on TIMx\_CCR1 disabled. TIMx\_CCR1 can be written at anytime, the new value is taken in account immediately.

1: Preload register on TIMx\_CCR1 enabled. Read/Write operations access the preload register. TIMx\_CCR1 preload value is loaded in the active register at each update event.

*Note: These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx\_BDTR register) and CC1S='00' (the channel is configured in output).*

Bit 2 **OC1FE**: Output Compare 1 fast enable

This bit is used to accelerate the effect of an event on the trigger input on the CC output.

0: CC1 behaves normally depending on counter and CCR1 values even when the trigger is ON. The minimum delay to activate CC1 output when an edge occurs on the trigger input is 5 clock cycles.

1: An active edge on the trigger input acts like a compare match on CC1 output. Then, OC is set to the compare level independently from the result of the comparison. Delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. OCFE acts only if the channel is configured in PWM1 or PWM2 mode.

Bits 1:0 **CC1S**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output

01: CC1 channel is configured as input, IC1 is mapped on TI1

10: CC1 channel is configured as input, IC1 is mapped on TI2

11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC1S bits are writable only when the channel is OFF (CC1E = '0' in TIMx\_CCER).*

## Input capture mode

Bits 15:12 **IC2F**: Input capture 2 filter

Bits 11:10 **IC2PSC[1:0]**: Input capture 2 prescaler

Bits 9:8 **CC2S**: Capture/Compare 2 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output

01: CC2 channel is configured as input, IC2 is mapped on TI2

10: CC2 channel is configured as input, IC2 is mapped on TI1

11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC2S bits are writable only when the channel is OFF (CC2E = '0' in TIMx\_CCER).*

**Bits 7:4 IC1F[3:0]: Input capture 1 filter**

This bit-field defines the frequency used to sample TI1 input and the length of the digital filter applied to TI1. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output:

- 0000: No filter, sampling is done at  $f_{DTS}$
- 0001:  $f_{SAMPLING} = f_{CK\_INT}$ , N = 2
- 0010:  $f_{SAMPLING} = f_{CK\_INT}$ , N = 4
- 0011:  $f_{SAMPLING} = f_{CK\_INT}$ , N = 8
- 0100:  $f_{SAMPLING} = f_{DTS} / 2$ , N = 6
- 0101:  $f_{SAMPLING} = f_{DTS} / 2$ , N = 8
- 0110:  $f_{SAMPLING} = f_{DTS} / 4$ , N = 6
- 0111:  $f_{SAMPLING} = f_{DTS} / 4$ , N = 8
- 1000:  $f_{SAMPLING} = f_{DTS} / 8$ , N = 6
- 1001:  $f_{SAMPLING} = f_{DTS} / 8$ , N = 8
- 1010:  $f_{SAMPLING} = f_{DTS} / 16$ , N = 5
- 1011:  $f_{SAMPLING} = f_{DTS} / 16$ , N = 6
- 1100:  $f_{SAMPLING} = f_{DTS} / 16$ , N = 8
- 1101:  $f_{SAMPLING} = f_{DTS} / 32$ , N = 5
- 1110:  $f_{SAMPLING} = f_{DTS} / 32$ , N = 6
- 1111:  $f_{SAMPLING} = f_{DTS} / 32$ , N = 8

*Note: Care must be taken that  $f_{DTS}$  is replaced in the formula by CK\_INT when ICxF[3:0] = 1, 2 or 3.*

**Bits 3:2 IC1PSC: Input capture 1 prescaler**

This bit-field defines the ratio of the prescaler acting on CC1 input (IC1).  
The prescaler is reset as soon as CC1E='0' (TIMx\_CCER register).

- 00: no prescaler, capture is done each time an edge is detected on the capture input
- 01: capture is done once every 2 events
- 10: capture is done once every 4 events
- 11: capture is done once every 8 events

**Bits 1:0 CC1S: Capture/Compare 1 Selection**

This bit-field defines the direction of the channel (input/output) as well as the used input.

- 00: CC1 channel is configured as output
- 01: CC1 channel is configured as input, IC1 is mapped on TI1
- 10: CC1 channel is configured as input, IC1 is mapped on TI2
- 11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC1S bits are writable only when the channel is OFF (CC1E = '0' in TIMx\_CCER).*

### 17.4.8 TIM1 capture/compare mode register 2 (TIM1\_CCMR2)

Address offset: 0x1C

Reset value: 0x0000

Refer to the above CCMR1 register description.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC4 CE	OC4M[2:0]		OC4 PE	OC4 FE	CC4S[1:0]		OC3 CE.	OC3M[2:0]		OC3 PE	OC3 FE	CC3S[1:0]			
IC4F[3:0]		IC4PSC[1:0]		IC3F[3:0]		IC3PSC[1:0]									
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

## Output compare mode

Bit 15 **OC4CE**: Output compare 4 clear enable

Bits 14:12 **OC4M**: Output compare 4 mode

Bit 11 **OC4PE**: Output compare 4 preload enable

Bit 10 **OC4FE**: Output compare 4 fast enable

Bits 9:8 **CC4S**: Capture/Compare 4 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC4 channel is configured as output

01: CC4 channel is configured as input, IC4 is mapped on TI4

10: CC4 channel is configured as input, IC4 is mapped on TI3

11: CC4 channel is configured as input, IC4 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC4S bits are writable only when the channel is OFF (CC4E = '0' in TIMx\_CCER).*

Bit 7 **OC3CE**: Output compare 3 clear enable

Bits 6:4 **OC3M**: Output compare 3 mode

Bit 3 **OC3PE**: Output compare 3 preload enable

Bit 2 **OC3FE**: Output compare 3 fast enable

Bits 1:0 **CC3S**: Capture/Compare 3 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC3 channel is configured as output

01: CC3 channel is configured as input, IC3 is mapped on TI3

10: CC3 channel is configured as input, IC3 is mapped on TI4

11: CC3 channel is configured as input, IC3 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC3S bits are writable only when the channel is OFF (CC3E = '0' in TIMx\_CCER).*

## Input capture mode

Bits 15:12 **IC4F**: Input capture 4 filter

Bits 11:10 **IC4PSC**: Input capture 4 prescaler

Bits 9:8 **CC4S**: Capture/Compare 4 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC4 channel is configured as output

01: CC4 channel is configured as input, IC4 is mapped on TI4

10: CC4 channel is configured as input, IC4 is mapped on TI3

11: CC4 channel is configured as input, IC4 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC4S bits are writable only when the channel is OFF (CC4E = '0' in TIMx\_CCER).*

Bits 7:4 **IC3F**: Input capture 3 filter

Bits 3:2 **IC3PSC**: Input capture 3 prescaler

Bits 1:0 **CC3S**: Capture/compare 3 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC3 channel is configured as output

01: CC3 channel is configured as input, IC3 is mapped on TI3

10: CC3 channel is configured as input, IC3 is mapped on TI4

11: CC3 channel is configured as input, IC3 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC3S bits are writable only when the channel is OFF (CC3E = '0' in TIMx\_CCER).*

#### 17.4.9 TIM1 capture/compare enable register (TIM1\_CCER)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bit 13 **CC4P**: Capture/Compare 4 output polarity  
refer to CC1P description

Bit 12 **CC4E**: Capture/Compare 4 output enable  
refer to CC1E description

Bit 11 **CC3NP**: Capture/Compare 3 complementary output polarity  
refer to CC1NP description

Bit 10 **CC3NE**: Capture/Compare 3 complementary output enable  
refer to CC1NE description

Bit 9 **CC3P**: Capture/Compare 3 output polarity  
refer to CC1P description

Bit 8 **CC3E**: Capture/Compare 3 output enable  
refer to CC1E description

Bit 7 **CC2NP**: Capture/Compare 2 complementary output polarity  
refer to CC1NP description

Bit 6 **CC2NE**: Capture/Compare 2 complementary output enable  
refer to CC1NE description

Bit 5 **CC2P**: Capture/Compare 2 output polarity  
refer to CC1P description

Bit 4 **CC2E**: Capture/Compare 2 output enable  
refer to CC1E description

Bit 3 **CC1NP**: Capture/Compare 1 complementary output polarity

**CC1 channel configuration as output:**

- 0: OC1N active high.
- 1: OC1N active low.

**CC1 channel configuration as input:**

This bit is used in conjunction with CC1P to define the polarity of TI1FP1 and TI2FP1. Refer to CC1P description.

*Note: On channels having a complementary output, this bit is preloaded. If the CCPC bit is set in the TIMx\_CR2 register then the CC1NP active bit takes the new value from the preloaded bits only when a Commutation event is generated.*

*Note: This bit is not writable as soon as LOCK level 2 or 3 has been programmed (LOCK bits in TIMx\_BDTR register) and CC1S="00" (the channel is configured in output).*

Bit 2 **CC1NE**: Capture/Compare 1 complementary output enable

- 0: Off - OC1N is not active. OC1N level is then function of MOE, OSS1, OSSR, OIS1, OIS1N and CC1E bits.

- 1: On - OC1N signal is output on the corresponding output pin depending on MOE, OSS1, OSSR, OIS1, OIS1N and CC1E bits.

*Note: On channels having a complementary output, this bit is preloaded. If the CCPC bit is set in the TIMx\_CR2 register then the CC1NE active bit takes the new value from the preloaded bits only when a Commutation event is generated.*

Bit 1 **CC1P**: Capture/Compare 1 output polarity

**CC1 channel configured as output:**

- 0: OC1 active high
- 1: OC1 active low

**CC1 channel configured as input:**

CC1NP/CC1P bits select the active polarity of TI1FP1 and TI2FP1 for trigger or capture operations.

00: non-inverted/rising edge

The circuit is sensitive to TIxFP1 rising edge (capture or trigger operations in reset, external clock or trigger mode), TIxFP1 is not inverted (trigger operation in gated mode or encoder mode).

01: inverted/falling edge

The circuit is sensitive to TIxFP1 falling edge (capture or trigger operations in reset, external clock or trigger mode), TIxFP1 is inverted (trigger operation in gated mode or encoder mode).

10: reserved, do not use this configuration.

11: non-inverted/both edges

The circuit is sensitive to both TIxFP1 rising and falling edges (capture or trigger operations in reset, external clock or trigger mode), TIxFP1 is not inverted (trigger operation in gated mode). This configuration must not be used in encoder mode.

*Note: On channels having a complementary output, this bit is preloaded. If the CCPC bit is set in the TIMx\_CR2 register then the CC1P active bit takes the new value from the preloaded bits only when a Commutation event is generated.*

*Note: This bit is not writable as soon as LOCK level 2 or 3 has been programmed (LOCK bits in TIMx\_BDTR register).*

Bit 0 **CC1E**: Capture/Compare 1 output enable

**CC1 channel configured as output:**

0: Off - OC1 is not active. OC1 level is then function of MOE, OSS1, OSSR, OIS1, OIS1N and CC1NE bits.

1: On - OC1 signal is output on the corresponding output pin depending on MOE, OSS1, OSSR, OIS1, OIS1N and CC1NE bits.

**CC1 channel configured as input:**

This bit determines if a capture of the counter value can actually be done into the input capture/compare register 1 (TIMx\_CCR1) or not.

0: Capture disabled.

1: Capture enabled.

*Note: On channels having a complementary output, this bit is preloaded. If the CCPC bit is set in the TIMx\_CR2 register then the CC1E active bit takes the new value from the preloaded bits only when a Commutation event is generated.*

**Table 63. Output control bits for complementary OCx and OCxN channels with break feature**

Control bits					Output states <sup>(1)</sup>	
MOE bit	OSSI bit	OSSR bit	CCxE bit	CCxNE bit	OCx output state	OCxN output state
1	X	0	0	0	Output Disabled (not driven by the timer) OCx=0, OCx_EN=0	Output Disabled (not driven by the timer) OCxN=0, OCx_N_EN=0
		0	0	1	Output Disabled (not driven by the timer) OCx=0, OCx_EN=0	OCxREF + Polarity OCxN=OCxREF xor CCxNP, OCx_N_EN=1
		0	1	0	OCxREF + Polarity OCx=OCxREF xor CCxP, OCx_EN=1	Output Disabled (not driven by the timer) OCxN=0, OCx_N_EN=0
		0	1	1	OCREF + Polarity + dead-time OCx_EN=1	Complementary to OCREF (not OCREF) + Polarity + dead-time OCx_N_EN=1
		1	0	0	Output Disabled (not driven by the timer) OCx=CCxP, OCx_EN=0	Output Disabled (not driven by the timer) OCxN=CCxNP, OCx_N_EN=0
		1	0	1	Off-State (output enabled with inactive state) OCx=CCxP, OCx_EN=1	OCxREF + Polarity OCxN=OCxREF xor CCxNP, OCx_N_EN=1
		1	1	0	OCxREF + Polarity OCx=OCxREF xor CCxP, OCx_EN=1	Off-State (output enabled with inactive state) OCxN=CCxNP, OCx_N_EN=1
		1	1	1	OCREF + Polarity + dead-time OCx_EN=1	Complementary to OCREF (not OCREF) + Polarity + dead-time OCx_N_EN=1

**Table 63. Output control bits for complementary OCx and OCxN channels with break feature (continued)**

Control bits					Output states <sup>(1)</sup>			
MOE bit	OSSI bit	OSSR bit	CCxE bit	CCxNE bit	OCx output state		OCxN output state	
0	0	X	0	0	Output Disabled (not driven by the timer) OCx=CCxP, OCx_EN=0		Output Disabled (not driven by the timer) OCxN=CCxNP, OCxN_EN=0	
	0		0	1	Output Disabled (not driven by the timer)			
	0		1	0	Asynchronously: OCx=CCxP, OCx_EN=0, OCxN=CCxNP, OCxN_EN=0			
	0		1	1	Then if the clock is present: OCx=OISx and OCxN=OISxN after a dead-time, assuming that OISx and OISxN do not correspond to OCX and OCxN both in active state.			
	1		0	0	Output Disabled (not driven by the timer) OCx=CCxP, OCx_EN=0		Output Disabled (not driven by the timer) OCxN=CCxNP, OCxN_EN=0	
	1		0	1	Off-State (output enabled with inactive state)			
	1		1	0	Asynchronously: OCx=CCxP, OCx_EN=1, OCxN=CCxNP, OCxN_EN=1			
	1		1	1	Then if the clock is present: OCx=OISx and OCxN=OISxN after a dead-time, assuming that OISx and OISxN do not correspond to OCX and OCxN both in active state.			

- When both outputs of a channel are not used (CCxE = CCxNE = 0), the OISx, OISxN, CCxP and CCxNP bits must be kept cleared.

**Note:** The state of the external I/O pins connected to the complementary OCx and OCxN channels depends on the OCx and OCxN channel state and the GPIO registers.

#### 17.4.10 TIM1 counter (TIM1\_CNT)

Address offset: 0x24

Reset value: 0x0000

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

Bits 15:0      **CNT[15:0]:** Counter value

### 17.4.11 TIM1 prescaler (TIM1\_PSC)

Address offset: 0x28

Reset value: 0x0000

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

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency ( $CK_{CNT}$ ) is equal to  $f_{CK\_PSC} / (PSC[15:0] + 1)$ .

PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx\_EGR register or through trigger controller when configured in “reset mode”).

### 17.4.12 TIM1 auto-reload register (TIM1\_ARR)

Address offset: 0x2C

Reset value: 0xFFFF

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

Bits 15:0 **ARR[15:0]**: Auto-reload value

ARR is the value to be loaded in the actual auto-reload register.

Refer to the [Section 17.3.1: Time-base unit on page 330](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

### 17.4.13 TIM1 repetition counter register (TIM1\_RCR)

Address offset: 0x30

Reset value: 0x0000

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

Bits 15:8 Reserved, must be kept at reset value.

Bits 7:0 **REP[7:0]**: Repetition counter value

These bits allow the user to set-up the update rate of the compare registers (i.e. periodic transfers from preload to active registers) when preload registers are enable, as well as the update interrupt generation rate, if this interrupt is enable.

Each time the REP\_CNT related downcounter reaches zero, an update event is generated and it restarts counting from REP value. As REP\_CNT is reloaded with REP value only at the repetition update event U\_RC, any write to the TIMx\_RCR register is not taken in account until the next repetition update event.

It means in PWM mode (REP+1) corresponds to:

- the number of PWM periods in edge-aligned mode
- the number of half PWM period in center-aligned mode.

#### 17.4.14 TIM1 capture/compare register 1 (TIM1\_CCR1)

Address offset: 0x34

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR1[15:0]**: Capture/Compare 1 value

**If channel CC1 is configured as output:**

CCR1 is the value to be loaded in the actual capture/compare 1 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR1 register (bit OC1PE). Else the preload value is copied in the active capture/compare 1 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signaled on OC1 output.

**If channel CC1 is configured as input:**

CCR1 is the counter value transferred by the last input capture 1 event (IC1).

#### 17.4.15 TIM1 capture/compare register 2 (TIM1\_CCR2)

Address offset: 0x38

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR2[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR2[15:0]**: Capture/Compare 2 value

**If channel CC2 is configured as output:**

CCR2 is the value to be loaded in the actual capture/compare 2 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR2 register (bit OC2PE). Else the preload value is copied in the active capture/compare 2 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signalled on OC2 output.

**If channel CC2 is configured as input:**

CCR2 is the counter value transferred by the last input capture 2 event (IC2).

#### 17.4.16 TIM1 capture/compare register 3 (TIM1\_CCR3)

Address offset: 0x3C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR3[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR3[15:0]**: Capture/Compare value

**If channel CC3 is configured as output:**

CCR3 is the value to be loaded in the actual capture/compare 3 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR3 register (bit OC3PE). Else the preload value is copied in the active capture/compare 3 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signalled on OC3 output.

**If channel CC3 is configured as input:**

CCR3 is the counter value transferred by the last input capture 3 event (IC3).

### 17.4.17 TIM1 capture/compare register 4 (TIM1\_CCR4)

Address offset: 0x40

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR4[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR4[15:0]**: Capture/Compare value

**If channel CC4 is configured as output:**

CCR4 is the value to be loaded in the actual capture/compare 4 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR4 register (bit OC4PE). Else the preload value is copied in the active capture/compare 4 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signalled on OC4 output.

**If channel CC4 is configured as input:**

CCR4 is the counter value transferred by the last input capture 4 event (IC4).

### 17.4.18 TIM1 break and dead-time register (TIM1\_BDTR)

Address offset: 0x44

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DTG[7:0]															
MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK[1:0]		rw							

**Note:** As the bits AOE, BKP, BKE, OSSR and DTG[7:0] can be write-locked depending on the LOCK configuration, it can be necessary to configure all of them during the first write access to the TIMx\_BDTR register.

Bit 15 **MOE**: Main output enable

This bit is cleared asynchronously by hardware as soon as the break input is active. It is set by software or automatically depending on the AOE bit. It is acting only on the channels which are configured in output.

0: OC and OCN outputs are disabled or forced to idle state.

1: OC and OCN outputs are enabled if their respective enable bits are set (CCxE, CCxNE in TIMx\_CCER register).

See OC/OCN enable description for more details ([Section 17.4.9: TIM1 capture/compare enable register \(TIM1\\_CCER\) on page 391](#)).

Bit 14 **AOE**: Automatic output enable

0: MOE can be set only by software

1: MOE can be set by software or automatically at the next update event (if the break input is not be active)

**Note:** This bit cannot be modified as long as LOCK level 1 has been programmed (LOCK bits in TIMx\_BDTR register).

Bit 13 **BKP**: Break polarity

- 0: Break input BRK is active low
- 1: Break input BRK is active high

*Note: This bit can not be modified as long as LOCK level 1 has been programmed (LOCK bits in TIMx\_BDTR register).*

*Note: Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.*

Bit 12 **BKE**: Break enable

- 0: Break inputs (BRK and CCS clock failure event) disabled
- 1: Break inputs (BRK and CCS clock failure event) enabled

*Note: This bit cannot be modified when LOCK level 1 has been programmed (LOCK bits in TIMx\_BDTR register).*

*Note: Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.*

Bit 11 **OSSR**: Off-state selection for Run mode

This bit is used when MOE=1 on channels having a complementary output which are configured as outputs. OSSR is not implemented if no complementary output is implemented in the timer.

See OC/OCN enable description for more details ([Section 17.4.9: TIM1 capture/compare enable register \(TIM1\\_CCER\) on page 391](#)).

- 0: When inactive, OC/OCN outputs are disabled (OC/OCN enable output signal=0).
- 1: When inactive, OC/OCN outputs are enabled with their inactive level as soon as CCxE=1 or CCxNE=1. Then, OC/OCN enable output signal=1

*Note: This bit can not be modified as soon as the LOCK level 2 has been programmed (LOCK bits in TIMx\_BDTR register).*

Bit 10 **OSSI**: Off-state selection for Idle mode

This bit is used when MOE=0 on channels configured as outputs.

See OC/OCN enable description for more details ([Section 17.4.9: TIM1 capture/compare enable register \(TIM1\\_CCER\) on page 391](#)).

- 0: When inactive, OC/OCN outputs are disabled (OC/OCN enable output signal=0).
- 1: When inactive, OC/OCN outputs are forced first with their idle level as soon as CCxE=1 or CCxNE=1. OC/OCN enable output signal=1

*Note: This bit can not be modified as soon as the LOCK level 2 has been programmed (LOCK bits in TIMx\_BDTR register).*

Bits 9:8 **LOCK[1:0]**: Lock configuration

These bits offer a write protection against software errors.

- 00: LOCK OFF - No bit is write protected.
- 01: LOCK Level 1 = DTG bits in TIMx\_BDTR register, OISx and OISxN bits in TIMx\_CR2 register and BKE/BKP/AOE bits in TIMx\_BDTR register can no longer be written.
- 10: LOCK Level 2 = LOCK Level 1 + CC Polarity bits (CCxP/CCxNP bits in TIMx\_CCER register, as long as the related channel is configured in output through the CCxS bits) as well as OSSR and OSSI bits can no longer be written.
- 11: LOCK Level 3 = LOCK Level 2 + CC Control bits (OCxM and OCxPE bits in TIMx\_CCMRx registers, as long as the related channel is configured in output through the CCxS bits) can no longer be written.

*Note: The LOCK bits can be written only once after the reset. Once the TIMx\_BDTR register has been written, their content is frozen until the next reset.*

**Bits 7:0 DTG[7:0]: Dead-time generator setup**

This bit-field defines the duration of the dead-time inserted between the complementary outputs. DT correspond to this duration.

DTG[7:5]=0xx => DT=DTG[7:0]x t<sub>dtg</sub> with t<sub>dtg</sub>=t<sub>DTS</sub>.

DTG[7:5]=10x => DT=(64+DTG[5:0])x t<sub>dtg</sub> with T<sub>dtg</sub>=2x t<sub>DTS</sub>.

DTG[7:5]=110 => DT=(32+DTG[4:0])x t<sub>dtg</sub> with T<sub>dtg</sub>=8x t<sub>DTS</sub>.

DTG[7:5]=111 => DT=(32+DTG[4:0])x t<sub>dtg</sub> with T<sub>dtg</sub>=16x t<sub>DTS</sub>.

Example if T<sub>DTS</sub>=125 ns (8 MHz), dead-time possible values are:

0 to 15875 ns by 125 ns steps,

16 us to 31750 ns by 250 ns steps,

32 us to 63 us by 1 us steps,

64 us to 126 us by 2 us steps

*Note: This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx\_BDTR register).*

#### 17.4.19 TIM1 DMA control register (TIM1\_DCR)

Address offset: 0x48

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.			DBL[4:0]		Res.	Res.	Res.			DBA[4:0]			

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

**Bits 12:8 DBL[4:0]: DMA burst length**

This 5-bit vector defines the number of DMA transfers (the timer recognizes a burst transfer when a read or a write access is done to the TIMx\_DMAR address)

00000: 1 transfer

00001: 2 transfers

00010: 3 transfers

...

10001: 18 transfers

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

**Bits 4:0 DBA[4:0]: DMA base address**

This 5-bit vector defines the base-address for DMA transfers (when read/write access are done through the TIMx\_DMAR address). DBA is defined as an offset starting from the address of the TIMx\_CR1 register.

Example:

00000: TIMx\_CR1,

00001: TIMx\_CR2,

00010: TIMx\_SMCR,

...

**Example:** Let us consider the following transfer: DBL = 7 transfers and DBA = TIMx\_CR1. In this case the transfer is done to/from 7 registers starting from the TIMx\_CR1 address.

### 17.4.20 TIM1 DMA address for full transfer (TIM1\_DMAR)

Address offset: 0x4C

Reset value: 0x0000

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

Bits 15:0 **DMAB[15:0]**: DMA register for burst accesses

A read or write operation to the DMAR register accesses the register located at the address (TIMx\_CR1 address) + (DBA + DMA index) × 4

where TIMx\_CR1 address is the address of the control register 1, DBA is the DMA base address configured in TIMx\_DCR register, DMA index is automatically controlled by the DMA transfer, and ranges from 0 to DBL (DBL configured in TIMx\_DCR).

#### Example of how to use the DMA burst feature

In this example the timer DMA burst feature is used to update the contents of the CCRx registers ( $x = 2, 3, 4$ ) with the DMA transferring half words into the CCRx registers.

This is done in the following steps:

1. Configure the corresponding DMA channel as follows:
  - DMA channel peripheral address is the DMAR register address
  - DMA channel memory address is the address of the buffer in the RAM containing the data to be transferred by DMA into CCRx registers.
  - Number of data to transfer = 3 (See note below).
  - Circular mode disabled.
2. Configure the DCR register by configuring the DBA and DBL bit fields as follows:  
DBL = 3 transfers, DBA = 0xE.
3. Enable the TIMx update DMA request (set the UDE bit in the DIER register).
4. Enable TIMx
5. Enable the DMA channel

Note:

*This example is for the case where every CCRx register to be updated once. If every CCRx register is to be updated twice for example, the number of data to transfer should be 6. Let's take the example of a buffer in the RAM containing data1, data2, data3, data4, data5 and data6. The data is transferred to the CCRx registers as follows: on the first update DMA request, data1 is transferred to CCR2, data2 is transferred to CCR3, data3 is transferred to CCR4 and on the second update DMA request, data4 is transferred to CCR2, data5 is transferred to CCR3 and data6 is transferred to CCR4.*

### 17.4.21 TIM1 register map

TIM1 registers are mapped as 16-bit addressable registers as described in the table below:

**Table 64. TIM1 register map and reset values**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	<b>TIM1_CR1</b>	Res.																															
	Reset value																																
0x04	<b>TIM1_CR2</b>	Res.																															
	Reset value																																
0x08	<b>TIM1_SMCR</b>	Res.																															
	Reset value																																
0x0C	<b>TIM1_DIER</b>	Res.																															
	Reset value																																
0x10	<b>TIM1_SR</b>	Res.																															
	Reset value																																
0x14	<b>TIM1_EGR</b>	Res.																															
	Reset value																																
0x18	<b>TIM1_CCMR1</b> Output compare mode	Res.																															
	Reset value																																
	<b>TIM1_CCMR1</b> Input capture mode	Res.																															
	Reset value																																
0x1C	<b>TIM1_CCMR2</b> Output compare mode	Res.																															
	Reset value																																
	<b>TIM1_CCMR2</b> Input capture mode	Res.																															
	Reset value																																
0x20	<b>TIM1_CCER</b>	Res.																															
	Reset value																																
0x24	<b>TIM1_CNT</b>	Res.																															
	Reset value																																
0x28	<b>TIM1_PSC</b>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Reset value																																
0x2C	<b>TIM1_ARR</b>	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	Reset value																																
0x30	<b>TIM1_RCR</b>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Reset value																																

CNT[15:0]

PSC[15:0]

ARR[15:0]

REP[7:0]

**Table 64. TIM1 register map and reset values (continued)**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x34	<b>TIM1_CCR1</b>	Res.																															
	Reset value																																
0x38	<b>TIM1_CCR2</b>	Res.																															
	Reset value																																
0x3C	<b>TIM1_CCR3</b>	Res.																															
	Reset value																																
0x40	<b>TIM1_CCR4</b>	Res.																															
	Reset value																																
0x44	<b>TIM1_BDTR</b>	Res.																															
	Reset value																																
0x48	<b>TIM1_DCR</b>	Res.																															
	Reset value																																
0x4C	<b>TIM1_DMAR</b>	Res.																															
	Reset value																																

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 18 General-purpose timers (TIM2 and TIM3)

### 18.1 TIM2 and TIM3 introduction

The general-purpose timers consist of a 16-bit or 32-bit auto-reload counter driven by a programmable prescaler.

They may be used for a variety of purposes, including measuring the pulse lengths of input signals (*input capture*) or generating output waveforms (*output compare and PWM*).

Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the RCC clock controller prescalers.

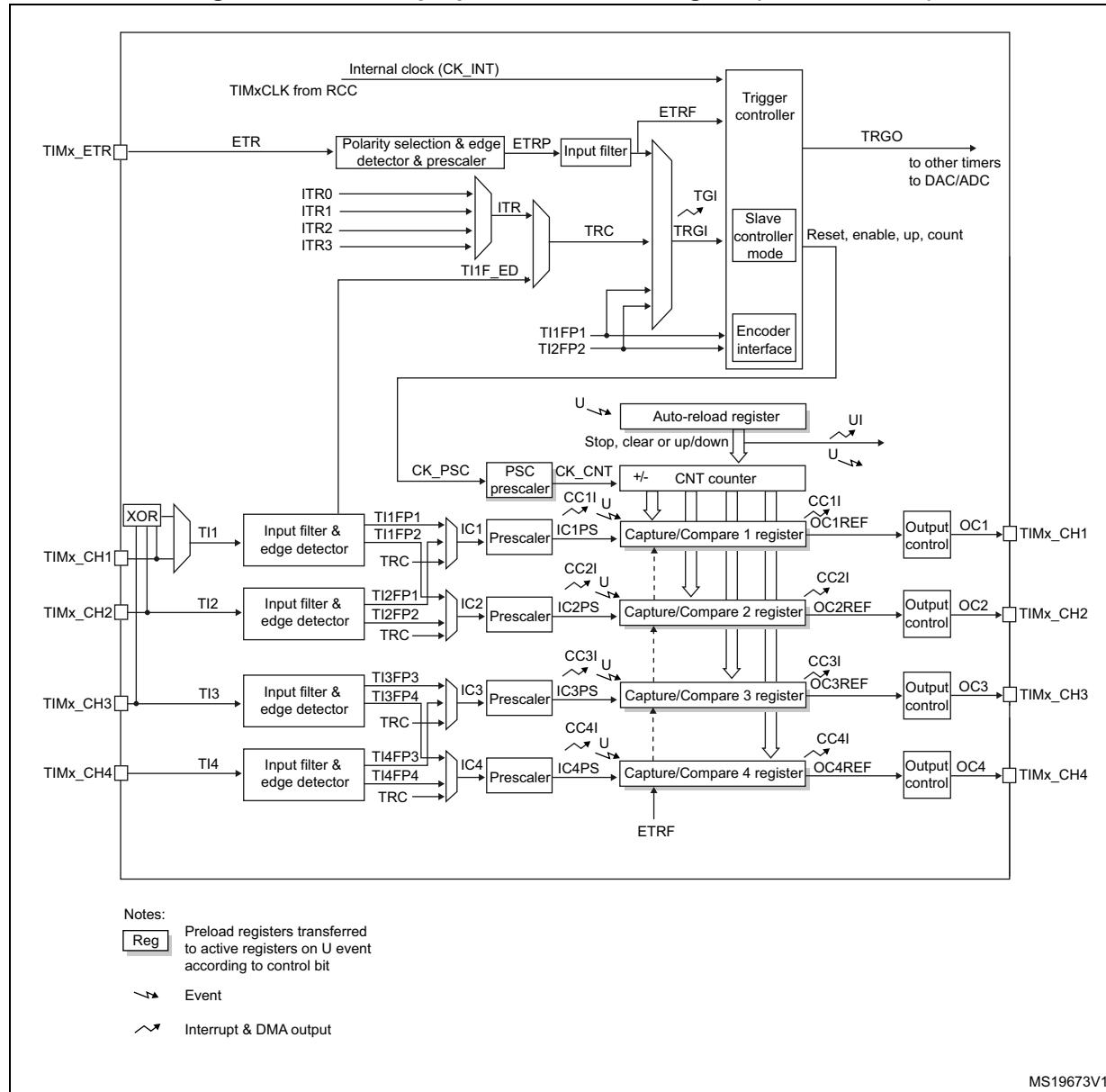
The timers are completely independent, and do not share any resources. They can be synchronized together as described in [Section 18.3.15](#).

### 18.2 TIM2 and TIM3 main features

General-purpose TIMx timer features include:

- 16-bit (TIM3) or 32-bit (TIM2) up, down, up/down auto-reload counter.
- 16-bit programmable prescaler used to divide (also “on the fly”) the counter clock frequency by any factor between 1 and 65535.
- Up to 4 independent channels for:
  - Input capture
  - Output compare
  - PWM generation (Edge- and Center-aligned modes)
  - One-pulse mode output
- Synchronization circuit to control the timer with external signals and to interconnect several timers.
- Interrupt/DMA generation on the following events:
  - Update: counter overflow/underflow, counter initialization (by software or internal/external trigger)
  - Trigger event (counter start, stop, initialization or count by internal/external trigger)
  - Input capture
  - Output compare
- Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes
- Trigger input for external clock or cycle-by-cycle current management

Figure 111. General-purpose timer block diagram (TIM2 and TIM3)



## 18.3 TIM2 and TIM3 functional description

### 18.3.1 Time-base unit

The main block of the programmable timer is a 16-bit/32-bit counter with its related auto-reload register. The counter can count up but also down or both up and down. The counter clock can be divided by a prescaler.

The counter, the auto-reload register and the prescaler register can be written or read by software. This is true even when the counter is running.

The time-base unit includes:

- Counter Register (TIMx\_CNT)
- Prescaler Register (TIMx\_PSC)
- Auto-Reload Register (TIMx\_ARR)

The auto-reload register is preloaded. Writing to or reading from the auto-reload register accesses the preload register. The content of the preload register are transferred into the shadow register permanently or at each update event (UEV), depending on the auto-reload preload enable bit (ARPE) in TIMx\_CR1 register. The update event is sent when the counter reaches the overflow (or underflow when downcounting) and if the UDIS bit equals 0 in the TIMx\_CR1 register. It can also be generated by software. The generation of the update event is described in detail for each configuration.

The counter is clocked by the prescaler output CK\_CNT, which is enabled only when the counter enable bit (CEN) in TIMx\_CR1 register is set (refer also to the slave mode controller description to get more details on counter enabling).

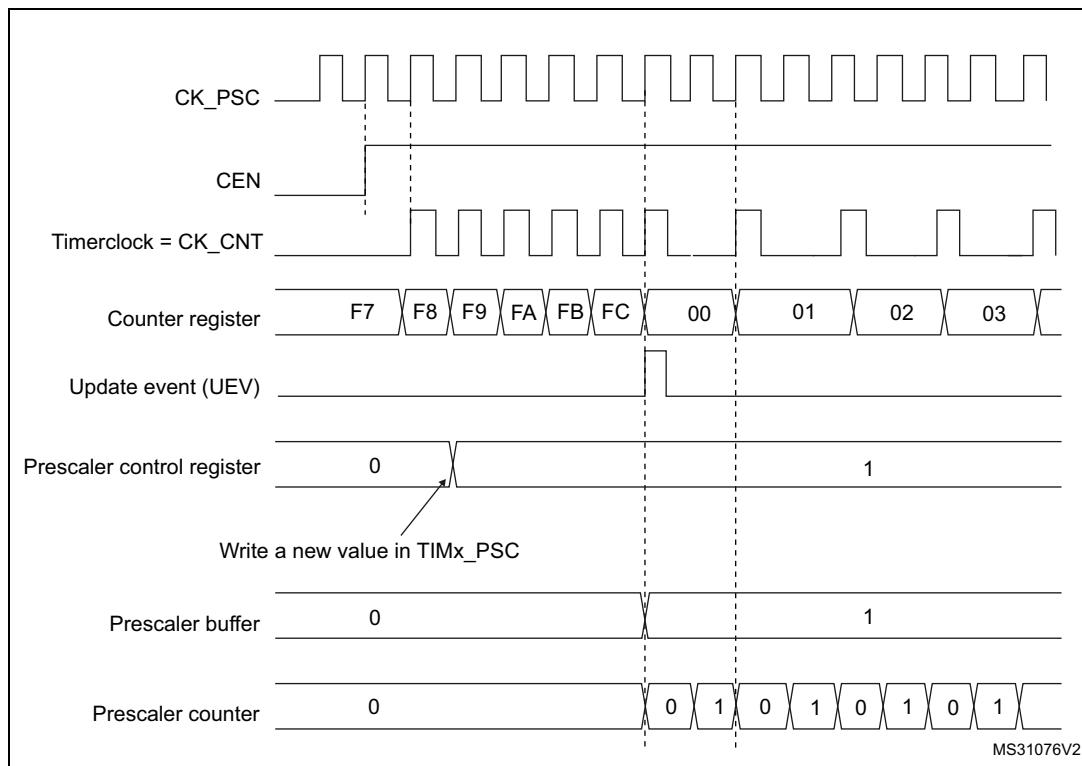
Note that the actual counter enable signal CNT\_EN is set 1 clock cycle after CEN.

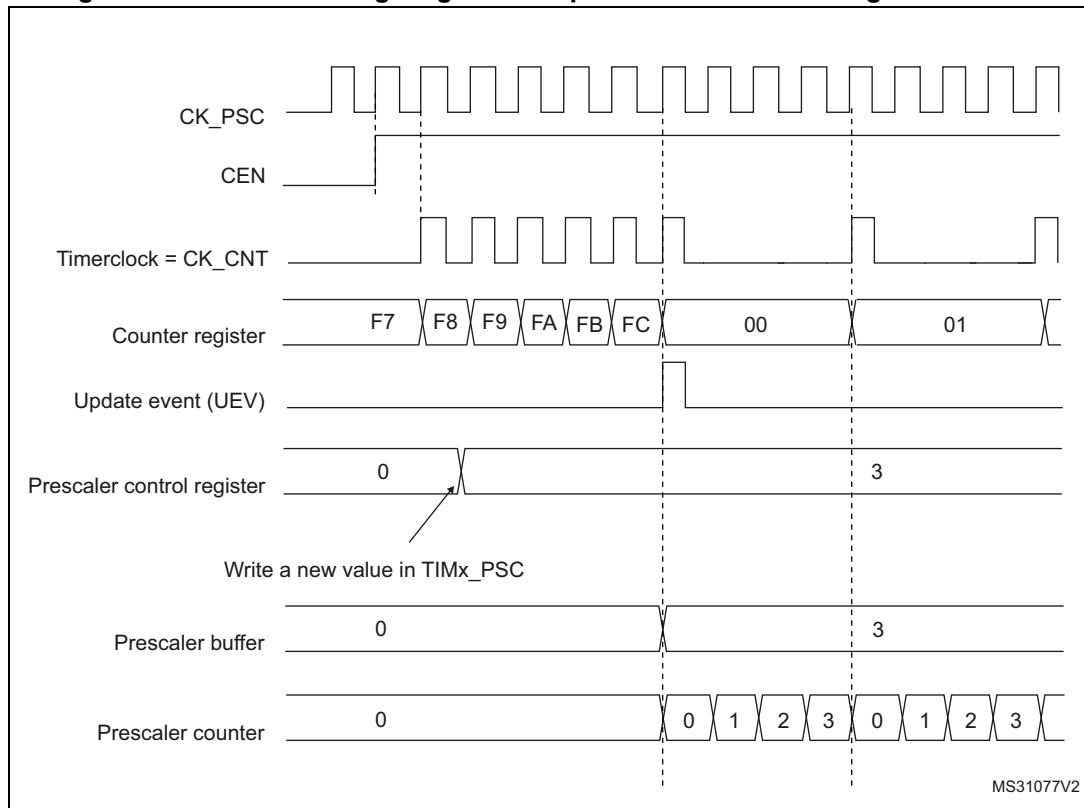
### Prescaler description

The prescaler can divide the counter clock frequency by any factor between 1 and 65536. It is based on a 16-bit counter controlled through a 16-bit/32-bit register (in the TIMx\_PSC register). It can be changed on the fly as this control register is buffered. The new prescaler ratio is taken into account at the next update event.

[Figure 112](#) and [Figure 113](#) give some examples of the counter behavior when the prescaler ratio is changed on the fly:

**Figure 112. Counter timing diagram with prescaler division change from 1 to 2**



**Figure 113. Counter timing diagram with prescaler division change from 1 to 4**

### 18.3.2 Counter modes

#### Upcounting mode

In upcounting mode, the counter counts from 0 to the auto-reload value (content of the TIMx\_ARR register), then restarts from 0 and generates a counter overflow event.

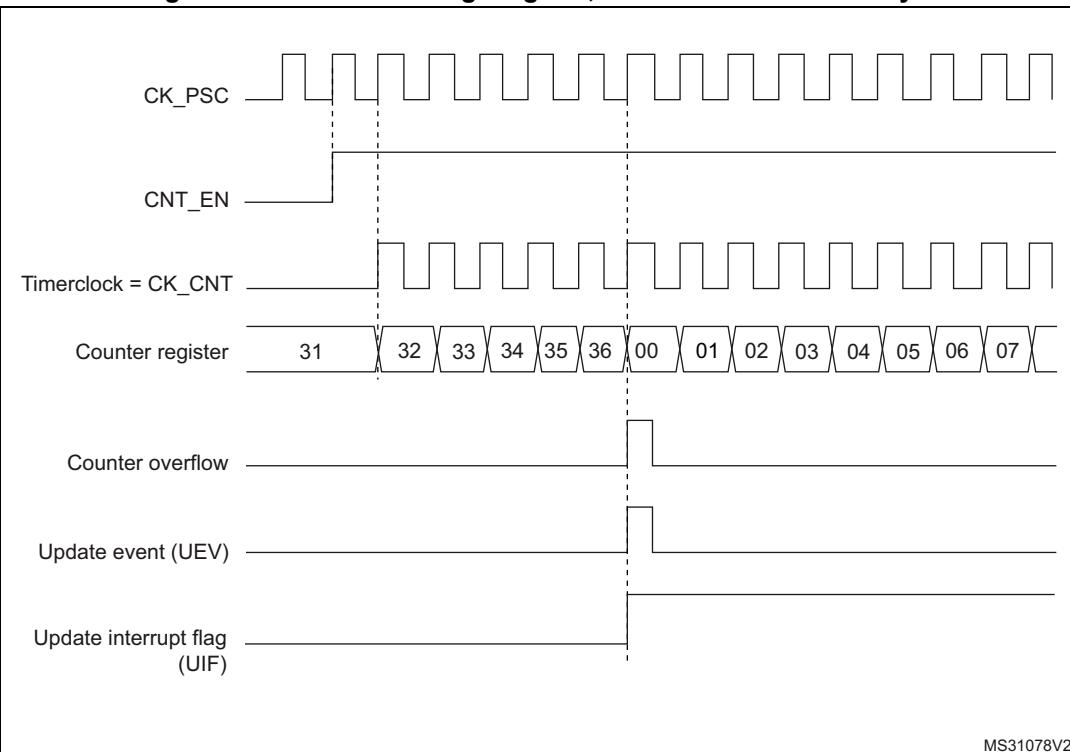
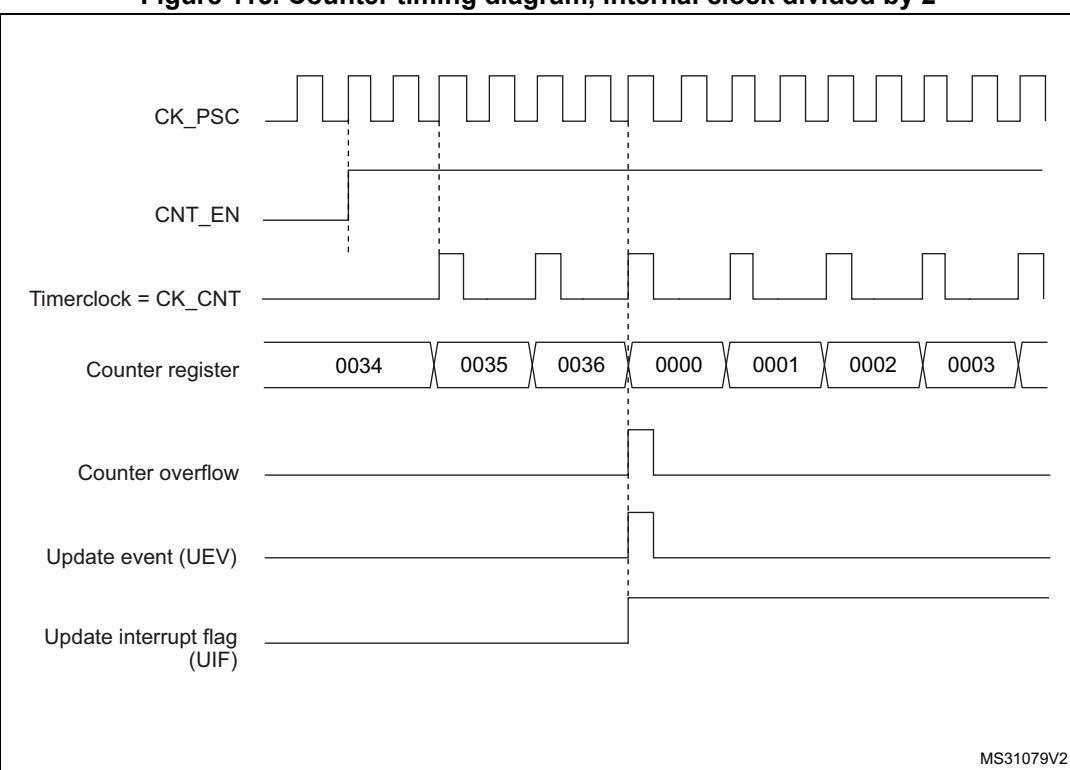
An Update event can be generated at each counter overflow or by setting the UG bit in the TIMx\_EGR register (by software or by using the slave mode controller).

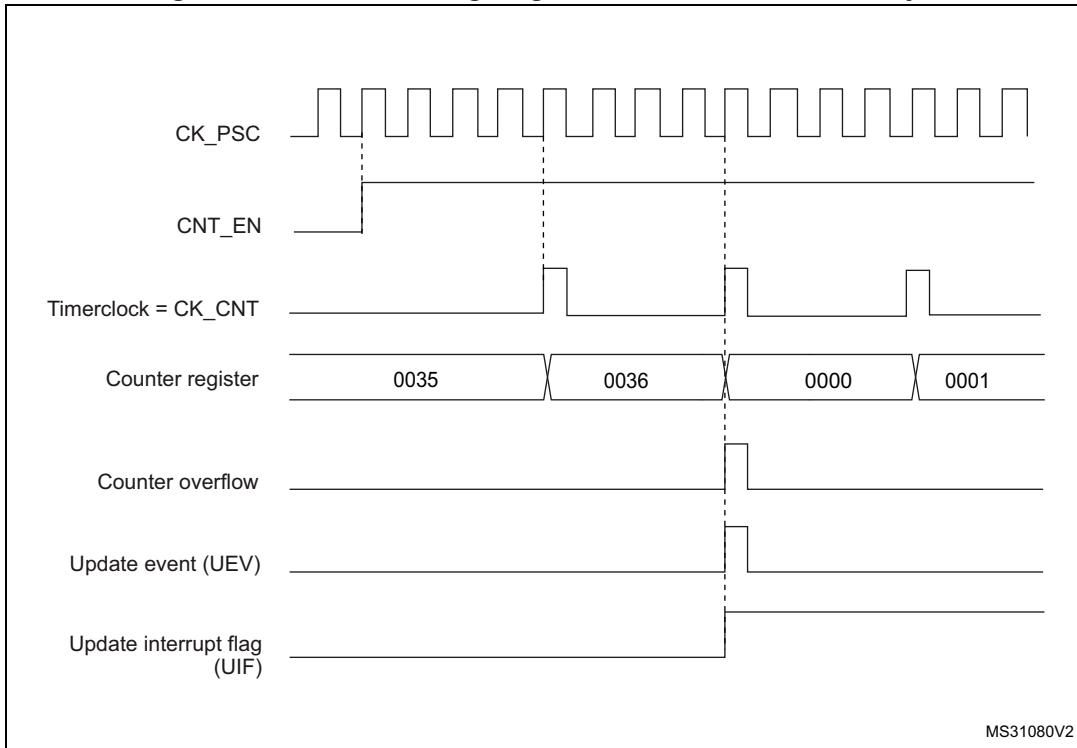
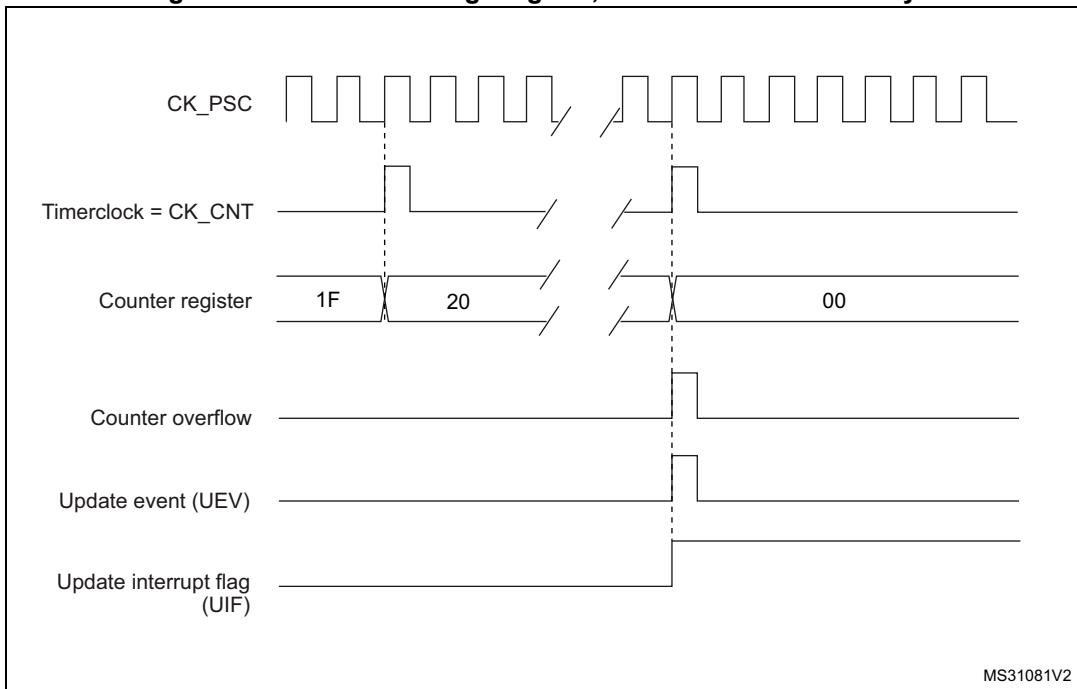
The UEV event can be disabled by software by setting the UDIS bit in TIMx\_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until the UDIS bit has been written to 0. However, the counter restarts from 0, as well as the counter of the prescaler (but the prescale rate does not change). In addition, if the URS bit (update request selection) in TIMx\_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

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

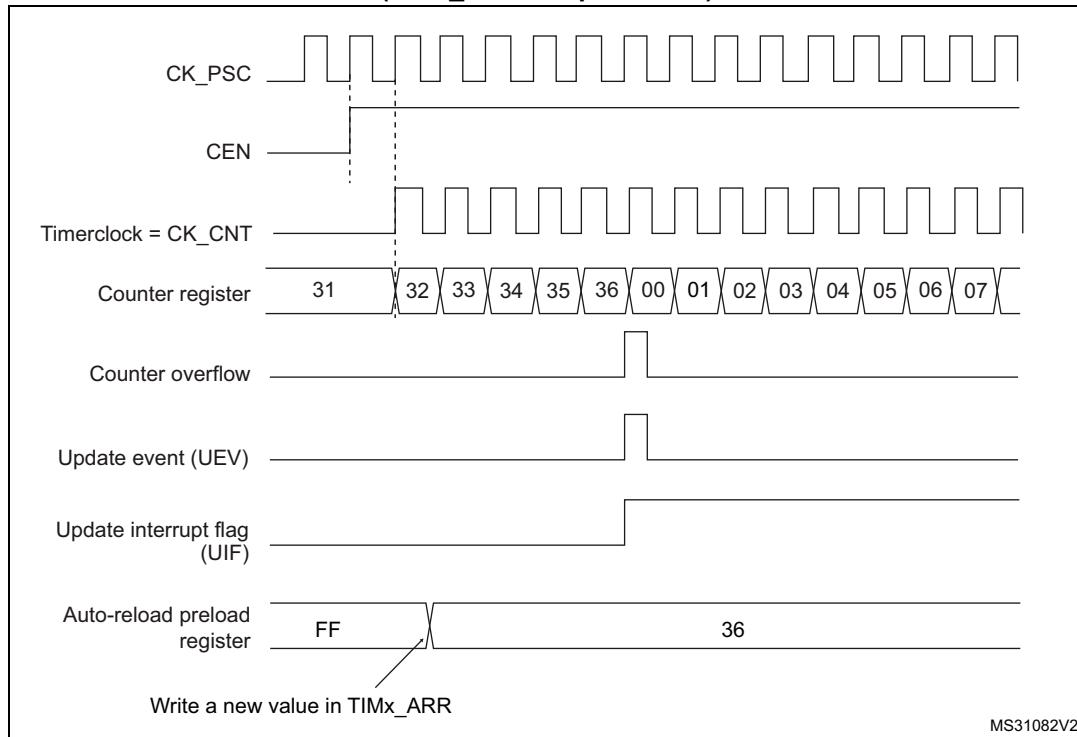
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx\_PSC register)
- The auto-reload shadow register is updated with the preload value (TIMx\_ARR)

The following figures show some examples of the counter behavior for different clock frequencies when TIMx\_ARR=0x36.

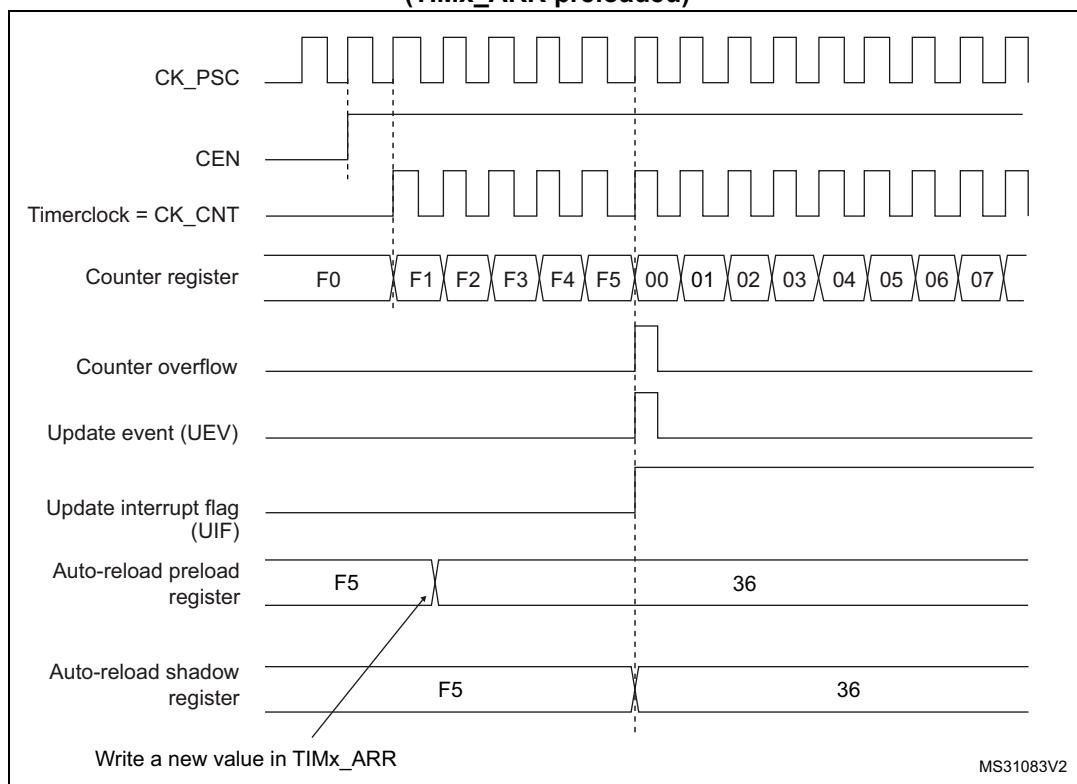
**Figure 114. Counter timing diagram, internal clock divided by 1****Figure 115. Counter timing diagram, internal clock divided by 2**

**Figure 116. Counter timing diagram, internal clock divided by 4****Figure 117. Counter timing diagram, internal clock divided by N**

**Figure 118. Counter timing diagram, Update event when ARPE=0  
(TIMx\_ARR not preloaded)**



**Figure 119. Counter timing diagram, Update event when ARPE=1  
(TIMx\_ARR preloaded)**



### Downcounting mode

In downcounting mode, the counter counts from the auto-reload value (content of the TIMx\_ARR register) down to 0, then restarts from the auto-reload value and generates a counter underflow event.

An Update event can be generated at each counter underflow or by setting the UG bit in the TIMx\_EGR register (by software or by using the slave mode controller).

The UEV update event can be disabled by software by setting the UDIS bit in TIMx\_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until UDIS bit has been written to 0.

However, the counter restarts from the current auto-reload value, whereas the counter of the prescaler restarts from 0 (but the prescale rate doesn't change).

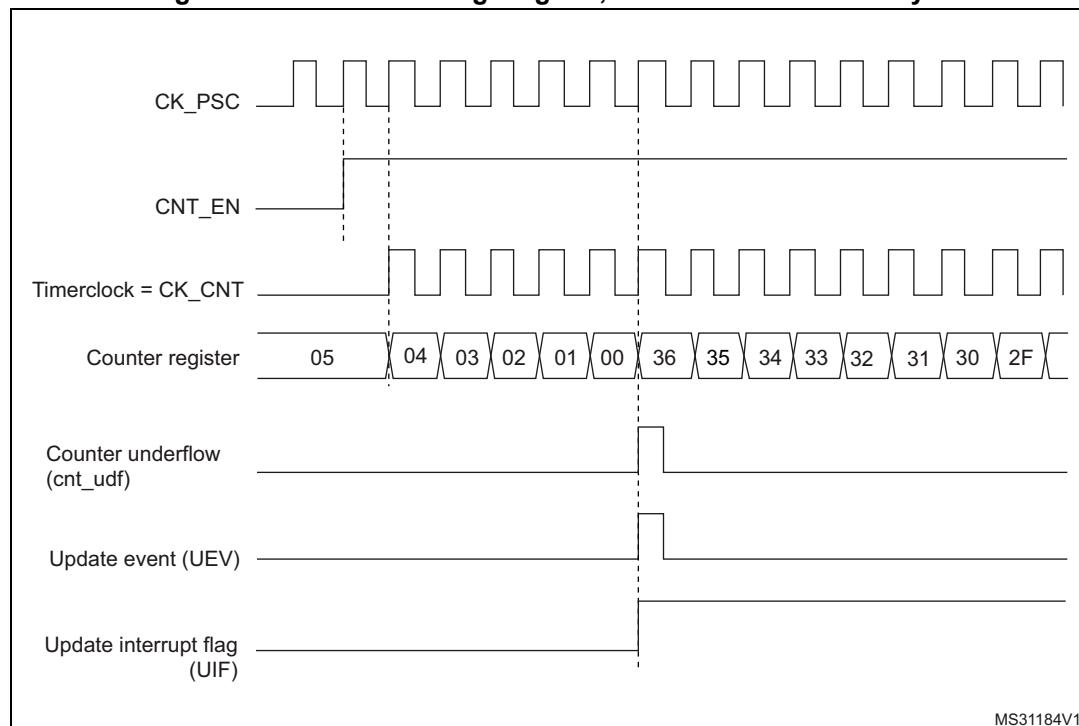
In addition, if the URS bit (update request selection) in TIMx\_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on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

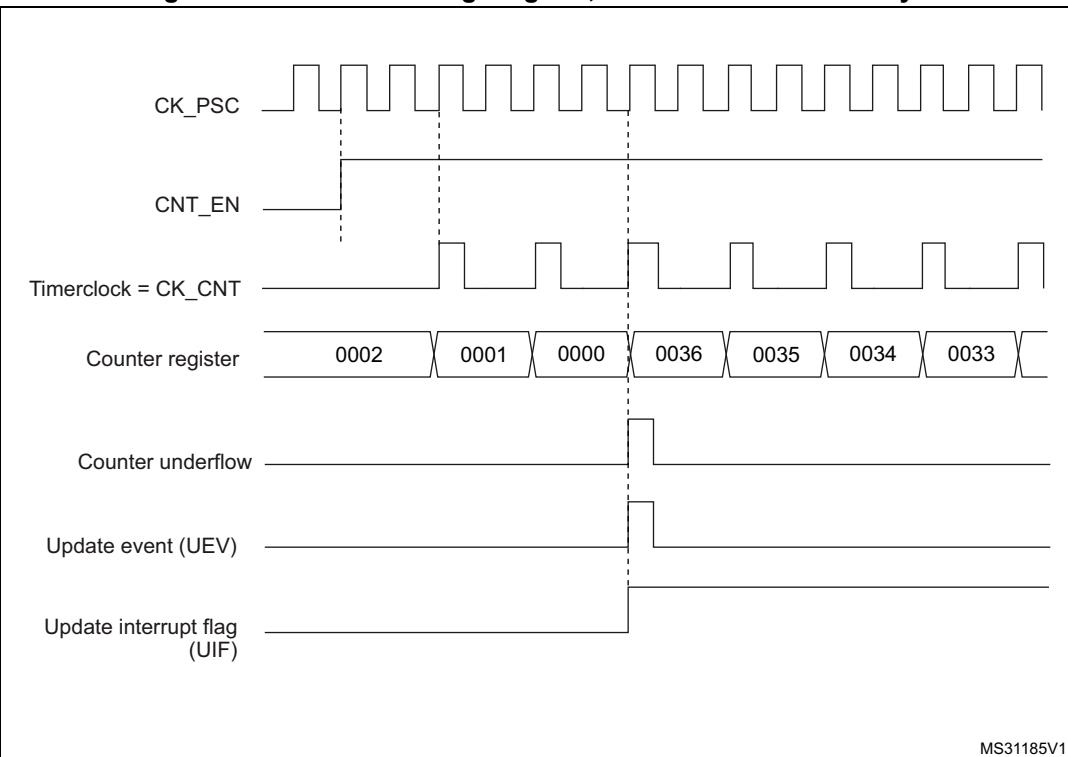
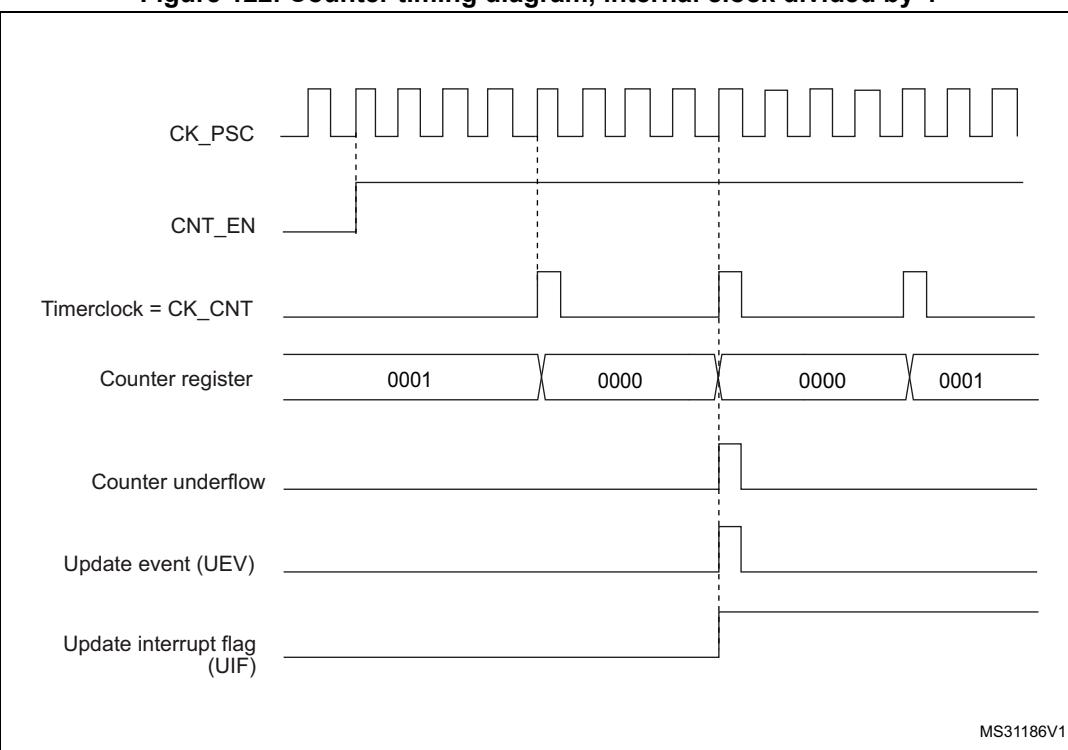
When an update event occurs, all the registers are updated and the update flag (UIF bit in TIMx\_SR register) is set (depending on the URS bit):

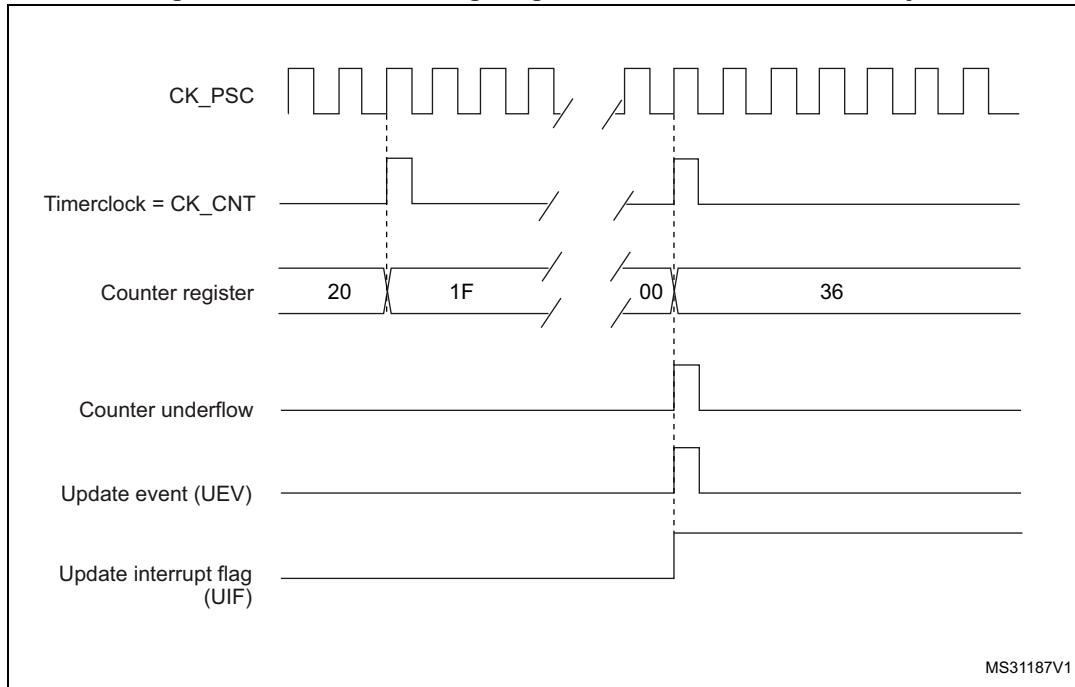
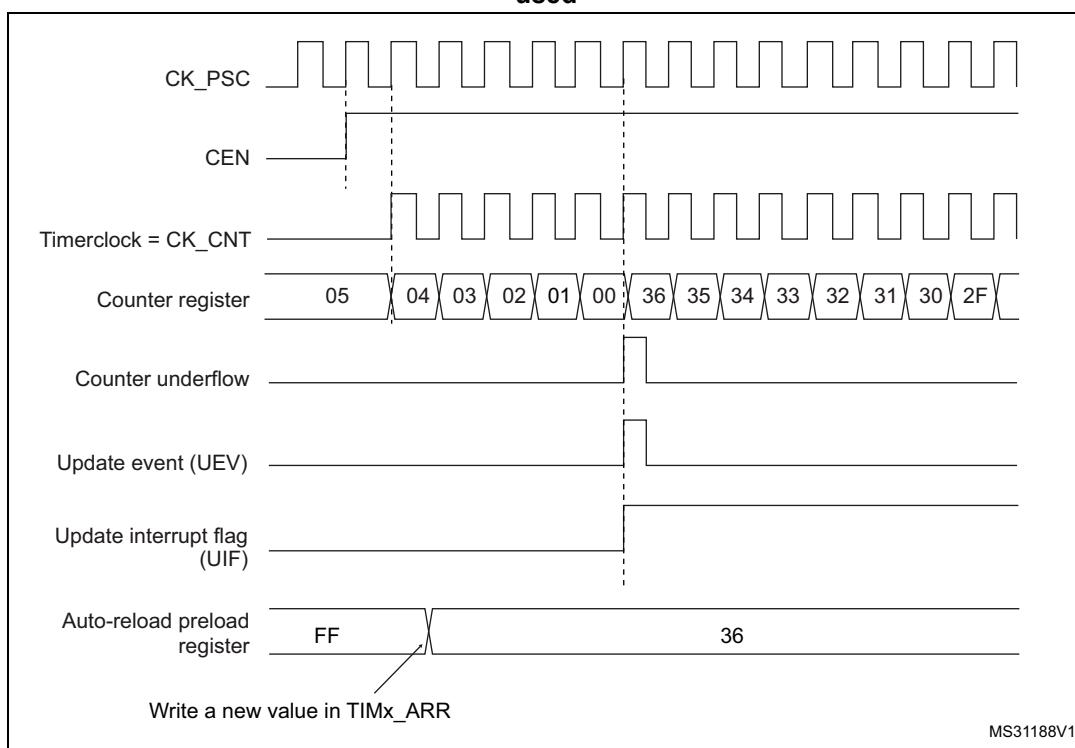
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx\_PSC register).
- The auto-reload active register is updated with the preload value (content of the TIMx\_ARR register). Note that the auto-reload is updated before the counter is reloaded, so that the next period is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

The following figures show some examples of the counter behavior for different clock frequencies when TIMx\_ARR=0x36.

**Figure 120. Counter timing diagram, internal clock divided by 1**



**Figure 121. Counter timing diagram, internal clock divided by 2****Figure 122. Counter timing diagram, internal clock divided by 4**

**Figure 123. Counter timing diagram, internal clock divided by N****Figure 124. Counter timing diagram, Update event when repetition counter is not used**

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

In center-aligned mode, the counter counts from 0 to the auto-reload value (content of the TIMx\_ARR register) – 1, generates a counter overflow event, then counts from the auto-reload value down to 1 and generates a counter underflow event. Then it restarts counting from 0.

Center-aligned mode is active when the CMS bits in TIMx\_CR1 register are not equal to '00'. The Output compare interrupt flag of channels configured in output is set when: the counter counts down (Center aligned mode 1, CMS = "01"), the counter counts up (Center aligned mode 2, CMS = "10") the counter counts up and down (Center aligned mode 3, CMS = "11").

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

The update event can be generated at each counter overflow and at each counter underflow or by setting the UG bit in the TIMx\_EGR register (by software or by using the slave mode controller) also generates an update event. In this case, the counter restarts counting from 0, as well as the counter of the prescaler.

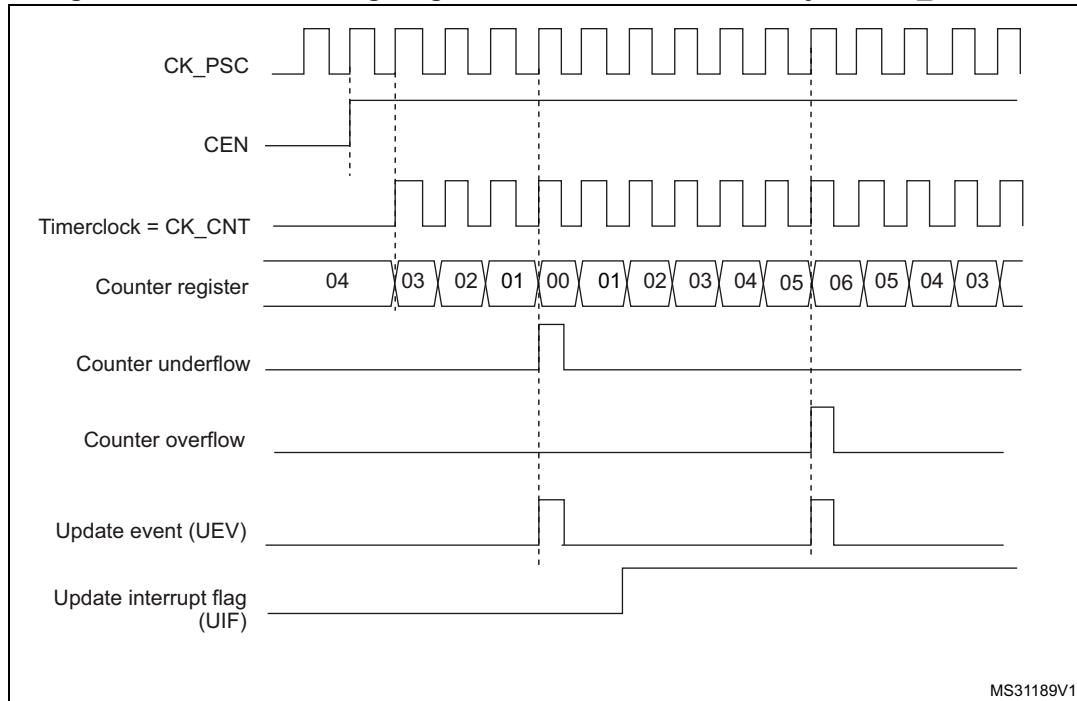
The UEV update event can be disabled by software by setting the UDIS bit in TIMx\_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until the UDIS bit has been written to 0. However, the counter continues counting up and down, based on the current auto-reload value.

In addition, if the URS bit (update request selection) in TIMx\_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupt when clearing the counter on the capture event.

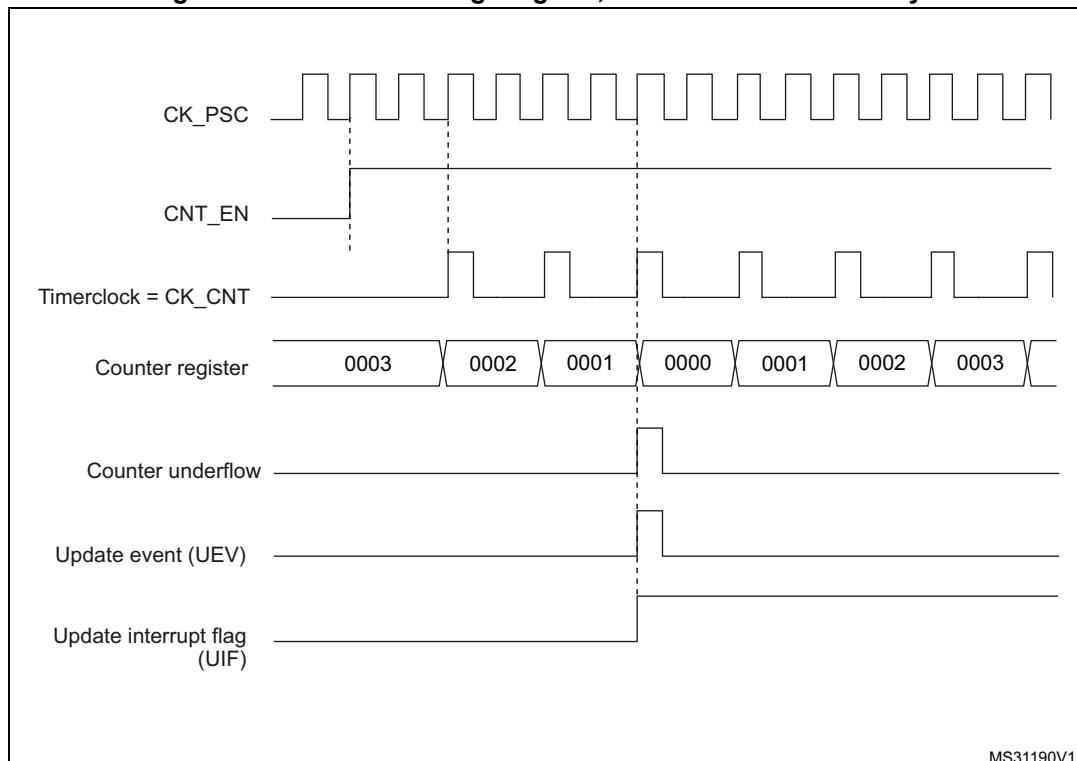
When an update event occurs, all the registers are updated and the update flag (UIF bit in TIMx\_SR register) is set (depending on the URS bit):

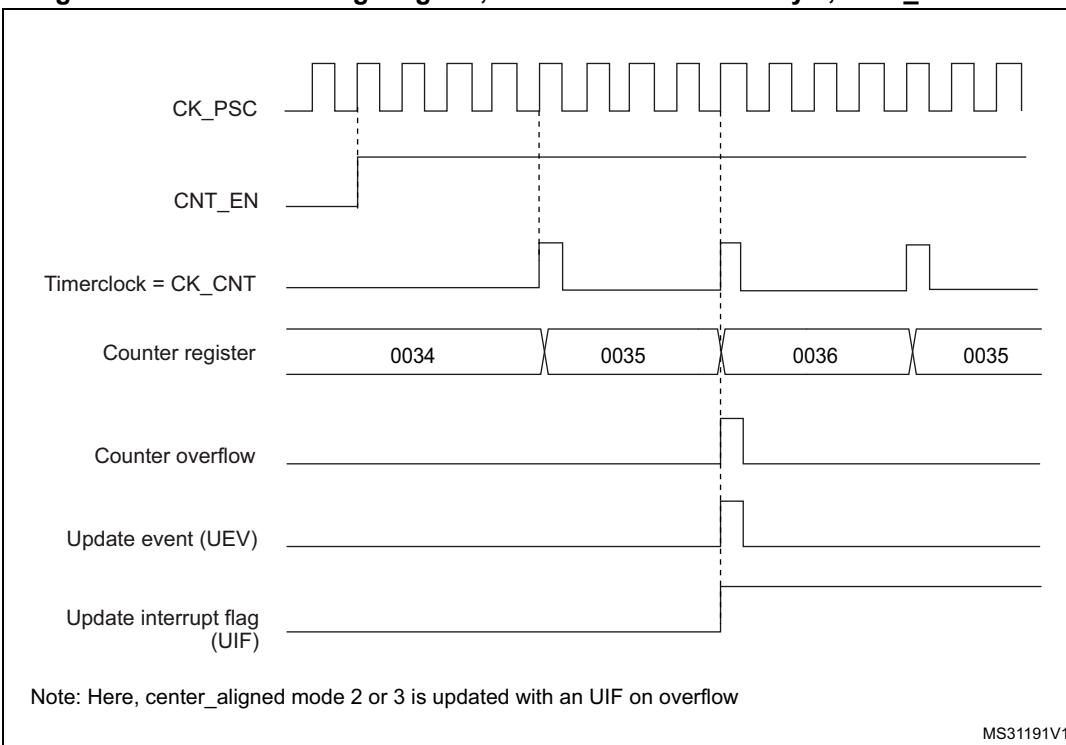
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx\_PSC register).
- The auto-reload active register is updated with the preload value (content of the TIMx\_ARR register). Note that if the update source is a counter overflow, the auto-reload is updated before the counter is reloaded, so that the next period is the expected one (the counter is loaded with the new value).

The following figures show some examples of the counter behavior for different clock frequencies.

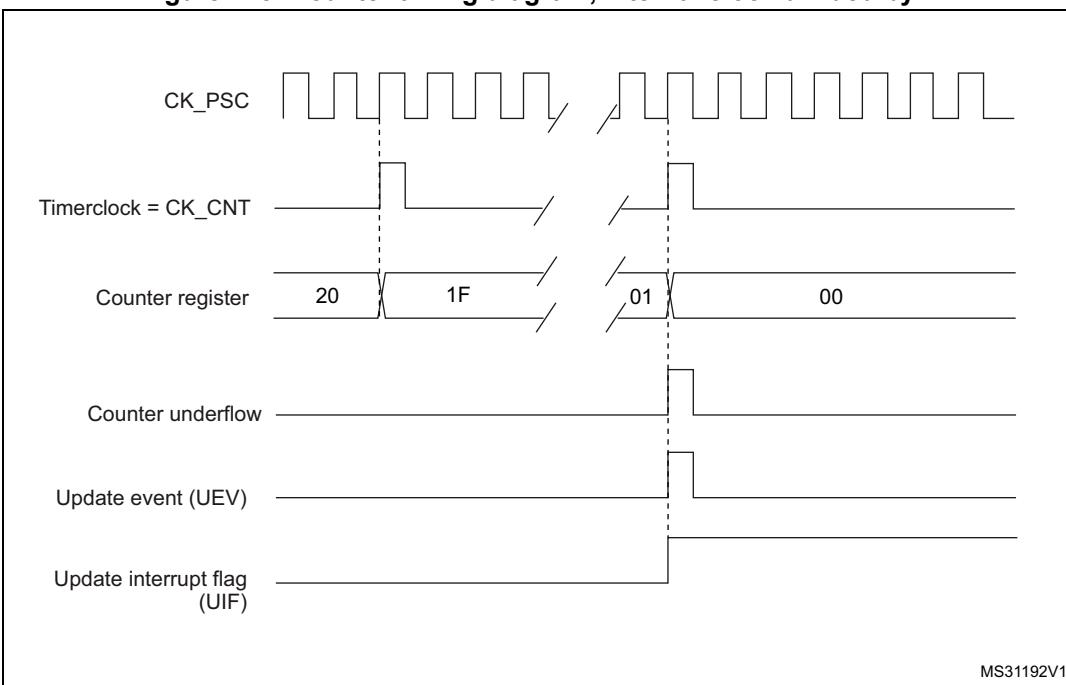
**Figure 125. Counter timing diagram, internal clock divided by 1, TIMx\_ARR=0x6**

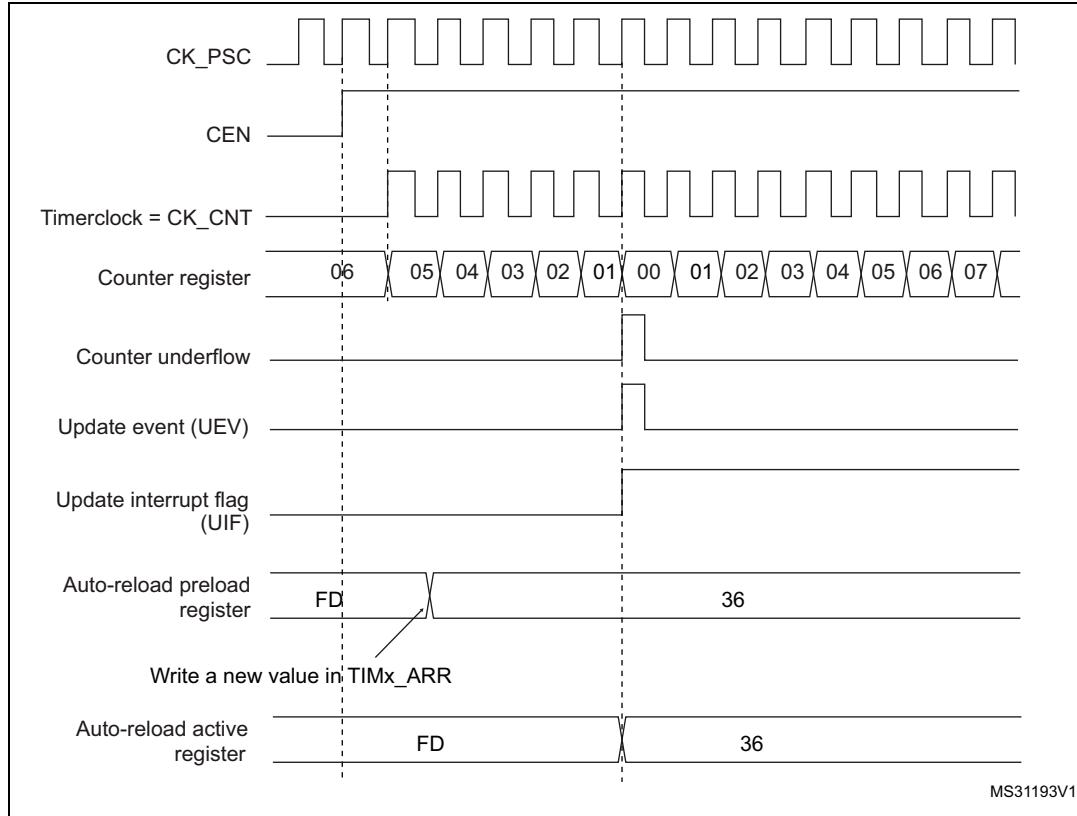
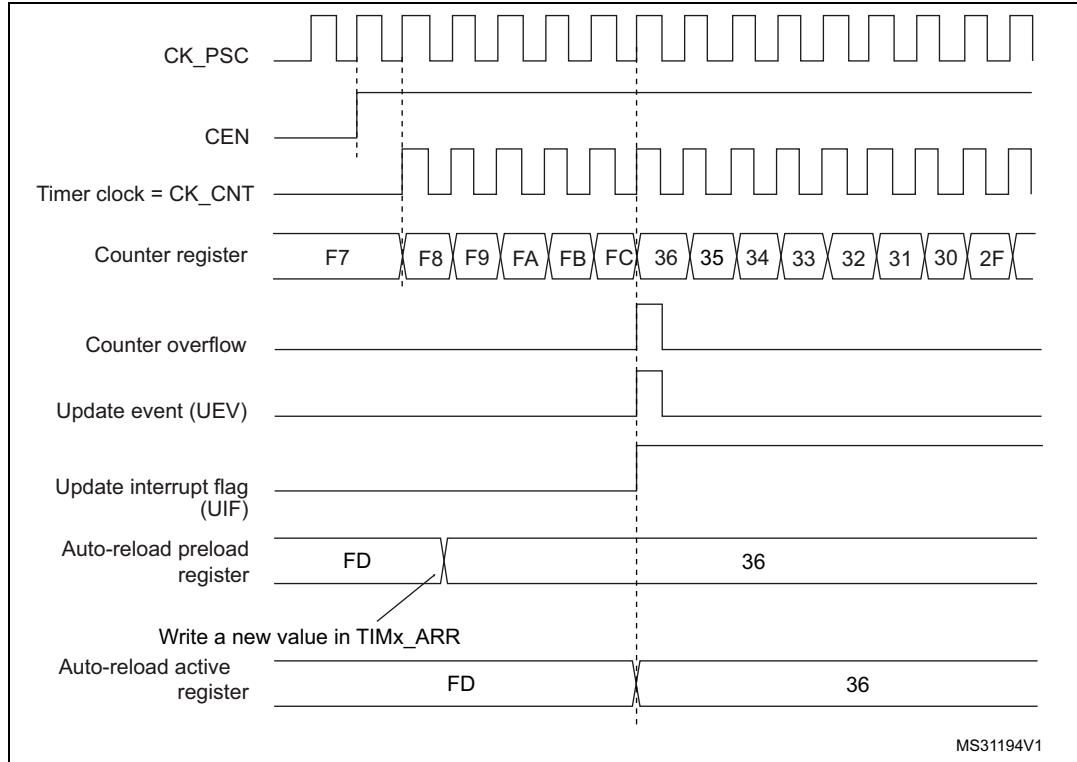
1. Here, center-aligned mode 1 is used (for more details refer to [Section 18.4.1: TIM2 and TIM3 control register 1 \(TIM2\\_CR1 and TIM3\\_CR1\) on page 446](#)).

**Figure 126. Counter timing diagram, internal clock divided by 2**

**Figure 127. Counter timing diagram, internal clock divided by 4, TIMx\_ARR=0x36**

1. Center-aligned mode 2 or 3 is used with an UIF on overflow.

**Figure 128. Counter timing diagram, internal clock divided by N**

**Figure 129. Counter timing diagram, Update event with ARPE=1 (counter underflow)****Figure 130. Counter timing diagram, Update event with ARPE=1 (counter overflow)**

### 18.3.3 Clock sources

The counter clock can be provided by the following clock sources:

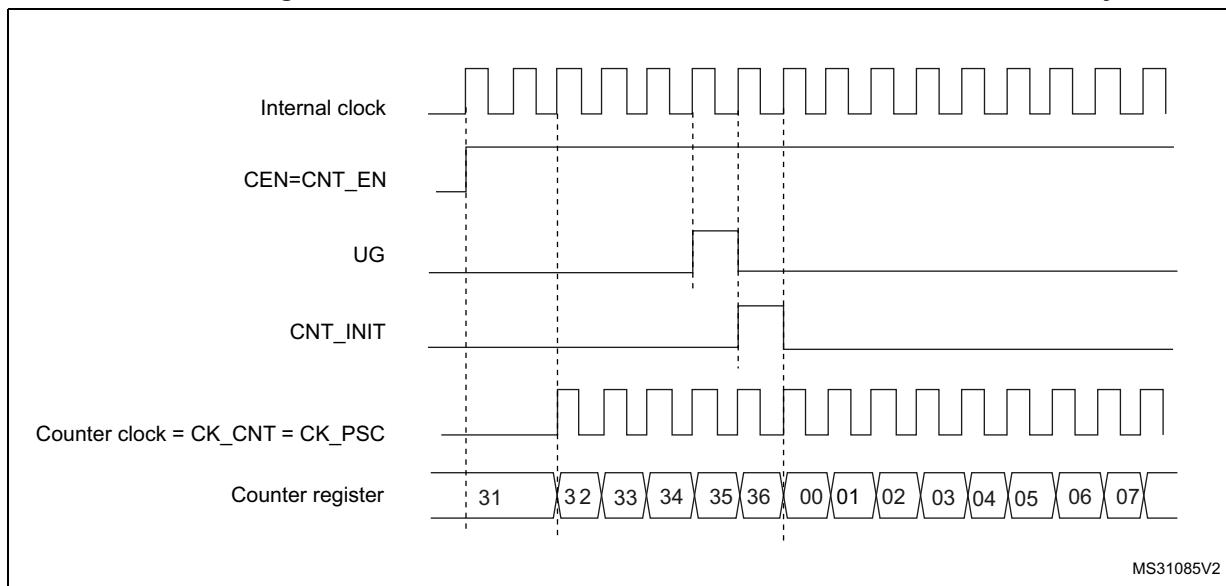
- Internal clock (CK\_INT)
- External clock mode1: external input pin (TIx)
- External clock mode2: external trigger input (ETR)
- Internal trigger inputs (ITRx): using one timer as prescaler for another timer, for example, Timer 1 can be configured to act as a prescaler for Timer 2. Refer to : [Using one timer as prescaler for another on page 440](#) for more details.

#### Internal clock source (CK\_INT)

If the slave mode controller is disabled (SMS=000 in the TIMx\_SMCR register), then the CEN, DIR (in the TIMx\_CR1 register) and UG bits (in the TIMx\_EGR register) are actual control bits and can be changed only by software (except UG which remains cleared automatically). As soon as the CEN bit is written to 1, the prescaler is clocked by the internal clock CK\_INT.

*Figure 131* shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

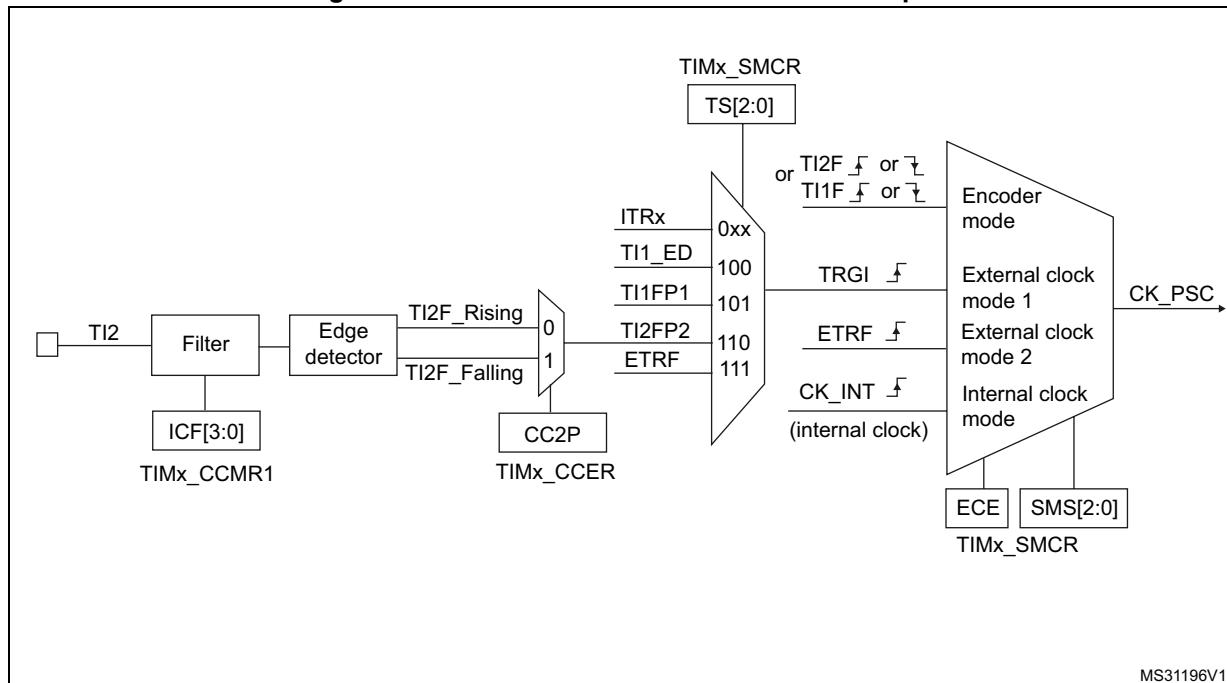
**Figure 131. Control circuit in normal mode, internal clock divided by 1**



#### External clock source mode 1

This mode is selected when SMS=111 in the TIMx\_SMCR register. The counter can count at each rising or falling edge on a selected input.

Figure 132. TI2 external clock connection example



MS31196V1

For example, to configure the upcounter to count in response to a rising edge on the TI2 input, use the following procedure:

1. Configure channel 2 to detect rising edges on the TI2 input by writing CC2S= '01 in the TIMx\_CCMR1 register.
2. Configure the input filter duration by writing the IC2F[3:0] bits in the TIMx\_CCMR1 register (if no filter is needed, keep IC2F=0000).

**Note:**

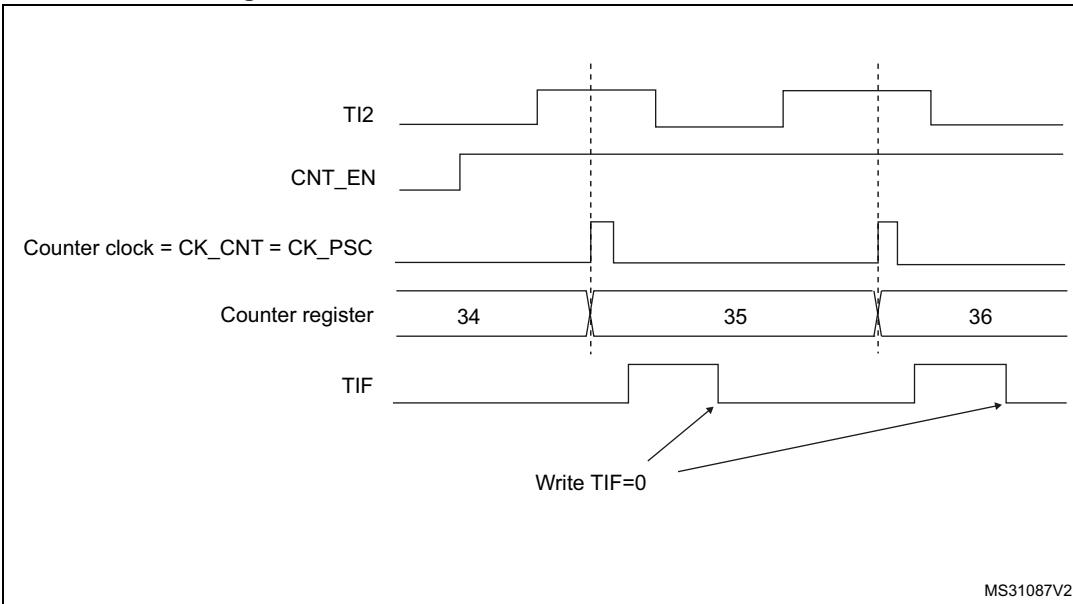
- The capture prescaler is not used for triggering, so it does not need to be configured.*
3. Select rising edge polarity by writing CC2P=0 and CC2NP=0 in the TIMx\_CCER register.
  4. Configure the timer in external clock mode 1 by writing SMS=111 in the TIMx\_SMCR register.
  5. Select TI2 as the input source by writing TS=110 in the TIMx\_SMCR register.
  6. Enable the counter by writing CEN=1 in the TIMx\_CR1 register.

For code example refer to the Appendix section [A.9.2: Up counter on each 2 ETR rising edges code example](#).

When a rising edge occurs on TI2, the counter counts once and the TIF flag is set.

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

Figure 133. Control circuit in external clock mode 1



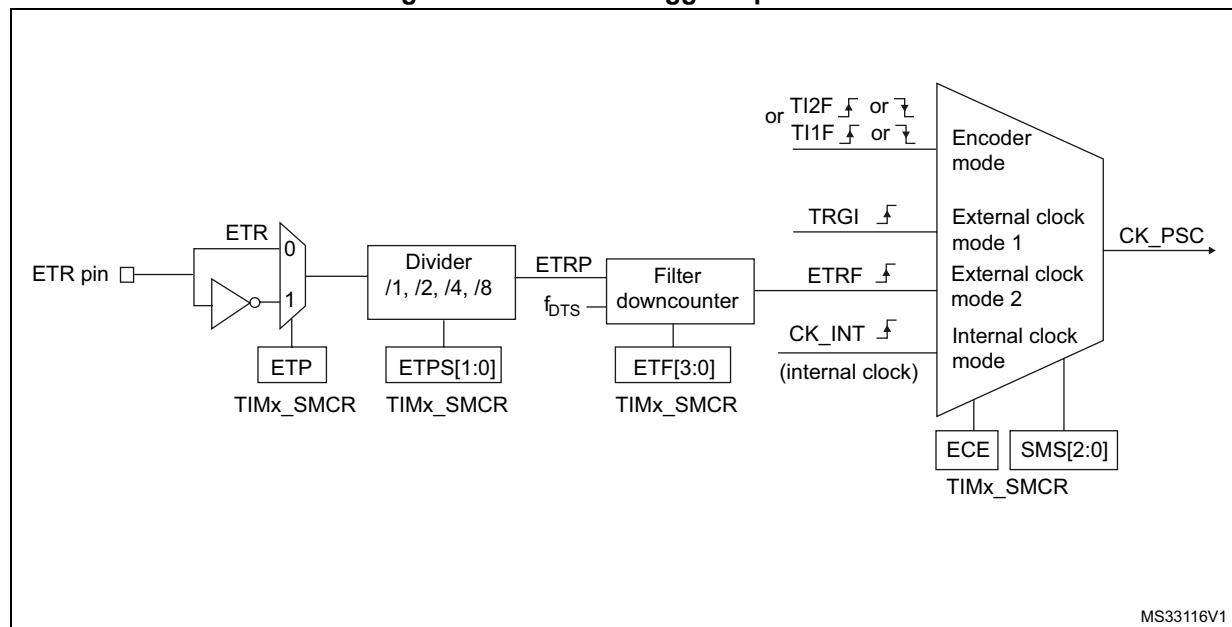
### External clock source mode 2

This mode is selected by writing ECE=1 in the TIMx\_SMCR register.

The counter can count at each rising or falling edge on the external trigger input ETR.

The [Figure 134](#) gives an overview of the external trigger input block.

Figure 134. External trigger input block



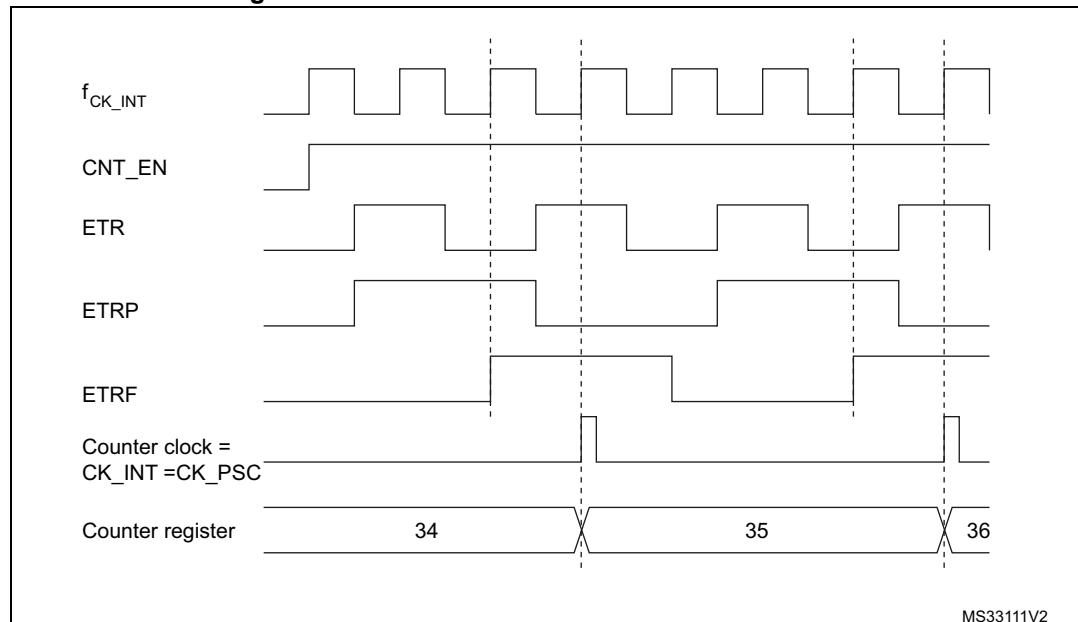
For example, to configure the upcounter to count each 2 rising edges on ETR, use the following procedure:

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

The counter counts once each 2 ETR rising edges.

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

**Figure 135. Control circuit in external clock mode 2**



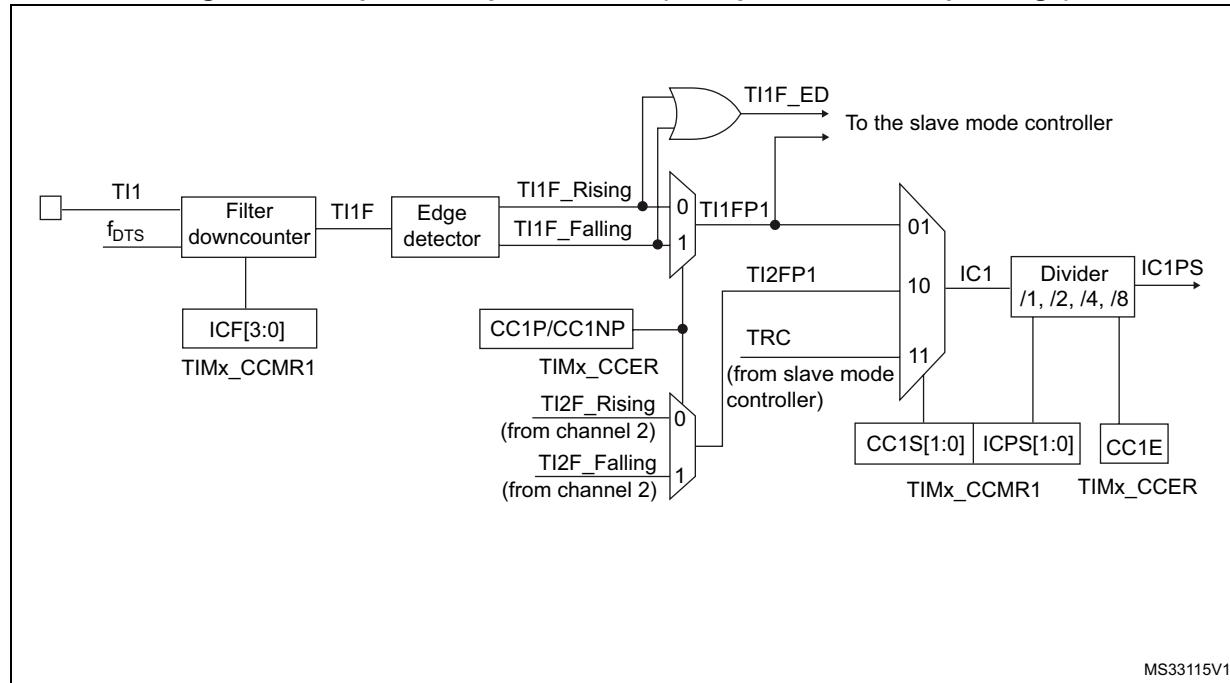
#### 18.3.4 Capture/compare channels

Each Capture/Compare channel is built around a capture/compare register (including a shadow register), a input stage for capture (with digital filter, multiplexing and prescaler) and an output stage (with comparator and output control).

The following figure gives an overview of one Capture/Compare channel.

The input stage samples the corresponding TIx input to generate a filtered signal TIxF. Then, an edge detector with polarity selection generates a signal (TIxFPx) which can be used as trigger input by the slave mode controller or as the capture command. It is prescaled before the capture register (ICxPS).

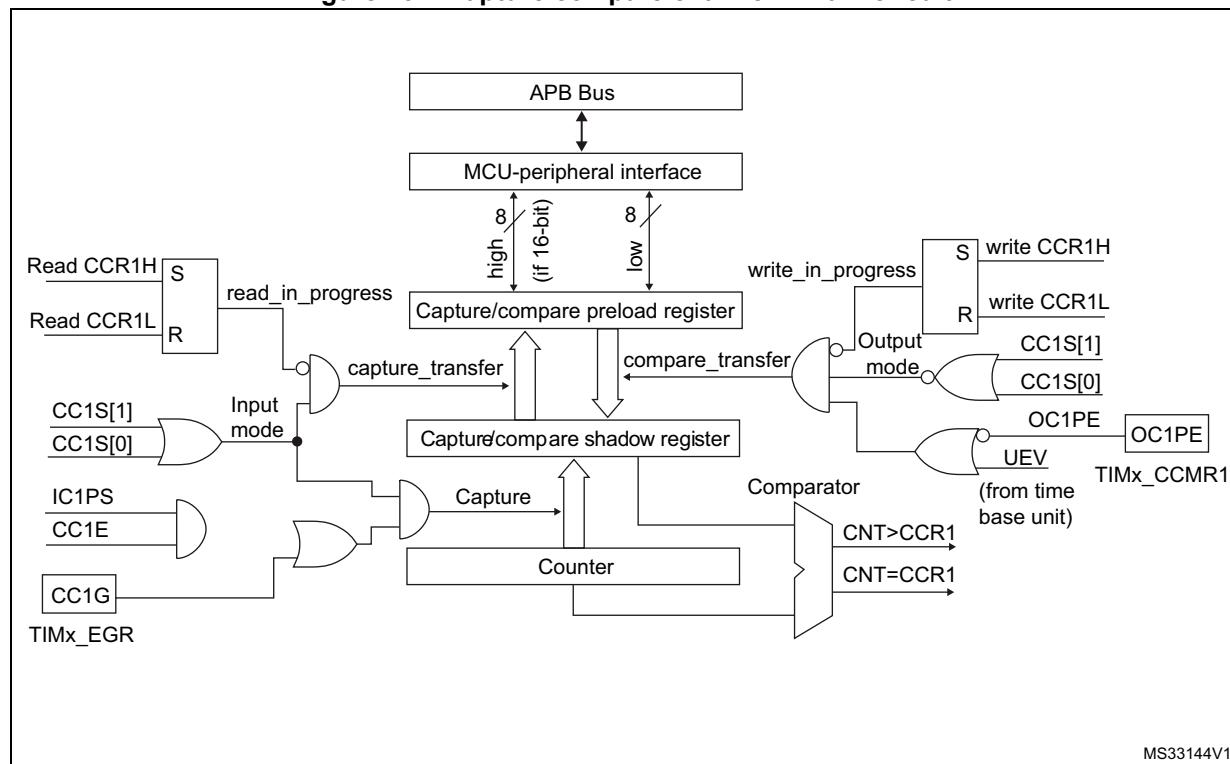
Figure 136. Capture/compare channel (example: channel 1 input stage)



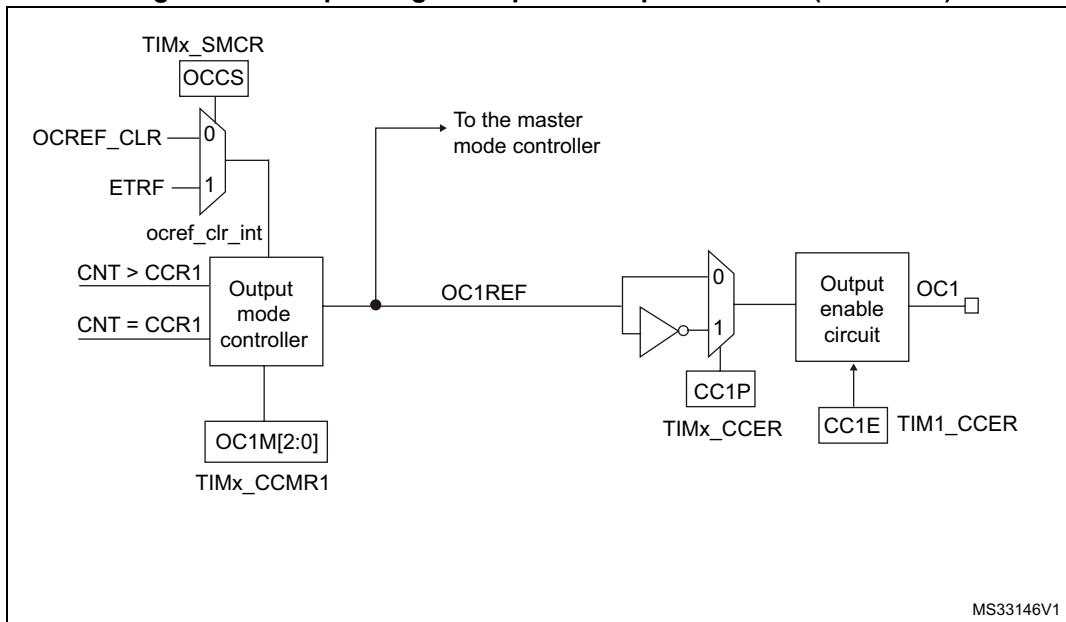
MS33115V1

The output stage generates an intermediate waveform which is then used for reference: OCxRef (active high). The polarity acts at the end of the chain.

Figure 137. Capture/compare channel 1 main circuit



MS33144V1

**Figure 138. Output stage of capture/compare channel (channel 1)**

MS33146V1

The capture/compare block is made of one preload register and one shadow register. Write and read always access the preload register.

In capture mode, captures are actually done in the shadow register, which is copied into the preload register.

In compare mode, the content of the preload register is copied into the shadow register which is compared to the counter.

### 18.3.5 Input capture mode

In Input capture mode, the Capture/Compare Registers (TIMx\_CCRx) are used to latch the value of the counter after a transition detected by the corresponding ICx signal. When a capture occurs, the corresponding CCxIF flag (TIMx\_SR register) is set and an interrupt or a DMA request can be sent if they are enabled. If a capture occurs while the CCxIF flag was already high, then the over-capture flag CCxOF (TIMx\_SR register) is set. CCxIF can be cleared by software by writing it to 0 or by reading the captured data stored in the TIMx\_CCRx register. CCxOF is cleared when it is written with 0.

The following example shows how to capture the counter value in TIMx\_CCR1 when TI1 input rises. To do this, use the following procedure:

- Select the active input: TIMx\_CCR1 must be linked to the TI1 input, so write the CC1S bits to 01 in the TIMx\_CCMR1 register. As soon as CC1S becomes different from 00, the channel is configured in input and the TIMx\_CCR1 register becomes read-only.
- Program the appropriate input filter duration in relation with the signal connected to the timer (when the input is one of the TIx (ICxF bits in the TIMx\_CCMRx register). Let's imagine that, when toggling, the input signal is not stable during at most 5 internal clock cycles. We must program a filter duration longer than these 5 clock cycles. We can validate a transition on TI1 when 8 consecutive samples with the new level have been

detected (sampled at  $f_{DTS}$  frequency). Then write IC1F bits to 0011 in the TIMx\_CCMR1 register.

- Select the edge of the active transition on the TI1 channel by writing the CC1P and CC1NP bits to 0 in the TIMx\_CCER register (rising edge in this case).
- Program the input prescaler. In our example, we wish the capture to be performed at each valid transition, so the prescaler is disabled (write IC1PS bits to 00 in the TIMx\_CCMR1 register).
- Enable capture from the counter into the capture register by setting the CC1E bit in the TIMx\_CCER register.
- If needed, enable the related interrupt request by setting the CC1IE bit in the TIMx\_DIER register, and/or the DMA request by setting the CC1DE bit in the TIMx\_DIER register.

For code example refer to the Appendix section [A.9.3: Input capture configuration code example](#).

When an input capture occurs:

- The TIMx\_CCR1 register gets the value of the counter on the active transition.
- CC1IF flag is set (interrupt flag). CC1OF is also set if at least two consecutive captures occurred whereas the flag was not cleared.
- An interrupt is generated depending on the CC1IE bit.
- A DMA request is generated depending on the CC1DE bit.

For code example refer to the Appendix section [A.9.4: Input capture data management code example](#).

In order to handle the overcapture, it is recommended to read the data before the overcapture flag. This is to avoid missing an overcapture which could happen after reading the flag and before reading the data.

*Note:* *IC interrupt and/or DMA requests can be generated by software by setting the corresponding CCxG bit in the TIMx\_EGR register.*

### 18.3.6 PWM input mode

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

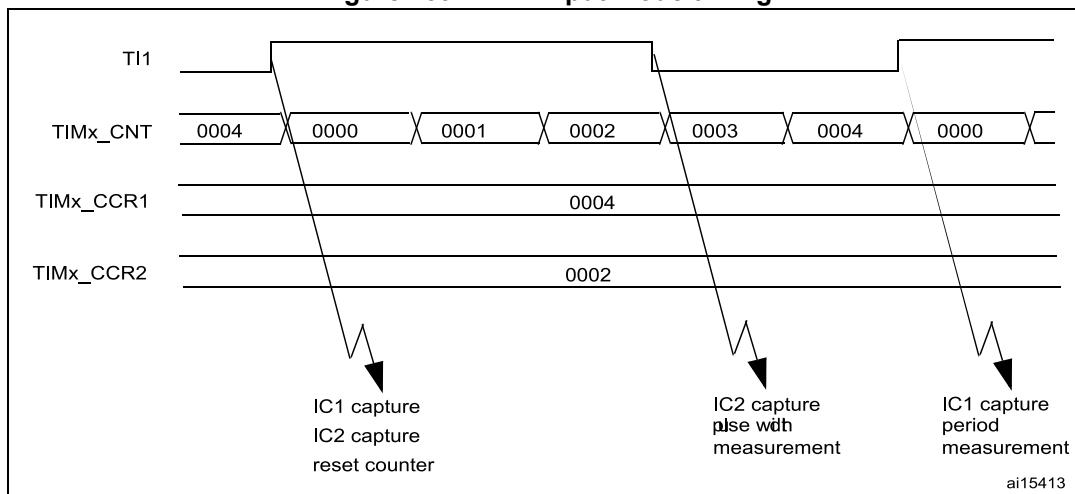
- Two ICx signals are mapped on the same TIx input.
- These 2 ICx signals are active on edges with opposite polarity.
- One of the two TIxFP signals is selected as trigger input and the slave mode controller is configured in reset mode.

For example, one can measure the period (in TIMx\_CCR1 register) and the duty cycle (in TIMx\_CCR2 register) of the PWM applied on TI1 using the following procedure (depending on CK\_INT frequency and prescaler value):

- Select the active input for TIMx\_CCR1: write the CC1S bits to 01 in the TIMx\_CCMR1 register (TI1 selected).
- Select the active polarity for TI1FP1 (used both for capture in TIMx\_CCR1 and counter clear): write the CC1P to '0' and the CC1NP bit to '0' (active on rising edge).
- Select the active input for TIMx\_CCR2: write the CC2S bits to 10 in the TIMx\_CCMR1 register (TI1 selected).
- Select the active polarity for TI1FP2 (used for capture in TIMx\_CCR2): write the CC2P bit to '1' and the CC2NP bit to '0'(active on falling edge).
- Select the valid trigger input: write the TS bits to 101 in the TIMx\_SMCR register (TI1FP1 selected).
- Configure the slave mode controller in reset mode: write the SMS bits to 100 in the TIMx\_SMCR register.
- Enable the captures: write the CC1E and CC2E bits to '1' in the TIMx\_CCER register.

For code example refer to the Appendix section [A.9.5: PWM input configuration code example](#).

**Figure 139. PWM input mode timing**



### 18.3.7 Forced output mode

In output mode (CCxS bits = 00 in the TIMx\_CCMRx register), each output compare signal (OCxREF and then OCx) can be forced to active or inactive level directly by software, independently of any comparison between the output compare register and the counter.

To force an output compare signal (ocxref/OCx) to its active level, one just needs to write 101 in the OCxM bits in the corresponding TIMx\_CCMRx register. Thus ocxref is forced high (OCxREF is always active high) and OCx get opposite value to CCxP polarity bit.

e.g.: CCxP=0 (OCx active high) => OCx is forced to high level.

ocxref signal can be forced low by writing the OCxM bits to 100 in the TIMx\_CCMRx register.

Anyway, the comparison between the TIMx\_CCRx shadow register and the counter is still performed and allows the flag to be set. Interrupt and DMA requests can be sent accordingly. This is described in the Output Compare Mode section.

### 18.3.8 Output compare mode

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

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

- Assigns the corresponding output pin to a programmable value defined by the output compare mode (OCxM bits in the TIMx\_CCMRx register) and the output polarity (CCxP bit in the TIMx\_CCER register). The output pin can keep its level (OCXM=000), be set active (OCXM=001), be set inactive (OCXM=010) or can toggle (OCXM=011) on match.
- Sets a flag in the interrupt status register (CCxIF bit in the TIMx\_SR register).
- Generates an interrupt if the corresponding interrupt mask is set (CCXIE bit in the TIMx\_DIER register).
- Sends a DMA request if the corresponding enable bit is set (CCxDE bit in the TIMx\_DIER register, CCDS bit in the TIMx\_CR2 register for the DMA request selection).

The TIMx\_CCRx registers can be programmed with or without preload registers using the OCxPE bit in the TIMx\_CCMRx register.

In output compare mode, the update event UEV has no effect on ocxref and OCx output. The timing resolution is one count of the counter. Output compare mode can also be used to output a single pulse (in One-pulse mode).

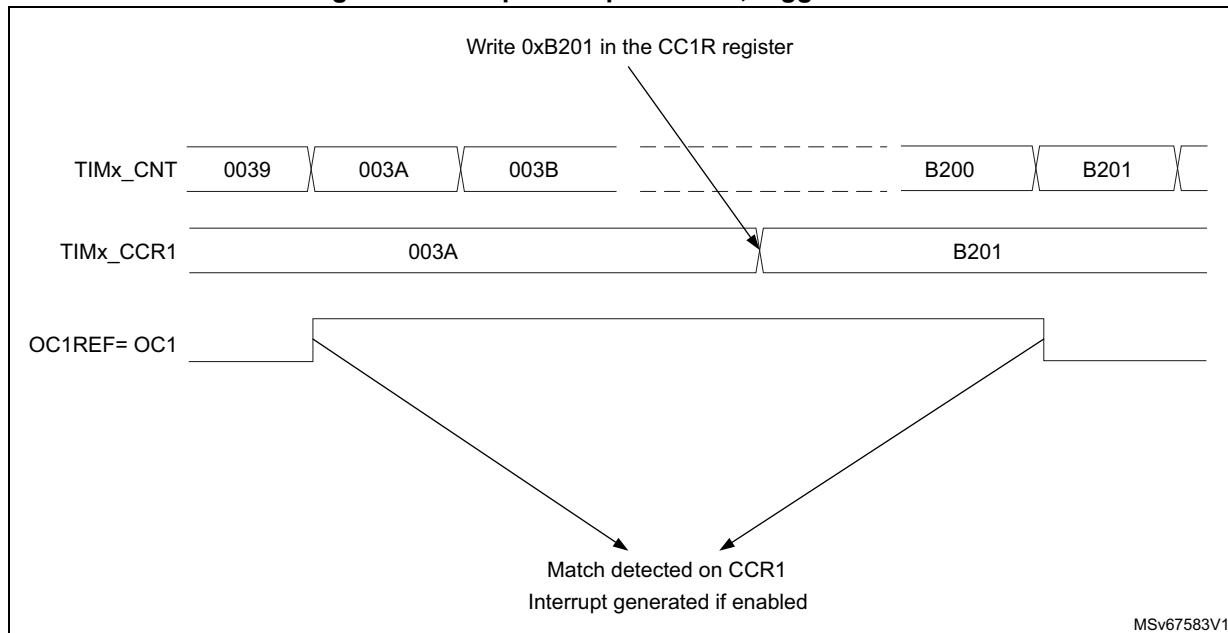
Procedure:

1. Select the counter clock (internal, external, prescaler).
2. Write the desired data in the TIMx\_ARR and TIMx\_CCRx registers.
3. Set the CCxIE and/or CCxDE bits if an interrupt and/or a DMA request is to be generated.
4. Select the output mode. For example, one must write OCxM=011, OCxPE=0, CCxP=0 and CCxE=1 to toggle OCx output pin when CNT matches CCRx, CCRx preload is not used, OCx is enabled and active high.
5. Enable the counter by setting the CEN bit in the TIMx\_CR1 register.

For code example refer to the Appendix section [A.9.7: Output compare configuration code example](#).

The TIMx\_CCRx register can be updated at any time by software to control the output waveform, provided that the preload register is not enabled (OCxPE=0, else TIMx\_CCRx shadow register is updated only at the next update event UEV). An example is given in [Figure 140](#).

**Figure 140. Output compare mode, toggle on OC1**



### 18.3.9 PWM mode

Pulse width modulation mode allows to generate a signal with a frequency determined by the value of the TIMx\_ARR register and a duty cycle determined by the value of the TIMx\_CCRx register.

The PWM mode can be selected independently on each channel (one PWM per OCx output) by writing 110 (PWM mode 1) or '111 (PWM mode 2) in the OCxM bits in the TIMx\_CCMRx register. The corresponding preload register must be enabled by setting the OCxPE bit in the TIMx\_CCMRx register, and eventually the auto-reload preload register (in upcounting or center-aligned modes) by setting the ARPE bit in the TIMx\_CR1 register.

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

OCx polarity is software programmable using the CCxP bit in the TIMx\_CCER register. It can be programmed as active high or active low. OCx output is enabled by the CCxE bit in the TIMx\_CCER register. Refer to the TIMx\_CCERx register description for more details.

In PWM mode (1 or 2), TIMx\_CNT and TIMx\_CCRx are always compared to determine whether TIMx\_CCRx ≤ TIMx\_CNT or TIMx\_CNT ≤ TIMx\_CCRx (depending on the direction of the counter). However, to comply with the OCREF\_CLR functionality (OCREF can be

cleared by an external event through the ETR signal until the next PWM period), the OCREF signal is asserted only:

- When the result of the comparison changes, or
- When the output compare mode (OCxM bits in TIMx\_CCMRx register) switches from the “frozen” configuration (no comparison, OCxM='000) to one of the PWM modes (OCxM='110 or '111).

This forces the PWM by software while the timer is running.

The timer is able to generate PWM in edge-aligned mode or center-aligned mode depending on the CMS bits in the TIMx\_CR1 register.

### PWM edge-aligned mode

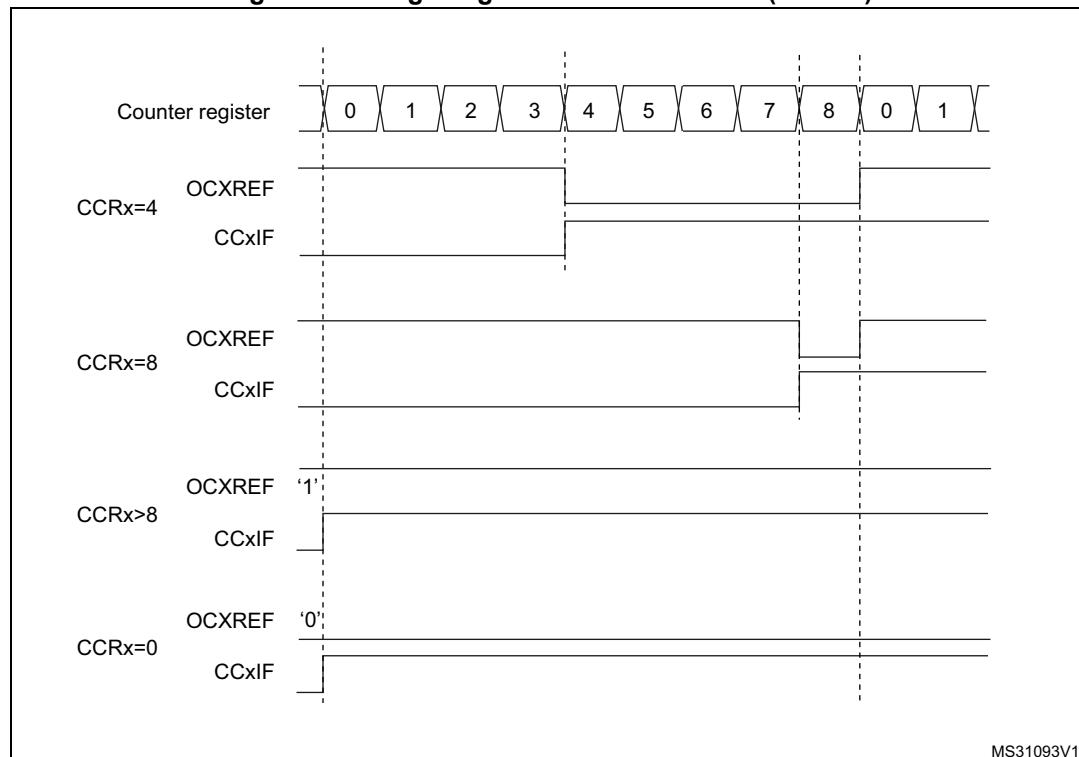
Upcounting configuration

Upcounting is active when the DIR bit in the TIMx\_CR1 register is low. Refer to the [Section : Upcounting mode on page 407](#).

In the following example, we consider PWM mode 1. The reference PWM signal OCxREF is high as long as TIMx\_CNT < TIMx\_CCRx else it becomes low. If the compare value in TIMx\_CCRx is greater than the auto-reload value (in TIMx\_ARR) then OCxREF is held at '1'. If the compare value is 0 then OCxREF is held at '0'. [Figure 141](#) shows some edge-aligned PWM waveforms in an example where TIMx\_ARR=8.

For code example refer to the Appendix section [A.9.9: Center-aligned PWM configuration example](#).

**Figure 141. Edge-aligned PWM waveforms (ARR=8)**



MS31093V1

## Downcounting configuration

Downcounting is active when DIR bit in TIMx\_CR1 register is high. Refer to [Downcounting mode on page 411](#)

In PWM mode 1, the reference signal ocxref is low as long as TIMx\_CNT>TIMx\_CCRx else it becomes high. If the compare value in TIMx\_CCRx is greater than the auto-reload value in TIMx\_ARR, then ocxref is held at '1. 0% PWM is not possible in this mode.

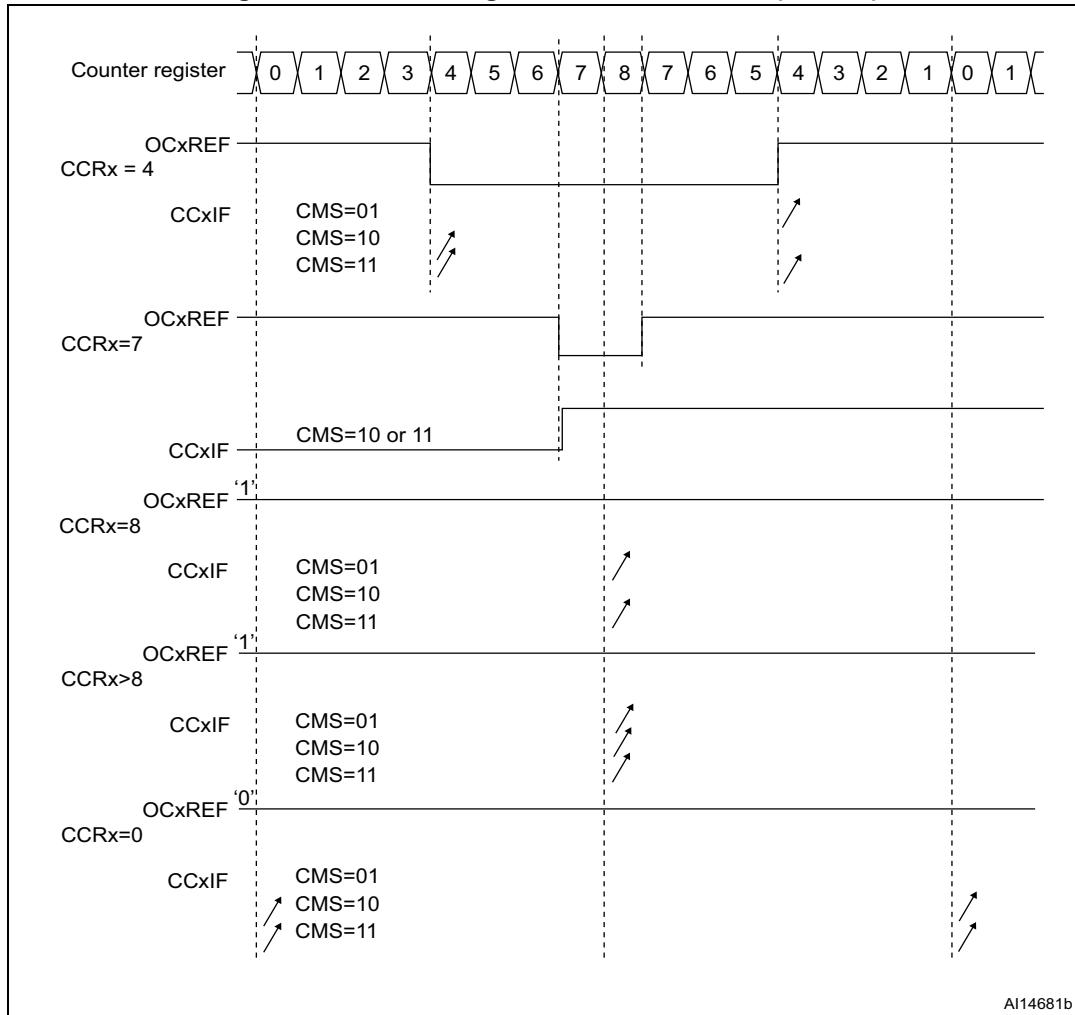
## PWM center-aligned mode

Center-aligned mode is active when the CMS bits in TIMx\_CR1 register are different from '00 (all the remaining configurations having the same effect on the ocxref/OCx signals). The compare flag is set when the counter counts up, when it counts down or both when it counts up and down depending on the CMS bits configuration. The direction bit (DIR) in the TIMx\_CR1 register is updated by hardware and must not be changed by software. Refer to the [Center-aligned mode \(up/down counting\) on page 414](#).

*Figure 142* shows some center-aligned PWM waveforms in an example where:

- TIMx\_ARR=8,
- PWM mode is the PWM mode 1,
- The flag is set when the counter counts down corresponding to the center-aligned mode 1 selected for CMS=01 in TIMx\_CR1 register.

Figure 142. Center-aligned PWM waveforms (ARR=8)



Hints on using center-aligned mode:

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

### 18.3.10 One-pulse mode

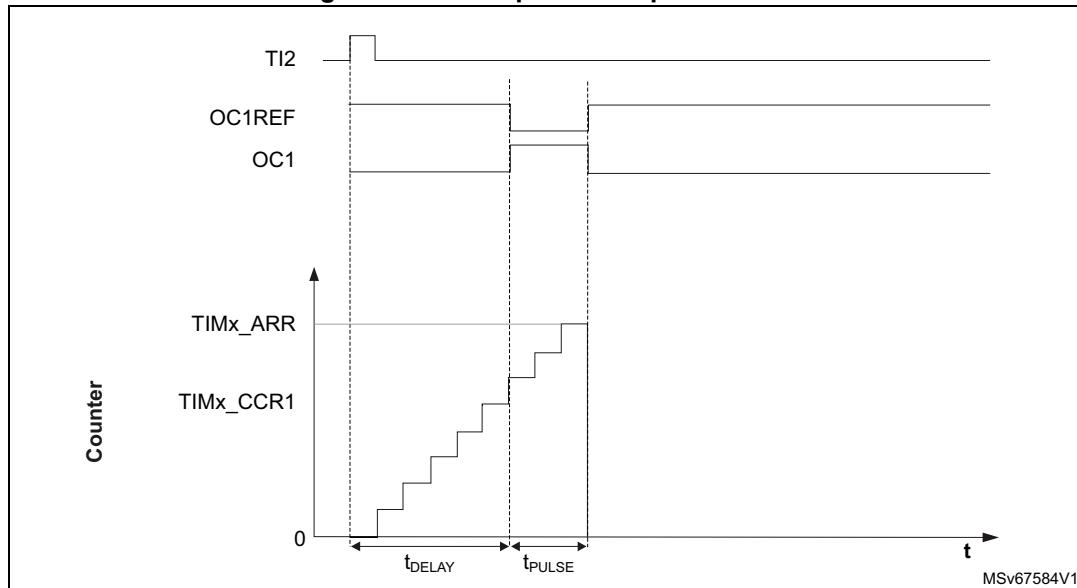
One-pulse mode (OPM) is a particular case of the previous modes. It allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length after a programmable delay.

Starting the counter can be controlled through the slave mode controller. Generating the waveform can be done in output compare mode or PWM mode. One-pulse mode is selected by setting the OPM bit in the TIMx\_CR1 register. This makes the counter stop automatically at the next update event UEV.

A pulse can be correctly generated only if the compare value is different from the counter initial value. Before starting (when the timer is waiting for the trigger), the configuration must be:

- In upcounting: CNT<CCR<sub>x</sub> ≤ ARR (in particular, 0<CCR<sub>x</sub>),
- In downcounting: CNT>CCR<sub>x</sub>.

**Figure 143. Example of one-pulse mode**



For example one may want to generate a positive pulse on OC1 with a length of  $t_{PULSE}$  and after a delay of  $t_{DELAY}$  as soon as a positive edge is detected on the TI2 input pin.

Use TI2FP2 as trigger 1:

- Map TI2FP2 on TI2 by writing CC2S=01 in the TIMx\_CCMR1 register.
- TI2FP2 must detect a rising edge, write CC2P=0 and CC2NP='0' in the TIMx\_CCER register.
- Configure TI2FP2 as trigger for the slave mode controller (TRGI) by writing TS=110 in the TIMx\_SMCR register.
- TI2FP2 is used to start the counter by writing SMS to '110 in the TIMx\_SMCR register (trigger mode).

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

- The  $t_{DELAY}$  is defined by the value written in the TIMx\_CCR1 register.
- The  $t_{PULSE}$  is defined by the difference between the auto-reload value and the compare value ( $\text{TIMx\_ARR} - \text{TIMx\_CCR1} + 1$ ).
- Let's say one want to build a waveform with a transition from '0' to '1' when a compare match occurs and a transition from '1' to '0' when the counter reaches the auto-reload value. To do this PWM mode 2 must be enabled by writing  $\text{OC1M}=111$  in the TIMx\_CCMR1 register. Optionally the preload registers can be enabled by writing  $\text{OC1PE}=1$  in the TIMx\_CCMR1 register and  $\text{ARPE}$  in the TIMx\_CR1 register. In this case one has to write the compare value in the TIMx\_CCR1 register, the auto-reload value in the TIMx\_ARR register, generate an update by setting the UG bit and wait for external trigger event on TI2. CC1P is written to '0' in this example.

In our example, the DIR and CMS bits in the TIMx\_CR1 register should be low.

For code example refer to the Appendix section [A.9.16: One-Pulse mode code example](#).

Since only 1 pulse (Single mode) is needed, a 1 must be written in the OPM bit in the TIMx\_CR1 register to stop the counter at the next update event (when the counter rolls over from the auto-reload value back to 0). When OPM bit in the TIMx\_CR1 register is set to '0', so the Repetitive Mode is selected.

#### Particular case: OCx fast enable

In One-pulse mode, the edge detection on TIx input set the CEN bit which enables the counter. Then the comparison between the counter and the compare value makes the output toggle. But several clock cycles are needed for these operations and it limits the minimum delay  $t_{DELAY}$  min we can get.

If one wants to output a waveform with the minimum delay, the OCxFE bit can be set in the TIMx\_CCMRx register. Then OCxRef (and OCx) are forced in response to the stimulus, without taking in account the comparison. Its new level is the same as if a compare match had occurred. OCxFE acts only if the channel is configured in PWM1 or PWM2 mode.

For code example refer to the part of code, conditioned by  $\text{PULSE\_WITHOUT\_DELAY} > 0$  in the Appendix section [A.9.16: One-Pulse mode code example](#).

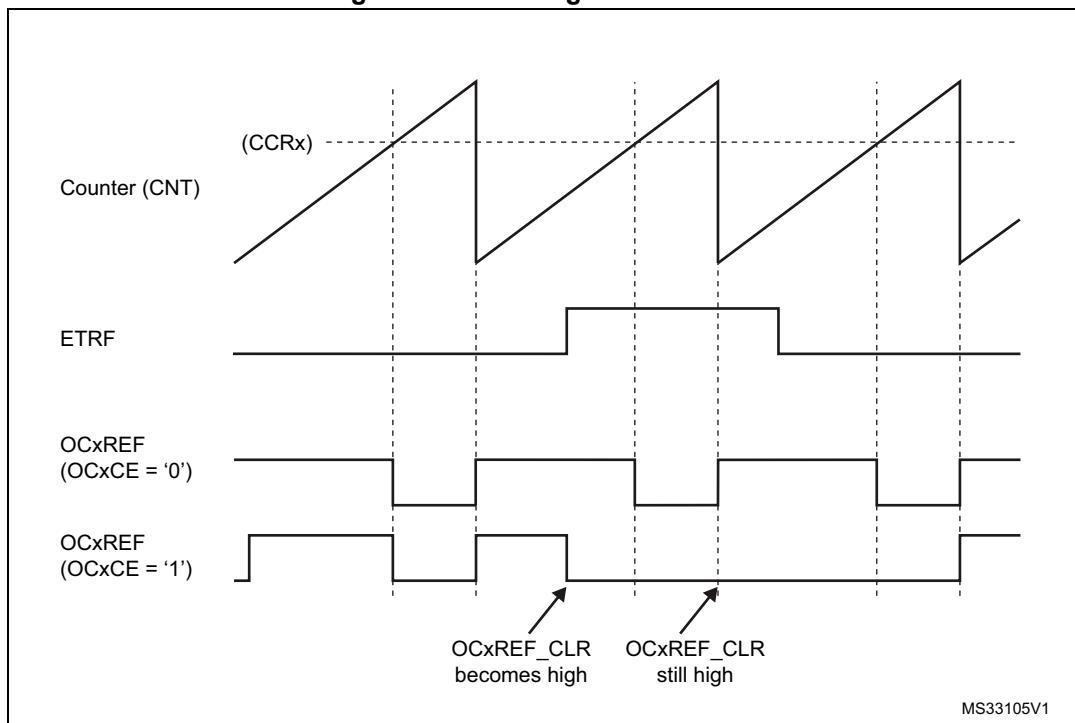
### 18.3.11 Clearing the OCxREF signal on an external event

1. The external trigger prescaler should be kept off: bits ETPS[1:0] in the TIMx\_SMCR register are cleared to 00.
2. The external clock mode 2 must be disabled: bit ECE in the TIM1\_SMCR register is cleared to 0.
3. The external trigger polarity (ETP) and the external trigger filter (ETF) can be configured according to the application's needs.

For code example refer to the Appendix section [A.9.10: ETR configuration to clear OCxREF code example](#).

*Figure 144* shows the behavior of the OCxREF signal when the ETRF input becomes high, for both values of the OCxCE enable bit. In this example, the timer TIMx is programmed in PWM mode.

Figure 144. Clearing TIMx OCxREF



1. In case of a PWM with a 100% duty cycle (if  $CCR_x > ARR$ ), OCxREF is enabled again at the next counter overflow.

### 18.3.12 Encoder interface mode

To select Encoder Interface mode write SMS='001 in the TIMx\_SMCR register if the counter is counting on TI2 edges only, SMS=010 if it is counting on TI1 edges only and SMS=011 if it is counting on both TI1 and TI2 edges.

Select the TI1 and TI2 polarity by programming the CC1P and CC2P bits in the TIMx\_CCER register. CC1NP and CC2NP must be kept cleared. When needed, the input filter can be programmed as well.

The two inputs TI1 and TI2 are used to interface to an incremental encoder. Refer to [Table 65](#). The counter is clocked by each valid transition on TI1FP1 or TI2FP2 (TI1 and TI2 after input filter and polarity selection, TI1FP1=TI1 if not filtered and not inverted, TI2FP2=TI2 if not filtered and not inverted) assuming that it is enabled (CEN bit in TIMx\_CR1 register written to '1'). The sequence of transitions of the two inputs is evaluated and generates count pulses as well as the direction signal. Depending on the sequence the counter counts up or down, the DIR bit in the TIMx\_CR1 register is modified by hardware accordingly. The DIR bit is calculated at each transition on any input (TI1 or TI2), whatever the counter is counting on TI1 only, TI2 only or both TI1 and TI2.

Encoder interface mode acts simply as an external clock with direction selection. This means that the counter just counts continuously between 0 and the auto-reload value in the TIMx\_ARR register (0 to ARR or ARR down to 0 depending on the direction). So the TIMx\_ARR must be configured before starting. In the same way, the capture, compare, prescaler, trigger output features continue to work as normal.

In this mode, the counter is modified automatically following the speed and the direction of the incremental encoder and its content, therefore, always represents the encoder's

position. The count direction correspond to the rotation direction of the connected sensor. The table summarizes the possible combinations, assuming TI1 and TI2 do not switch at the same time.

**Table 65. Counting direction versus encoder signals**

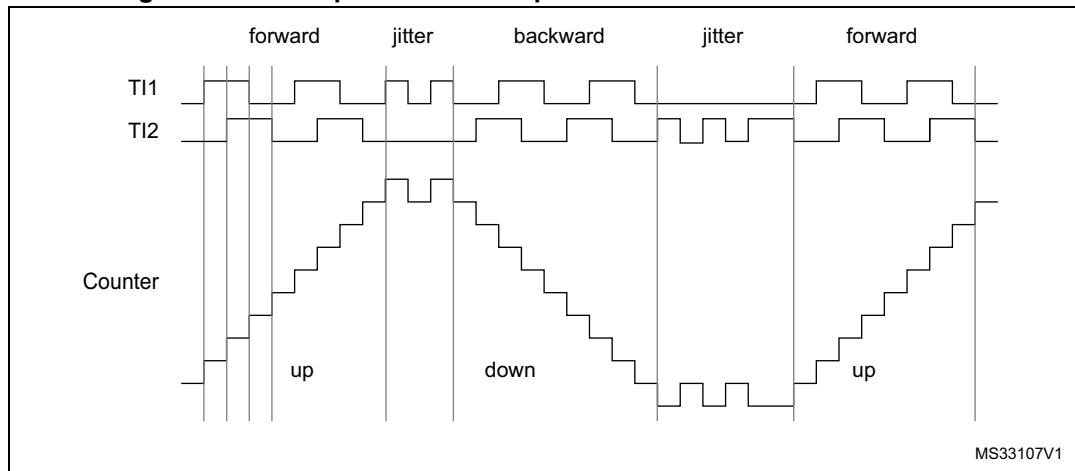
Active edge	Level on opposite signal (TI1FP1 for TI2, TI2FP2 for TI1)	TI1FP1 signal		TI2FP2 signal	
		Rising	Falling	Rising	Falling
Counting on TI1 only	High	Down	Up	No Count	No Count
	Low	Up	Down	No Count	No Count
Counting on TI2 only	High	No Count	No Count	Up	Down
	Low	No Count	No Count	Down	Up
Counting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

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

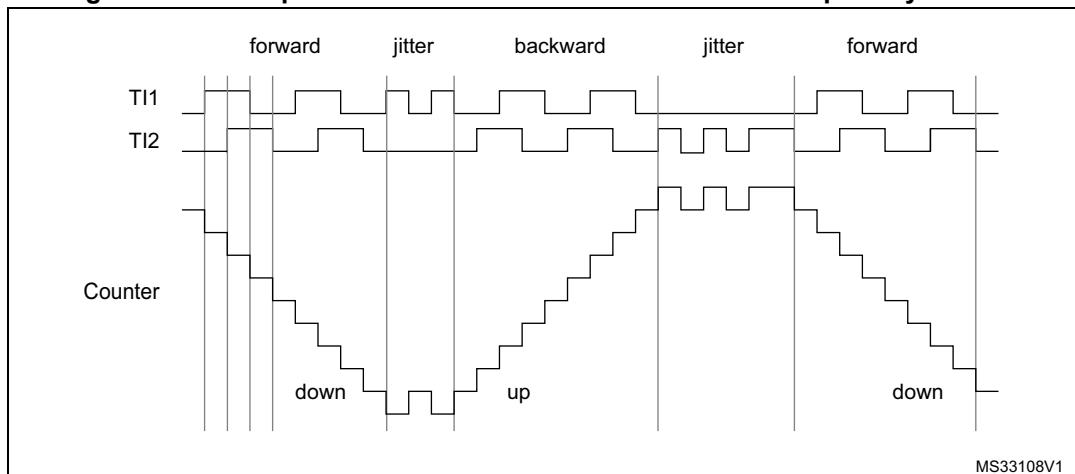
*Figure 145* gives an example of counter operation, showing count signal generation and direction control. It also shows how input jitter is compensated where both edges are selected. This might occur if the sensor is positioned near to one of the switching points. For this example we assume that the configuration is the following:

- CC1S= 01 (TIMx\_CCMR1 register, TI1FP1 mapped on TI1)
- CC2S= 01 (TIMx\_CCMR2 register, TI2FP2 mapped on TI2)
- CC1P=0, CC1NP = '0' (TIMx\_CCER register, TI1FP1 noninverted, TI1FP1=TI1)
- CC2P=0, CC2NP = '0' (TIMx\_CCER register, TI2FP2 noninverted, TI2FP2=TI2)
- SMS= 011 (TIMx\_SMCR register, both inputs are active on both rising and falling edges)
- CEN= 1 (TIMx\_CR1 register, Counter is enabled)

For code example refer to the Appendix section [A.9.10: ETR configuration to clear OCxREF code example](#).

**Figure 145. Example of counter operation in encoder interface mode**

*Figure 146* gives an example of counter behavior when TI1FP1 polarity is inverted (same configuration as above except CC1P=1).

**Figure 146. Example of encoder interface mode with TI1FP1 polarity inverted**

The timer, when configured in Encoder Interface mode provides information on the sensor's current position. Dynamic information can be obtained (speed, acceleration, deceleration) by measuring the period between two encoder events using a second timer configured in capture mode. The output of the encoder which indicates the mechanical zero can be used for this purpose. Depending on the time between two events, the counter can also be read at regular times. This can be done by latching the counter value into a third input capture register if available (then the capture signal must be periodic and can be generated by another timer). When available, it is also possible to read its value through a DMA request generated by a Real-Time clock.

### 18.3.13 Timer input XOR function

The TI1S bit in the TIM1\_CR2 register, allows the input filter of channel 1 to be connected to the output of a XOR gate, combining the three input pins TIMx\_CH1 to TIMx\_CH3.

The XOR output can be used with all the timer input functions such as trigger or input capture.

An example of this feature used to interface Hall sensors is given in [Section 17.3.18 on page 369](#).

### 18.3.14 Timers and external trigger synchronization

The TIMx Timers can be synchronized with an external trigger in several modes: Reset mode, Gated mode and Trigger mode.

#### Slave mode: Reset mode

The counter and its prescaler can be reinitialized in response to an event on a trigger input. Moreover, if the URS bit from the TIMx\_CR1 register is low, an update event UEV is generated. Then all the preloaded registers (TIMx\_ARR, TIMx\_CCRx) are updated.

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

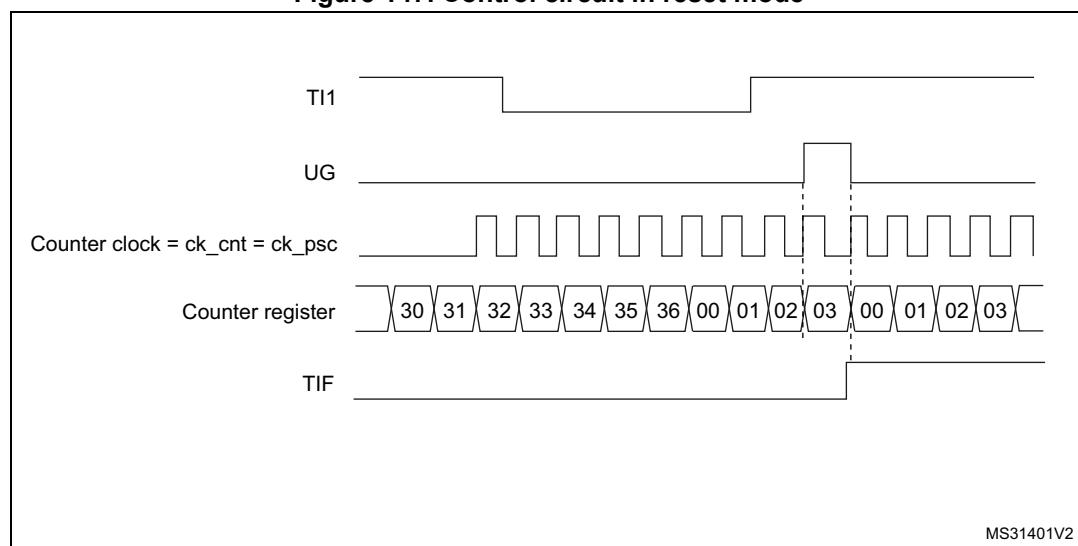
- Configure the channel 1 to detect rising edges on TI1. Configure the input filter duration (in this example, we do not need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. The CC1S bits select the input capture source only, CC1S = 01 in the TIMx\_CCMR1 register. Write CC1P=0 and CC1NP=0 in TIMx\_CCER register to validate the polarity (and detect rising edges only).
- Configure the timer in reset mode by writing SMS=100 in TIMx\_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx\_SMCR register.
- Start the counter by writing CEN=1 in the TIMx\_CR1 register.

For code example refer to the Appendix section [A.9.12: Reset mode code example](#).

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

The following figure shows this behavior when the auto-reload register TIMx\_ARR=0x36. The delay between the rising edge on TI1 and the actual reset of the counter is due to the resynchronization circuit on TI1 input.

**Figure 147. Control circuit in reset mode**



### Slave mode: Gated mode

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

In the following example, the upcounter counts only when TI1 input is low:

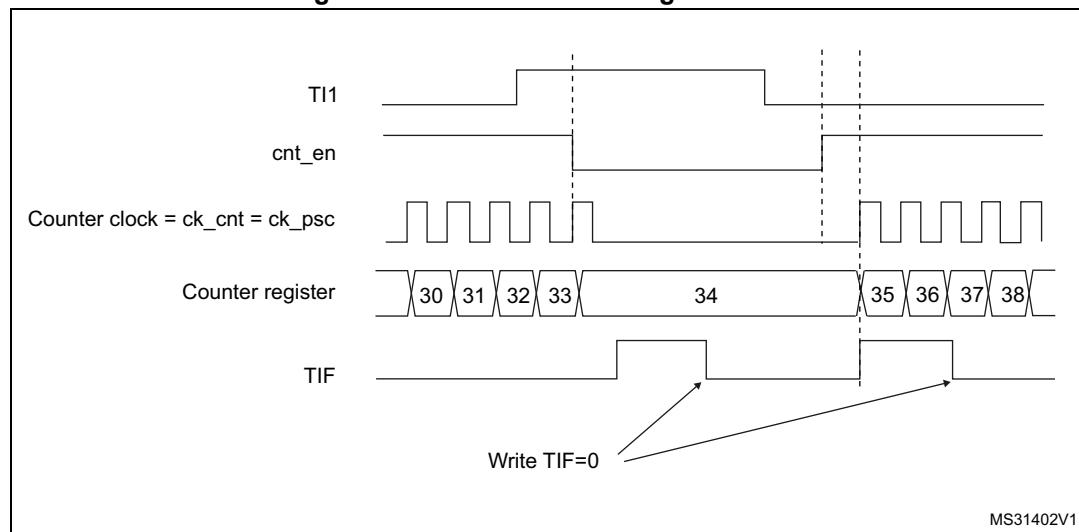
- Configure the channel 1 to detect low levels on TI1. Configure the input filter duration (in this example, we do not need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. The CC1S bits select the input capture source only, CC1S=01 in TIMx\_CCMR1 register. Write CC1P=1 and CC1NP=0 in TIMx\_CCER register to validate the polarity (and detect low level only).
- Configure the timer in gated mode by writing SMS=101 in TIMx\_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx\_SMCR register.
- Enable the counter by writing CEN=1 in the TIMx\_CR1 register (in gated mode, the counter doesn't start if CEN=0, whatever is the trigger input level).

For code example refer to the Appendix section [A.9.13: Gated mode code example](#).

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

The delay between the rising edge on TI1 and the actual stop of the counter is due to the resynchronization circuit on TI1 input.

**Figure 148. Control circuit in gated mode**



- The configuration "CCxP=CCxNP=1" (detection of both rising and falling edges) does not have any effect in gated mode because gated mode acts on a level and not on an edge.

### Slave mode: Trigger mode

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

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

- Configure the channel 2 to detect rising edges on TI2. Configure the input filter duration (in this example, we do not need any filter, so we keep IC2F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. CC2S bits are selecting the input capture source only, CC2S=01 in TIMx\_CCMR1 register. Write

CC2P=1 and CC2NP=0 in TIMx\_CCER register to validate the polarity (and detect low level only).

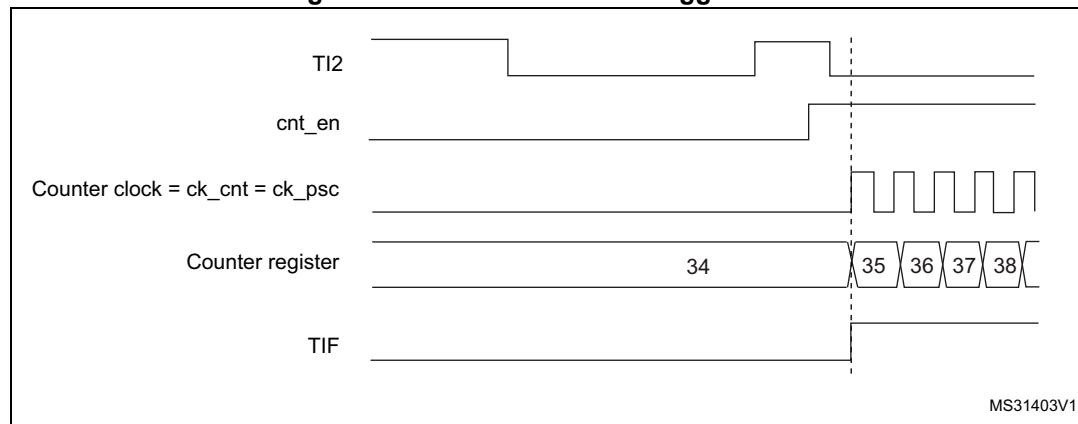
- Configure the timer in trigger mode by writing SMS=110 in TIMx\_SMCR register. Select TI2 as the input source by writing TS=110 in TIMx\_SMCR register.

For code example refer to the Appendix section [A.9.14: Trigger mode code example](#).

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

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

**Figure 149. Control circuit in trigger mode**



### Slave mode: External Clock mode 2 + trigger mode

The external clock mode 2 can be used in addition to another slave mode (except external clock mode 1 and encoder mode). In this case, the ETR signal is used as external clock input, and another input can be selected as trigger input when operating in reset mode, gated mode or trigger mode. It is recommended not to select ETR as TRGI through the TS bits of TIMx\_SMCR register.

In the following example, the upcounter is incremented at each rising edge of the ETR signal as soon as a rising edge of TI1 occurs:

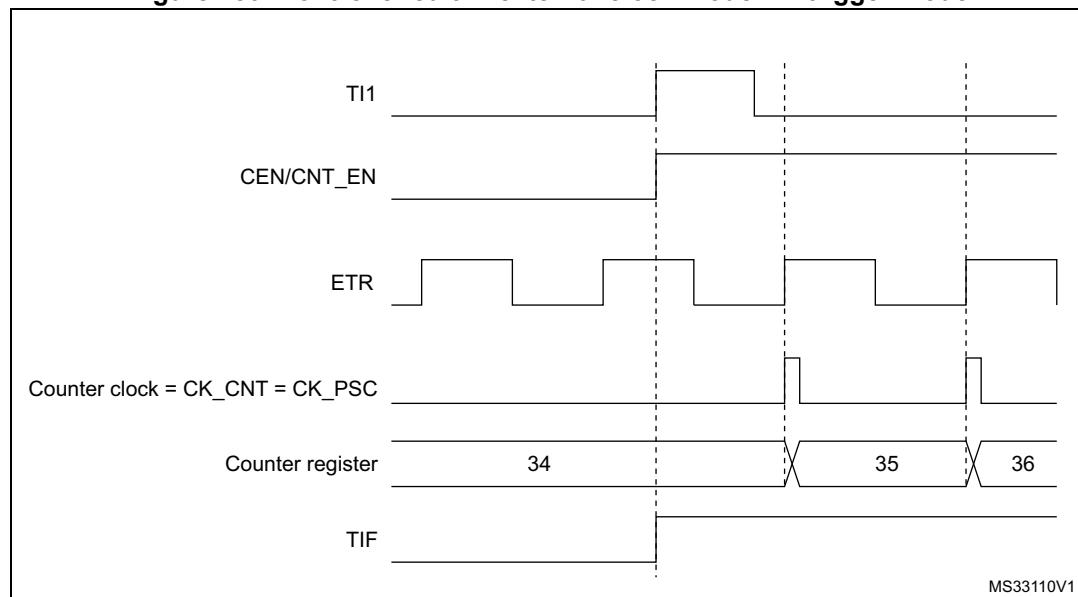
1. Configure the external trigger input circuit by programming the TIMx\_SMCR register as follows:
  - ETF = 0000: no filter
  - ETPS=00: prescaler disabled
  - ETP=0: detection of rising edges on ETR and ECE=1 to enable the external clock mode 2.
2. Configure the channel 1 as follows, to detect rising edges on TI:
  - IC1F=0000: no filter.
  - The capture prescaler is not used for triggering and does not need to be configured.
  - CC1S=01in TIMx\_CCMR1 register to select only the input capture source
  - CC1P=0 and CC1NP=0 in TIMx\_CCER register to validate the polarity (and detect rising edge only).
3. Configure the timer in trigger mode by writing SMS=110 in TIMx\_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx\_SMCR register.

For code example refer to the Appendix section [A.9.15: External clock mode 2 + trigger mode code example](#).

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

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

**Figure 150. Control circuit in external clock mode 2 + trigger mode**



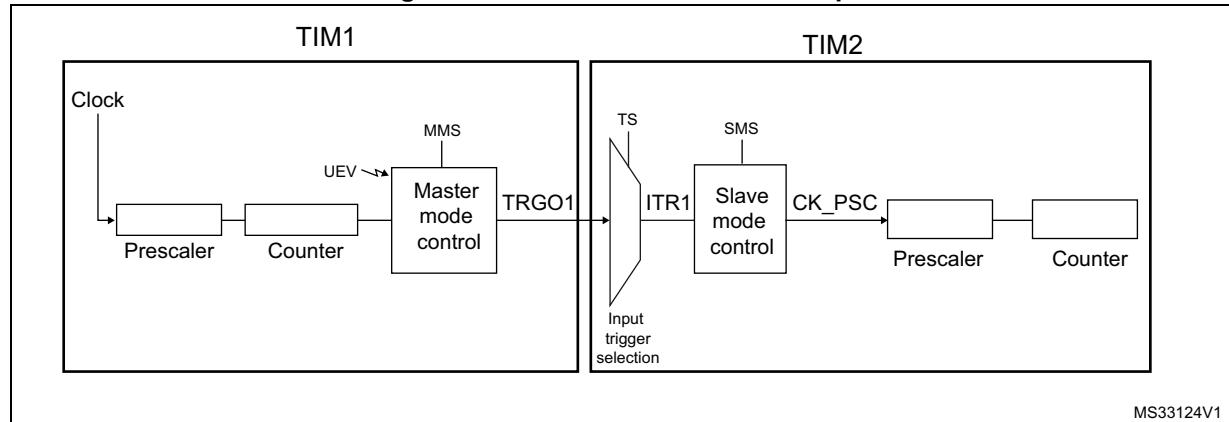
### 18.3.15 Timer synchronization

The TIMx timers are linked together internally for timer synchronization or chaining. When one Timer is configured in Master Mode, it can reset, start, stop or clock the counter of another Timer configured in Slave Mode.

[Figure 151: Master/Slave timer example](#) presents an overview of the trigger selection and the master mode selection blocks.

### Using one timer as prescaler for another

**Figure 151. Master/Slave timer example**



MS33124V1

For example, Timer 1 can be configured to act as a prescaler for Timer 2. Refer to [Figure 151](#). To do this:

- Configure Timer 1 in master mode so that it outputs a periodic trigger signal on each update event UEV. If MMS=010 is written in the TIM1\_CR2 register, a rising edge is output on TRGO1 each time an update event is generated.
- To connect the TRGO1 output of Timer 1 to Timer 2, Timer 2 must be configured in slave mode using ITR1 as internal trigger. This is selected through the TS bits in the TIM2\_SMCR register (writing TS=000).
- Then the Timer2's slave mode controller should be configured in external clock mode 1 (write SMS=111 in the TIM2\_SMCR register). This causes Timer 2 to be clocked by the rising edge of the periodic Timer 1 trigger signal (which correspond to the timer 1 counter overflow).
- Finally both timers must be enabled by setting their respective CEN bits within their respective TIMx\_CR1 registers. Make sure to enable Timer2 before enabling Timer1.

For code example refer to the Appendix section [A.9.17: Timer prescaling another timer code example](#).

**Note:** If OCx is selected on Timer 1 as trigger output (MMS=1xx), its rising edge is used to clock the counter of timer 2.

### Using one timer to enable another timer

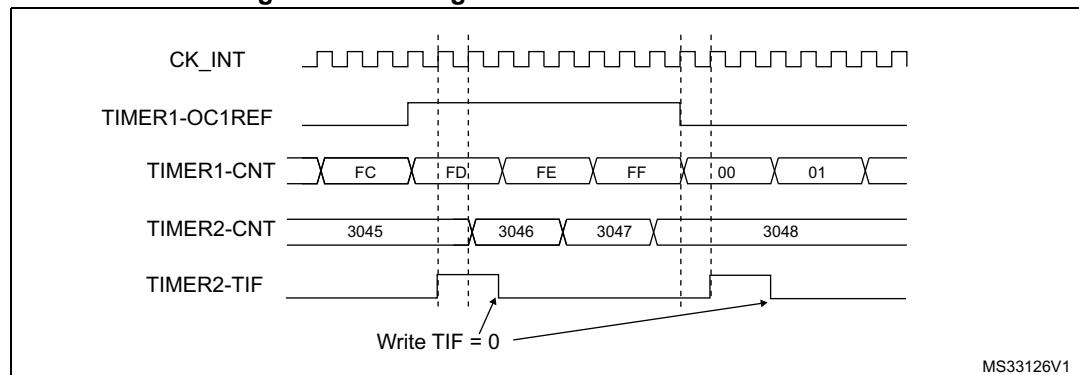
In this example, we control the enable of Timer 2 with the output compare 1 of Timer 1. Refer to [Figure 151](#) for connections. Timer 2 counts on the divided internal clock only when OC1REF of Timer 1 is high. Both counter clock frequencies are divided by 3 by the prescaler compared to CK\_INT ( $f_{CK\_CNT} = f_{CK\_INT}/3$ ).

- Configure Timer 1 master mode to enable the slave timer(MMS=001 in the TIM1\_CR2 register).
- Configure the Timer 1 OC1REF waveform (TIM1\_CCMR1 register).
- Configure Timer 2 to get the input trigger from Timer 1 (TS=000 in the TIM2\_SMCR register).
- Configure Timer 2 in gated mode (SMS=101 in TIM2\_SMCR register).
- Enable Timer 2 by writing '1 in the CEN bit (TIM2\_CR1 register).
- Start Timer 1 by writing '1 in the CEN bit (TIM1\_CR1 register).

For code example refer to the Appendix section [A.9.18: Timer enabling another timer code example](#).

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

**Figure 152. Gating timer 2 with OC1REF of timer 1**



MS33126V1

In the example in [Figure 152](#), the Timer 2 counter and prescaler are not initialized before being started. So they start counting from their current value. It is possible to start from a given value by resetting both timers before starting Timer 1. Then any value can be written in the timer counters. The timers can easily be reset by software using the UG bit in the TIMx\_EGR registers.

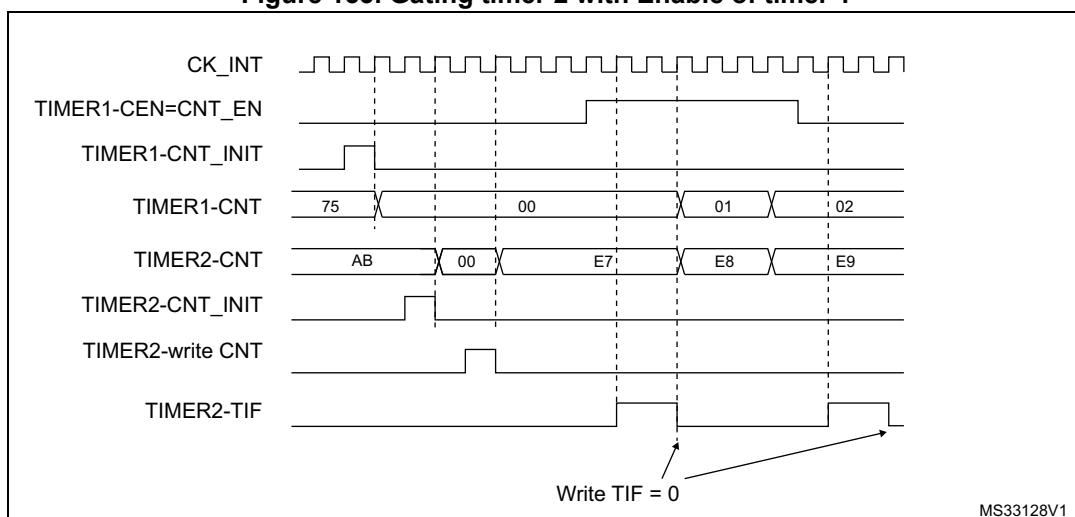
In the next example, we synchronize Timer 1 and Timer 2. Timer 1 is the master and starts from 0. Timer 2 is the slave and starts from 0xE7. The prescaler ratio is the same for both

timers. Timer 2 stops when Timer 1 is disabled by writing '0' to the CEN bit in the TIM1\_CR1 register:

- Configure Timer 1 master mode to send its Counter Enable signal (CNT\_EN) as a trigger output (MMS=001 in the TIM1\_CR2 register).
  - Configure the Timer 1 OC1REF waveform (TIM1\_CCMR1 register).
  - Configure Timer 2 to get the input trigger from Timer 1 (TS=000 in the TIM2\_SMCR register).
  - Configure Timer 2 in gated mode (SMS=101 in TIM2\_SMCR register).
  - Reset Timer 1 by writing ‘1 in UG bit (TIM1\_EGR register).
  - Reset Timer 2 by writing ‘1 in UG bit (TIM2\_EGR register).
  - Initialize Timer 2 to 0xE7 by writing ‘0xE7’ in the timer 2 counter (TIM2\_CNTL).
  - Enable Timer 2 by writing ‘1 in the CEN bit (TIM2\_CR1 register).
  - Start Timer 1 by writing ‘1 in the CEN bit (TIM1\_CR1 register).
  - Stop Timer 1 by writing ‘0 in the CEN bit (TIM1\_CR1 register).

For code example refer to the Appendix section [A.9.19: Master and slave synchronization code example](#).

**Figure 153.** Gating timer 2 with Enable of timer 1

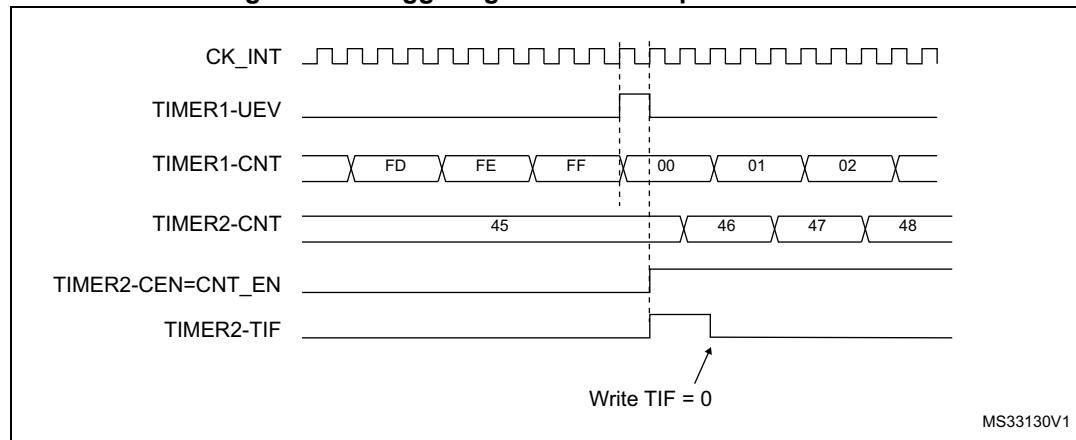


### Using one timer to start another timer

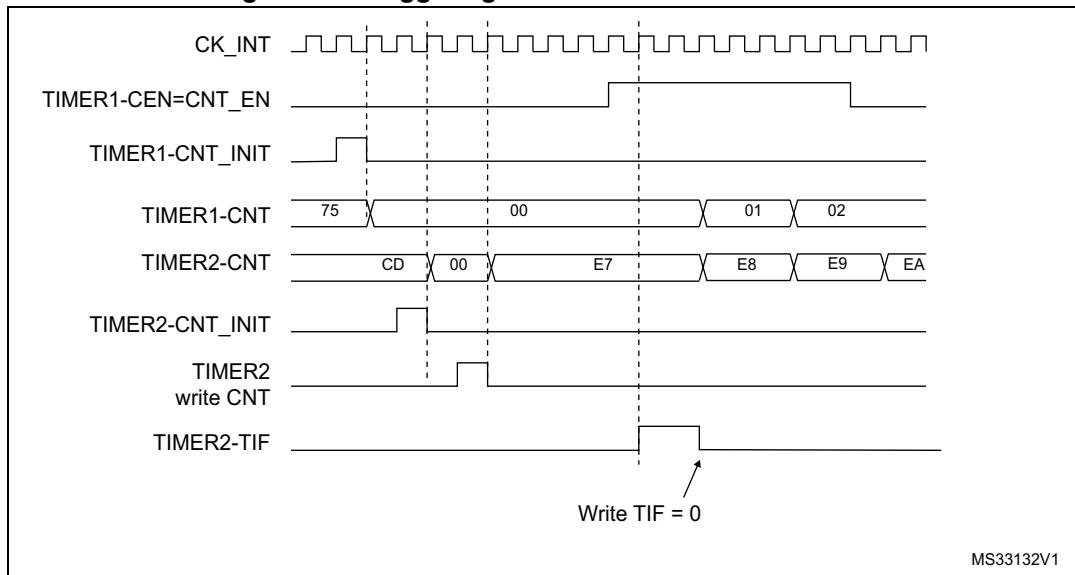
In this example, we set the enable of Timer 2 with the update event of Timer 1. Refer to [Figure 151](#) for connections. Timer 2 starts counting from its current value (which can be nonzero) on the divided internal clock as soon as the update event is generated by Timer 1. When Timer 2 receives the trigger signal its CEN bit is automatically set and the counter counts until we write '0' to the CEN bit in the TIM2\_CR1 register. Both counter clock frequencies are divided by 3 by the prescaler compared to CK\_INT ( $f_{CK\_CNT} = f_{CK\_INT}/3$ ).

- Configure Timer 1 master mode to send its Update Event (UEV) as trigger output (MMS=010 in the TIM1\_CR2 register).
- Configure the Timer 1 period (TIM1\_ARR registers).
- Configure Timer 2 to get the input trigger from Timer 1 (TS=000 in the TIM2\_SMCR register).
- Configure Timer 2 in trigger mode (SMS=110 in TIM2\_SMCR register).
- Start Timer 1 by writing '1' in the CEN bit (TIM1\_CR1 register).

**Figure 154. Triggering timer 2 with update of timer 1**



As in the previous example, both counters can be initialized before starting counting. [Figure 155](#) shows the behavior with the same configuration as in [Figure 154](#) but in trigger mode instead of gated mode (SMS=110 in the TIM2\_SMCR register).

**Figure 155. Triggering timer 2 with Enable of timer 1**

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

In this example, we set the enable of timer 1 when its TI1 input rises, and the enable of Timer 2 with the enable of Timer 1. Refer to [Figure 151](#) for connections. To ensure the counters are aligned, Timer 1 must be configured in Master/Slave mode (slave with respect to TI1, master with respect to Timer 2):

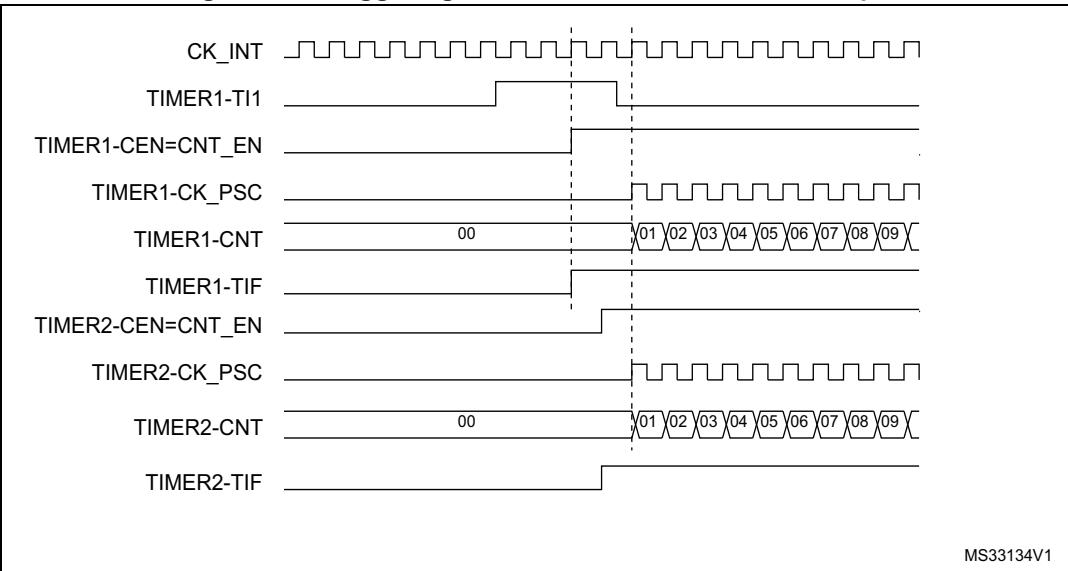
- Configure Timer 1 master mode to send its Enable as trigger output (MMS=001 in the **TIM1\_CR2** register).
- Configure Timer 1 slave mode to get the input trigger from TI1 (TS=100 in the **TIM1\_SMCR** register).
- Configure Timer 1 in trigger mode (SMS=110 in the **TIM1\_SMCR** register).
- Configure the Timer 1 in Master/Slave mode by writing MSM=1 (**TIM1\_SMCR** register).
- Configure Timer 2 to get the input trigger from Timer 1 (TS=000 in the **TIM2\_SMCR** register).
- Configure Timer 2 in trigger mode (SMS=110 in the **TIM2\_SMCR** register).

For code example refer to the Appendix section [A.9.20: Two timers synchronized by an external trigger code example](#).

When a rising edge occurs on TI1 (Timer 1), both counters starts counting synchronously on the internal clock and both TIF flags are set.

*Note:*

*In this example both timers are initialized before starting (by setting their respective UG bits). Both counters starts from 0, but an offset can easily be inserted between them by writing any of the counter registers (TIMx\_CNT). One can see that the master/slave mode insert a delay between CNT\_EN and CK\_PSC on timer 1.*

**Figure 156. Triggering timer 1 and 2 with timer 1 TI1 input**

### 18.3.16 Debug mode

When the microcontroller enters debug mode (Arm® Cortex®-M0 core - halted), the TIMx counter either continues to work normally or stops, depending on DBG\_TIMx\_STOP configuration bit in DBGMCU module.

## 18.4 TIM2 and TIM3 registers

Refer to [Section 1.2 on page 42](#) for a list of abbreviations used in register descriptions.

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

### 18.4.1 TIM2 and TIM3 control register 1 (TIM2\_CR1 and TIM3\_CR1)

Address offset: 0x000

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	CKD[1:0]	ARPE	CMS[1:0]	DIR	OPM	URS	UDIS	CEN		

Bits 15:10 Reserved, must be kept at reset value.

Bits 9:8 **CKD**: Clock division

This bit-field indicates the division ratio between the timer clock (CK\_INT) frequency and sampling clock used by the digital filters (ETR, TIx),

- 00:  $t_{DTS} = t_{CK\_INT}$
- 01:  $t_{DTS} = 2 \times t_{CK\_INT}$
- 10:  $t_{DTS} = 4 \times t_{CK\_INT}$
- 11: Reserved

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

- 0: TIMx\_ARR register is not buffered
- 1: TIMx\_ARR register is buffered

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

00: Edge-aligned mode. The counter counts up or down depending on the direction bit (DIR).

01: Center-aligned mode 1. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx\_CCMRx register) are set only when the counter is counting down.

10: Center-aligned mode 2. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx\_CCMRx register) are set only when the counter is counting up.

11: Center-aligned mode 3. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx\_CCMRx register) are set both when the counter is counting up or down.

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

Bit 4 **DIR**: Direction

- 0: Counter used as upcounter
- 1: Counter used as downcounter

*Note:* This bit is read only when the timer is configured in Center-aligned mode or Encoder mode.

Bit 3 **OPM**: One-pulse mode

- 0: Counter is not stopped at update event
- 1: Counter stops counting at the next update event (clearing the bit CEN)

**Bit 2 URS:** Update request source

This bit is set and cleared by software to select the UEV event sources.

0: Any of the following events generate an update interrupt or DMA request if enabled.

These events can be:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

1: Only counter overflow/underflow generates an update interrupt or DMA request if enabled.

**Bit 1 UDIS:** Update disable

This bit is set and cleared by software to enable/disable UEV event generation.

0: UEV enabled. The Update (UEV) event is generated by one of the following events:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

Buffered registers are then loaded with their preload values.

1: UEV disabled. The Update event is not generated, shadow registers keep their value (ARR, PSC, CCRx). However the counter and the prescaler are reinitialized if the UG bit is set or if a hardware reset is received from the slave mode controller.

**Bit 0 CEN:** Counter enable

0: Counter disabled

1: Counter enabled

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

CEN is cleared automatically in one-pulse mode, when an update event occurs.

### 18.4.2 TIM2 and TIM3 control register 2 (TIM2\_CR2 and TIM3\_CR2)

Address offset: 0x04

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TI1S	MMS[2:0]	CCDS	Res.	Res.	Res.									
								rw	rw	rw	rw	rw			

Bits 15:8 Reserved, must be kept at reset value.

Bit 7 **TI1S**: TI1 selection

- 0: The TIMx\_CH1 pin is connected to TI1 input
  - 1: The TIMx\_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)
- See also [Section 17.3.18: Interfacing with Hall sensors on page 369](#)

Bits 6:4 **MMS**: Master mode selection

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

000: **Reset** - the UG bit from the TIMx\_EGR register is used as trigger output (TRGO). If the reset is generated by the trigger input (slave mode controller configured in reset mode) then the signal on TRGO is delayed compared to the actual reset.

001: **Enable** - the Counter enable signal, CNT\_EN, is used as trigger output (TRGO). It is useful to start several timers at the same time or to control a window in which a slave timer is enabled. The Counter Enable signal is generated by a logic OR between CEN control bit and the trigger input when configured in gated mode.

When the Counter Enable signal is controlled by the trigger input, there is a delay on TRGO, except if the master/slave mode is selected (see the MSM bit description in TIMx\_SMCR register).

010: **Update** - The update event is selected as trigger output (TRGO). For instance a master timer can then be used as a prescaler for a slave timer.

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

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

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

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

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

Bit 3 **CCDS**: Capture/compare DMA selection

- 0: CCx DMA request sent when CCx event occurs
- 1: CCx DMA requests sent when update event occurs

Bits 2:0 Reserved, must be kept at reset value.

### 18.4.3 TIM2 and TIM3 slave mode control register (TIM2\_SMCR and TIM3\_SMCR)

Address offset: 0x08

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ETP	ECE	ETPS[1:0]		ETF[3:0]			MSM	TS[2:0]			OCCS	SMS[2:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 **ETP**: External trigger polarity

This bit selects whether ETR or ETR is used for trigger operations

- 0: ETR is noninverted, active at high level or rising edge
- 1: ETR is inverted, active at low level or falling edge

Bit 14 **ECE**: External clock enable

This bit enables External clock mode 2.

- 0: External clock mode 2 disabled
- 1: External clock mode 2 enabled. The counter is clocked by any active edge on the ETRF signal.
- 1: Setting the ECE bit has the same effect as selecting external clock mode 1 with TRGI connected to ETRF (SMS=111 and TS=111).**
- 2: It is possible to simultaneously use external clock mode 2 with the following slave modes: reset mode, gated mode and trigger mode. Nevertheless, TRGI must not be connected to ETRF in this case (TS bits must not be 111).**
- 3: If external clock mode 1 and external clock mode 2 are enabled at the same time, the external clock input is ETRF.**

Bits 13:12 **ETPS[1:0]**: External trigger prescaler

External trigger signal ETRP frequency must be at most 1/4 of CK\_INT frequency. A prescaler can be enabled to reduce ETRP frequency. It is useful when inputting fast external clocks.

- 00: Prescaler OFF
- 01: ETRP frequency divided by 2
- 10: ETRP frequency divided by 4
- 11: ETRP frequency divided by 8

Bits 11:8 **ETF[3:0]**: External trigger filter

This bit-field defines the frequency used to sample ETRP signal and the length of the digital filter applied to ETRP. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output:

- 0000: No filter, sampling is done at  $f_{DTS}$
- 0001:  $f_{SAMPLING} = f_{CK\_INT}$ ,  $N = 2$
- 0010:  $f_{SAMPLING} = f_{CK\_INT}$ ,  $N = 4$
- 0011:  $f_{SAMPLING} = f_{CK\_INT}$ ,  $N = 8$
- 0100:  $f_{SAMPLING} = f_{DTS} / 2$ ,  $N = 6$
- 0101:  $f_{SAMPLING} = f_{DTS} / 2$ ,  $N = 8$
- 0110:  $f_{SAMPLING} = f_{DTS} / 4$ ,  $N = 6$
- 0111:  $f_{SAMPLING} = f_{DTS} / 4$ ,  $N = 8$
- 1000:  $f_{SAMPLING} = f_{DTS} / 8$ ,  $N = 6$
- 1001:  $f_{SAMPLING} = f_{DTS} / 8$ ,  $N = 8$
- 1010:  $f_{SAMPLING} = f_{DTS} / 16$ ,  $N = 5$
- 1011:  $f_{SAMPLING} = f_{DTS} / 16$ ,  $N = 6$
- 1100:  $f_{SAMPLING} = f_{DTS} / 16$ ,  $N = 8$
- 1101:  $f_{SAMPLING} = f_{DTS} / 32$ ,  $N = 5$
- 1110:  $f_{SAMPLING} = f_{DTS} / 32$ ,  $N = 6$
- 1111:  $f_{SAMPLING} = f_{DTS} / 32$ ,  $N = 8$

*Note: Care must be taken that  $f_{DTS}$  is replaced in the formula by CK\_INT when ETF[3:0] = 1, 2 or 3.*

Bit 7 **MSM**: Master/Slave mode

- 0: No action
- 1: The effect of an event on the trigger input (TRGI) is delayed to allow a perfect synchronization between the current timer and its slaves (through TRGO). It is useful if we want to synchronize several timers on a single external event.

Bits 6:4 **TS**: Trigger selection

This bit-field selects the trigger input to be used to synchronize the counter.

- 000: Internal Trigger 0 (ITR0).
- 001: Internal Trigger 1 (ITR1).
- 010: Internal Trigger 2 (ITR2).
- 011: Internal Trigger 3 (ITR3).
- 100: TI1 Edge Detector (TI1F\_ED)
- 101: Filtered Timer Input 1 (TI1FP1)
- 110: Filtered Timer Input 2 (TI2FP2)
- 111: External Trigger input (ETRF)

See [Table 66: TIM2 and TIM3 internal trigger connection on page 451](#) for more details on ITRx meaning for each Timer.

*Note: These bits must be changed only when they are not used (e.g. when SMS=000) to avoid wrong edge detections at the transition.*

Bit 3 **OCCS**: OCREF clear selection.

This bit is used to select the OCREF clear source.

- 0: OCREF\_CLR\_INT is connected to the OCREF\_CLR input
- 1: OCREF\_CLR\_INT is connected to ETRF

Bits 2:0 **SMS:** Slave mode selection

When external signals are selected the active edge of the trigger signal (TRGI) is linked to the polarity selected on the external input (see Input Control register and Control Register description).

000: Slave mode disabled - if CEN = '1 then the prescaler is clocked directly by the internal clock.

001: Encoder mode 1 - Counter counts up/down on TI1FP1 edge depending on TI2FP2 level.

010: Encoder mode 2 - Counter counts up/down on TI2FP2 edge depending on TI1FP1 level.

011: Encoder mode 3 - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input.

100: Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter and generates an update of the registers.

101: Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high. The counter stops (but is not reset) as soon as the trigger becomes low. Both start and stop of the counter are controlled.

110: Trigger Mode - The counter starts at a rising edge of the trigger TRGI (but it is not reset). Only the start of the counter is controlled.

111: External Clock Mode 1 - Rising edges of the selected trigger (TRGI) clock the counter.

*Note: The gated mode must not be used if TI1F\_ED is selected as the trigger input (TS=100). Indeed, TI1F\_ED outputs 1 pulse for each transition on TI1F, whereas the gated mode checks the level of the trigger signal.*

*Note: The clock of the slave timer must be enabled prior to receive events from the master timer, and must not be changed on-the-fly while triggers are received from the master timer.*

Table 66. TIM2 and TIM3 internal trigger connection

Slave TIM	ITR0 (TS = 000)	ITR1 (TS = 001)	ITR2 (TS = 010)	ITR3 (TS = 011)
TIM2	TIM1	TIM15	TIM3	TIM14
TIM3	TIM1	TIM2	TIM15	TIM14

#### 18.4.4 TIM2 and TIM3 DMA/Interrupt enable register (TIM2\_DIER and TIM3\_DIER)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	TDE	Res	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Res	TIE	Res	CC4IE	CC3IE	CC2IE	CC1IE	UIE

Bit 15 Reserved, must be kept at reset value.

Bit 14 **TDE:** Trigger DMA request enable

0: Trigger DMA request disabled.

1: Trigger DMA request enabled.

Bit 13 Reserved, must be kept at reset value.

- Bit 12 **CC4DE**: Capture/Compare 4 DMA request enable  
0: CC4 DMA request disabled.  
1: CC4 DMA request enabled.
- Bit 11 **CC3DE**: Capture/Compare 3 DMA request enable  
0: CC3 DMA request disabled.  
1: CC3 DMA request enabled.
- Bit 10 **CC2DE**: Capture/Compare 2 DMA request enable  
0: CC2 DMA request disabled.  
1: CC2 DMA request enabled.
- Bit 9 **CC1DE**: Capture/Compare 1 DMA request enable  
0: CC1 DMA request disabled.  
1: CC1 DMA request enabled.
- Bit 8 **UDE**: Update DMA request enable  
0: Update DMA request disabled.  
1: Update DMA request enabled.
- Bit 7 Reserved, must be kept at reset value.
- Bit 6 **TIE**: Trigger interrupt enable  
0: Trigger interrupt disabled.  
1: Trigger interrupt enabled.
- Bit 5 Reserved, must be kept at reset value.
- 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

#### 18.4.5 TIM2 and TIM3 status register (TIM2\_SR and TIM3\_SR)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	CC4OF	CC3OF	CC2OF	CC1OF	Res.	Res.	TIF	Res.	CC4IF	CC3IF	CC2IF	CC1IF	UIF
			rc_w0	rc_w0	rc_w0	rc_w0			rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

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

Bit 12 **CC4OF**: Capture/Compare 4 overcapture flag

Refer to CC1OF description

Bit 11 **CC3OF**: Capture/Compare 3 overcapture flag

Refer to CC1OF description

Bit 10 **CC2OF**: Capture/compare 2 overcapture flag

Refer to CC1OF description

Bit 9 **CC1OF**: Capture/Compare 1 overcapture flag

This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to '0'.

0: No overcapture has been detected

1: The counter value has been captured in TIMx\_CCR1 register while CC1IF flag was already set

Bits 8:7 Reserved, must be kept at reset value.

Bit 6 **TIF**: Trigger interrupt flag

This flag is set by hardware on trigger event (active edge detected on TRGI input when the slave mode controller is enabled in all modes but gated mode. It is set when the counter starts or stops when gated mode is selected. It is cleared by software.

0: No trigger event occurred

1: Trigger interrupt pending

Bit 5 Reserved, must be kept at reset value.

Bit 4 **CC4IF**: Capture/Compare 4 interrupt flag

Refer to CC1IF description

Bit 3 **CC3IF**: Capture/Compare 3 interrupt flag

Refer to CC1IF description

Bit 2 **CC2IF**: Capture/Compare 2 interrupt flag

Refer to CC1IF description

Bit 1 **CC1IF**: Capture/compare 1 interrupt flag

**If channel CC1 is configured as output:**

This flag is set by hardware when the counter matches the compare value, with some exception in center-aligned mode (refer to the CMS bits in the TIMx\_CR1 register description). It is cleared by software.

0: No match

1: The content of the counter TIMx\_CNT matches the content of the TIMx\_CCR1 register.

When the contents of TIMx\_CCR1 are greater than the contents of TIMx\_ARR, the CC1IF bit goes high on the counter overflow (in upcounting and up/down-counting modes) or underflow (in downcounting mode)

**If channel CC1 is configured as input:**

This bit is set by hardware on a capture. It is cleared by software or by reading the TIMx\_CCR1 register.

0: No input capture occurred

1: The counter value has been captured in TIMx\_CCR1 register (An edge has been detected on IC1 which matches the selected polarity)

Bit 0 **UIF**: Update interrupt flag

This bit is set by hardware on an update event. It is cleared by software.

0: No update occurred.

1: Update interrupt pending.

This bit is set by hardware when the registers are updated:

At overflow or underflow and if UDIS=0 in the TIMx\_CR1 register.

When CNT is reinitialized by software using the UG bit in TIMx\_EGR register, if URS=0 and UDIS=0 in the TIMx\_CR1 register.

When CNT is reinitialized by a trigger event (refer to the synchro control register description), if URS=0 and UDIS=0 in the TIMx\_CR1 register.

### 18.4.6 TIM2 and TIM3 event generation register (TIM2\_EGR and TIM3\_EGR)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TG	Res.	CC4G	CC3G	CC2G	CC1G	UG								

Bits 15:7 Reserved, must be kept at reset value.

Bit 6 **TG**: Trigger generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: The TIF flag is set in TIMx\_SR register. Related interrupt or DMA transfer can occur if enabled.

Bit 5 Reserved, must be kept at reset value.

Bit 4 **CC4G**: Capture/compare 4 generation

Refer to CC1G description

Bit 3 **CC3G**: Capture/compare 3 generation

Refer to CC1G description

Bit 2 **CC2G**: Capture/compare 2 generation

Refer to CC1G description

Bit 1 **CC1G**: Capture/compare 1 generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: A capture/compare event is generated on channel 1:

**If channel CC1 is configured as output:**

CC1IF flag is set, Corresponding interrupt or DMA request is sent if enabled.

**If channel CC1 is configured as input:**

The current value of the counter is captured in TIMx\_CCR1 register. The CC1IF flag is set, the corresponding interrupt or DMA request is sent if enabled. The CC1OF flag is set if the CC1IF flag was already high.

Bit 0 **UG**: Update generation

This bit can be set by software, it is automatically cleared by hardware.

0: No action

1: Re-initialize the counter and generates an update of the registers. Note that the prescaler counter is cleared too (anyway the prescaler ratio is not affected). The counter is cleared if the center-aligned mode is selected or if DIR=0 (upcounting), else it takes the auto-reload value (TIMx\_ARR) if DIR=1 (downcounting).

### 18.4.7 TIM2 and TIM3 capture/compare mode register 1 (TIM2\_CCMR1 and TIM3\_CCMR1)

Address offset: 0x18

Reset value: 0x0000

The channels can be used in input (capture mode) or in output (compare mode). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OCxx describes its function when the channel is configured in output, ICxx describes its function when the channel is configured in input. So one must take care that the same bit can have a different meaning for the input stage and for the output stage.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]		
IC2F[3:0]			IC2PSC[1:0]				IC1F[3:0]			IC1PSC[1:0]					
rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw

#### Output compare mode

Bit 15 **OC2CE**: Output compare 2 clear enable

Bits 14:12 **OC2M[2:0]**: Output compare 2 mode

Bit 11 **OC2PE**: Output compare 2 preload enable

Bit 10 **OC2FE**: Output compare 2 fast enable

Bits 9:8 **CC2S[1:0]**: Capture/Compare 2 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output

01: CC2 channel is configured as input, IC2 is mapped on TI2

10: CC2 channel is configured as input, IC2 is mapped on TI1

11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode is working only if an internal trigger input is selected through the TS bit (TIMx\_SMCR register)

*Note: CC2S bits are writable only when the channel is OFF (CC2E = 0 in TIMx\_CCER).*

Bit 7 **OC1CE**: Output compare 1 clear enable

OC1CE: Output Compare 1 Clear Enable

0: OC1Ref is not affected by the ETRF input

1: OC1Ref is cleared as soon as a High level is detected on ETRF input

**Bits 6:4 OC1M:** Output compare 1 mode

These bits define the behavior of the output reference signal OC1REF from which OC1 and OC1N are derived. OC1REF is active high whereas OC1 and OC1N active level depends on CC1P and CC1NP bits.

000: Frozen - The comparison between the output compare register TIMx\_CCR1 and the counter TIMx\_CNT has no effect on the outputs.(this mode is used to generate a timing base).

001: Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

011: Toggle - OC1REF toggles when TIMx\_CNT=TIMx\_CCR1.

100: Force inactive level - OC1REF is forced low.

101: Force active level - OC1REF is forced high.

110: PWM mode 1 - In upcounting, channel 1 is active as long as TIMx\_CNT<TIMx\_CCR1 else inactive. In downcounting, channel 1 is inactive (OC1REF='0) as long as TIMx\_CNT>TIMx\_CCR1 else active (OC1REF=1).

111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIMx\_CNT<TIMx\_CCR1 else active. In downcounting, channel 1 is active as long as TIMx\_CNT>TIMx\_CCR1 else inactive.

*Note: 1: These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx\_BDTR register) and CC1S=00 (the channel is configured in output).*

*2: In PWM mode 1 or 2, the OCREF level changes only when the result of the comparison changes or when the output compare mode switches from “frozen” mode to “PWM” mode.*

**Bit 3 OC1PE:** Output compare 1 preload enable

0: Preload register on TIMx\_CCR1 disabled. TIMx\_CCR1 can be written at anytime, the new value is taken in account immediately.

1: Preload register on TIMx\_CCR1 enabled. Read/Write operations access the preload register. TIMx\_CCR1 preload value is loaded in the active register at each update event.

*Note: These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx\_BDTR register) and CC1S=00 (the channel is configured in output).*

**Bit 2 OC1FE:** Output compare 1 fast enable

This bit is used to accelerate the effect of an event on the trigger in input on the CC output.

0: CC1 behaves normally depending on counter and CCR1 values even when the trigger is ON. The minimum delay to activate CC1 output when an edge occurs on the trigger input is 5 clock cycles.

1: An active edge on the trigger input acts like a compare match on CC1 output. Then, OC is set to the compare level independently from the result of the comparison. Delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. OC1FE acts only if the channel is configured in PWM1 or PWM2 mode.

**Bits 1:0 CC1S:** Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output.

01: CC1 channel is configured as input, IC1 is mapped on TI1.

10: CC1 channel is configured as input, IC1 is mapped on TI2.

11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC1S bits are writable only when the channel is OFF (CC1E = 0 in TIMx\_CCER).*

**Input capture mode**

Bits 15:12 **IC2F**: Input capture 2 filter

Bits 11:10 **IC2PSC[1:0]**: Input capture 2 prescaler

Bits 9:8 **CC2S**: Capture/compare 2 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output.

01: CC2 channel is configured as input, IC2 is mapped on TI2.

10: CC2 channel is configured as input, IC2 is mapped on TI1.

11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC2S bits are writable only when the channel is OFF (CC2E = 0 in TIMx\_CCER).*

Bits 7:4 **IC1F**: Input capture 1 filter

This bit-field defines the frequency used to sample TI1 input and the length of the digital filter applied to TI1. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output:

0000: No filter, sampling is done at  $f_{DTS}$

0001:  $f_{SAMPLING} = f_{CK\_INT}$ , N = 2

0010:  $f_{SAMPLING} = f_{CK\_INT}$ , N = 4

0011:  $f_{SAMPLING} = f_{CK\_INT}$ , N = 8

0100:  $f_{SAMPLING} = f_{DTS} / 2$ , N = 6

0101:  $f_{SAMPLING} = f_{DTS} / 2$ , N = 8

0110:  $f_{SAMPLING} = f_{DTS} / 4$ , N = 6

0111:  $f_{SAMPLING} = f_{DTS} / 4$ , N = 8

1000:  $f_{SAMPLING} = f_{DTS} / 8$ , N = 6

1001:  $f_{SAMPLING} = f_{DTS} / 8$ , N = 8

1010:  $f_{SAMPLING} = f_{DTS} / 16$ , N = 5

1011:  $f_{SAMPLING} = f_{DTS} / 16$ , N = 6

1100:  $f_{SAMPLING} = f_{DTS} / 16$ , N = 8

1101:  $f_{SAMPLING} = f_{DTS} / 32$ , N = 5

1110:  $f_{SAMPLING} = f_{DTS} / 32$ , N = 6

1111:  $f_{SAMPLING} = f_{DTS} / 32$ , N = 8

*Note: Care must be taken that  $f_{DTS}$  is replaced in the formula by CK\_INT when ICxF[3:0] = 1, 2 or 3.*

Bits 3:2 **IC1PSC**: Input capture 1 prescaler

This bit-field defines the ratio of the prescaler acting on CC1 input (IC1).

The prescaler is reset as soon as CC1E=0 (TIMx\_CCER register).

00: no prescaler, capture is done each time an edge is detected on the capture input

01: capture is done once every 2 events

10: capture is done once every 4 events

11: capture is done once every 8 events

Bits 1:0 **CC1S**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output

01: CC1 channel is configured as input, IC1 is mapped on TI1

10: CC1 channel is configured as input, IC1 is mapped on TI2

11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC1S bits are writable only when the channel is OFF (CC1E = 0 in TIMx\_CCER).*

### 18.4.8 TIM2 and TIM3 capture/compare mode register 2 (TIM2\_CCMR2 and TIM3\_CCMR2)

Address offset: 0x1C

Reset value: 0x0000

Refer to the above CCMR1 register description.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
OC4CE	OC4M[2:0]			OC4PE	OC4FE	CC4S[1:0]	OC3CE	OC3M[2:0]			OC3PE	OC3FE	CC3S[1:0]			
IC4F[3:0]			IC4PSC[1:0]				IC3F[3:0]			IC3PSC[1:0]						
rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	

#### Output compare mode

Bit 15 **OC4CE**: Output compare 4 clear enable

Bits 14:12 **OC4M**: Output compare 4 mode

Bit 11 **OC4PE**: Output compare 4 preload enable

Bit 10 **OC4FE**: Output compare 4 fast enable

Bits 9:8 **CC4S**: Capture/Compare 4 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC4 channel is configured as output

01: CC4 channel is configured as input, IC4 is mapped on TI4

10: CC4 channel is configured as input, IC4 is mapped on TI3

11: CC4 channel is configured as input, IC4 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC4S bits are writable only when the channel is OFF (CC4E = 0 in TIMx\_CCER).*

Bit 7 **OC3CE**: Output compare 3 clear enable

Bits 6:4 **OC3M**: Output compare 3 mode

Bit 3 **OC3PE**: Output compare 3 preload enable

Bit 2 **OC3FE**: Output compare 3 fast enable

Bits 1:0 **CC3S**: Capture/Compare 3 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC3 channel is configured as output

01: CC3 channel is configured as input, IC3 is mapped on TI3

10: CC3 channel is configured as input, IC3 is mapped on TI4

11: CC3 channel is configured as input, IC3 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC3S bits are writable only when the channel is OFF (CC3E = 0 in TIMx\_CCER).*

## Input capture mode

Bits 15:12 **IC4F**: Input capture 4 filter

Bits 11:10 **IC4PSC**: Input capture 4 prescaler

Bits 9:8 **CC4S**: Capture/Compare 4 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC4 channel is configured as output

01: CC4 channel is configured as input, IC4 is mapped on TI4

10: CC4 channel is configured as input, IC4 is mapped on TI3

11: CC4 channel is configured as input, IC4 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC4S bits are writable only when the channel is OFF (CC4E = 0 in TIMx\_CCER).*

Bits 7:4 **IC3F**: Input capture 3 filter

Bits 3:2 **IC3PSC**: Input capture 3 prescaler

Bits 1:0 **CC3S**: Capture/Compare 3 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC3 channel is configured as output

01: CC3 channel is configured as input, IC3 is mapped on TI3

10: CC3 channel is configured as input, IC3 is mapped on TI4

11: CC3 channel is configured as input, IC3 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC3S bits are writable only when the channel is OFF (CC3E = 0 in TIMx\_CCER).*

## 18.4.9 TIM2 and TIM3 capture/compare enable register (TIM2\_CCER and TIM3\_CCER)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC4NP	Res.	CC4P	CC4E	CC3NP	Res.	CC3P	CC3E	CC2NP	Res.	CC2P	CC2E	CC1NP	Res.	CC1P	CC1E
rw		rw	rw												

Bit 15 **CC4NP**: Capture/Compare 4 output Polarity.

Refer to CC1NP description

Bit 14 Reserved, must be kept at reset value.

Bit 13 **CC4P**: Capture/Compare 4 output Polarity.

Refer to CC1P description

Bit 12 **CC4E**: Capture/Compare 4 output enable.

Refer to CC1E description

Bit 11 **CC3NP**: Capture/Compare 3 output Polarity.

Refer to CC1NP description

Bit 10 Reserved, must be kept at reset value.

Bit 9 **CC3P**: Capture/Compare 3 output Polarity.

Refer to CC1P description

Bit 8 **CC3E**: *Capture/Compare 3 output enable.*

Refer to CC1E description

Bit 7 **CC2NP**: *Capture/Compare 2 output Polarity.*

Refer to CC1NP description

Bit 6 Reserved, must be kept at reset value.

Bit 5 **CC2P**: *Capture/Compare 2 output Polarity.*

Refer to CC1P description

Bit 4 **CC2E**: *Capture/Compare 2 output enable.*

Refer to CC1E description

Bit 3 **CC1NP**: *Capture/Compare 1 output Polarity.*

**CC1 channel configured as output:**

CC1NP must be kept cleared in this case.

**CC1 channel configured as input:**

This bit is used in conjunction with CC1P to define TI1FP1/TI2FP1 polarity. refer to CC1P description.

Bit 2 Reserved, must be kept at reset value.

Bit 1 **CC1P**: *Capture/Compare 1 output Polarity.*

**CC1 channel configured as output:**

- 0: OC1 active high
- 1: OC1 active low

**CC1 channel configured as input:**

CC1NP/CC1P bits select TI1FP1 and TI2FP1 polarity for trigger or capture operations.

00: noninverted/rising edge

Circuit is sensitive to TIxFP1 rising edge (capture, trigger in reset, external clock or trigger mode), TIxFP1 is not inverted (trigger in gated mode, encoder mode).

01: inverted/falling edge

Circuit is sensitive to TIxFP1 falling edge (capture, trigger in reset, external clock or trigger mode), TIxFP1 is inverted (trigger in gated mode, encoder mode).

10: reserved, do not use this configuration.

11: noninverted/both edges

Circuit is sensitive to both TIxFP1 rising and falling edges (capture, trigger in reset, external clock or trigger mode), TIxFP1 is not inverted (trigger in gated mode). This configuration must not be used for encoder mode.

Bit 0 **CC1E**: *Capture/Compare 1 output enable.*

**CC1 channel configured as output:**

- 0: Off - OC1 is not active
- 1: On - OC1 signal is output on the corresponding output pin

**CC1 channel configured as input:**

This bit determines if a capture of the counter value can actually be done into the input capture/compare register 1 (TIMx\_CCR1) or not.

0: Capture disabled

1: Capture enabled

**Table 67. Output control bit for standard OCx channels**

CCxE bit	OCx output state
0	Output Disabled (OCx=0, OCx_EN=0)
1	OCx=OCxREF + Polarity, OCx_EN=1

**Note:** *The state of the external IO pins connected to the standard OCx channels depends on the OCx channel state and the GPIO registers.*

#### 18.4.10 TIM2 and TIM3 counter (TIM2\_CNT and TIM3\_CNT)

Address offset: 0x24

Reset value: 0x0000 0000

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

Bits 31:16 **CNT[31:16]**: High counter value (on TIM2 only).

Bits 15:0 **CNT[15:0]**: Low counter value

#### 18.4.11 TIM2 and TIM3 prescaler (TIM2\_PSC and TIM3\_PSC)

Address offset: 0x28

Reset value: 0x0000

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

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK\_CNT is equal to  $f_{CK\_PSC} / (PSC[15:0] + 1)$ .

PSC contains the value to be loaded in the active prescaler register at each update event.

### 18.4.12 TIM2 and TIM3 auto-reload register (TIM2\_ARR and TIM3\_ARR)

Address offset: 0x2C

Reset value: 0xFFFF FFFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ARR[31:16] (TIM2 only)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **ARR[31:16]**: High auto-reload value (on TIM2).

Bits 15:0 **ARR[15:0]**: Low Auto-reload value

ARR is the value to be loaded in the actual auto-reload register.

Refer to [Section 18.3.1: Time-base unit on page 405](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

### 18.4.13 TIM2 and TIM3 capture/compare register 1 (TIM2\_CCR1 and TIM3\_CCR1)

Address offset: 0x34

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR1[31:16] (TIM2 only)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **CCR1[31:16]**: High Capture/Compare 1 value (on TIM2).

Bits 15:0 **CCR1[15:0]**: Low Capture/Compare 1 value

**If channel CC1 is configured as output:**

CCR1 is the value to be loaded in the actual capture/compare 1 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR1 register (bit OC1PE). Otherwise the preload value is copied in the active capture/compare 1 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signaled on OC1 output.

**If channel CC1 is configured as input:**

CCR1 is the counter value transferred by the last input capture 1 event (IC1).

#### 18.4.14 TIM2 and TIM3 capture/compare register 2 (TIM2\_CCR2 and TIM3\_CCR2)

Address offset: 0x38

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR2[31:16] (TIM2 only)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR2[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **CCR2[31:16]**: High Capture/Compare 2 value (on TIM2).

Bits 15:0 **CCR2[15:0]**: Low Capture/Compare 2 value

**If channel CC2 is configured as output:**

CCR2 is the value to be loaded in the actual capture/compare 2 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR2 register (bit OC2PE). Else the preload value is copied in the active capture/compare 2 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signaled on OC2 output.

**If channel CC2 is configured as input:**

CCR2 is the counter value transferred by the last input capture 2 event (IC2).

#### 18.4.15 TIM2 and TIM3 capture/compare register 3 (TIM2\_CCR3 and TIM3\_CCR3)

Address offset: 0x3C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR3[31:16] (TIM2 only)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR3[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **CCR3[31:16]**: High Capture/Compare 3 value (on TIM2).

Bits 15:0 **CCR3[15:0]**: Low Capture/Compare 3 value

**If channel CC3 is configured as output:**

CCR3 is the value to be loaded in the actual capture/compare 3 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR3 register (bit OC3PE). Else the preload value is copied in the active capture/compare 3 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signaled on OC3 output.

**If channel CC3 is configured as input:**

CCR3 is the counter value transferred by the last input capture 3 event (IC3).

### 18.4.16 TIM2 and TIM3 capture/compare register 4 (TIM2\_CCR4 and TIM3\_CCR4)

Address offset: 0x40

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR4[31:16] (TIM2 only)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR4[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **CCR4[31:16]**: High Capture/Compare 4 value (on TIM2)

Bits 15:0 **CCR4[15:0]**: Low Capture/Compare 4 value

1. If CC4 channel is configured as output (CC4S bits):  
CCR4 is the value to be loaded in the actual capture/compare 4 register (preload value). It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR4 register (bit OC4PE). Otherwise, the preload value is copied in the active capture/compare 4 register when an update event occurs.  
The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signaled on OC4 output.
2. If CC4 channel is configured as input (CC4S bits in TIMx\_CCMR4 register):  
CCR4 is the counter value transferred by the last input capture 4 event (IC4).

### 18.4.17 TIM2 and TIM3 DMA control register (TIM2\_DCR and TIM3\_DCR)

Address offset: 0x48

Reset value: 0x0000

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

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

Bits 12:8 **DBL[4:0]**: DMA burst length

This 5-bit vector defines the number of DMA transfers (the timer recognizes a burst transfer when a read or a write access is done to the TIMx\_DMAR address).

- 00000: 1 transfer,
- 00001: 2 transfers,
- 00010: 3 transfers,
- ...
- 10001: 18 transfers.

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

Bits 4:0 **DBA[4:0]**: DMA base address

This 5-bit vector defines the base-address for DMA transfers (when read/write access are done through the TIMx\_DMAR address). DBA is defined as an offset starting from the address of the TIMx\_CR1 register.

Example:

- 00000: TIMx\_CR1,
- 00001: TIMx\_CR2,
- 00010: TIMx\_SMCR,
- ...

**Example:** Let us consider the following transfer: DBL = 7 transfers & DBA = TIMx\_CR1. In this case the transfer is done to/from 7 registers starting from the TIMx\_CR1 address.

#### 18.4.18 TIM2 and TIM3 DMA address for full transfer (TIM2\_DMAR and TIM3\_DMAR)

Address offset: 0x4C

Reset value: 0x0000

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

Bits 15:0 **DMAB[15:0]**: DMA register for burst accesses

A read or write operation to the DMAR register accesses the register located at the address (TIMx\_CR1 address) + (DBA + DMA index) x 4

where TIMx\_CR1 address is the address of the control register 1, DBA is the DMA base address configured in TIMx\_DCR register, DMA index is automatically controlled by the DMA transfer, and ranges from 0 to DBL (DBL configured in TIMx\_DCR).

### Example of how to use the DMA burst feature

In this example the timer DMA burst feature is used to update the contents of the CCRx registers ( $x = 2, 3, 4$ ) with the DMA transferring half words into the CCRx registers.

This is done in the following steps:

1. Configure the corresponding DMA channel as follows:
  - DMA channel peripheral address is the DMAR register address
  - DMA channel memory address is the address of the buffer in the RAM containing the data to be transferred by DMA into CCRx registers.
  - Number of data to transfer = 3 (See note below).
  - Circular mode disabled.
2. Configure the DCR register by configuring the DBA and DBL bit fields as follows:  
DBL = 3 transfers, DBA = 0xE.
3. Enable the TIMx update DMA request (set the UDE bit in the DIER register).
4. Enable TIMx
5. Enable the DMA channel

For code example refer to the Appendix section [A.9.20: Two timers synchronized by an external trigger code example](#).

Note:

*This example is for the case where every CCRx register to be updated once. If every CCRx register is to be updated twice for example, the number of data to transfer should be 6. Let us take the example of a buffer in the RAM containing data1, data2, data3, data4, data5 and data6. The data is transferred to the CCRx registers as follows: on the first update DMA request, data1 is transferred to CCR2, data2 is transferred to CCR3, data3 is transferred to CCR4 and on the second update DMA request, data4 is transferred to CCR2, data5 is transferred to CCR3 and data6 is transferred to CCR4.*

#### **18.4.19 TIM2 and TIM3 register map**

TIM2 and TIM3 registers are mapped as described in the table below:

**Table 68. TIM2 and TIM3 register map and reset values**

**Table 68. TIM2 and TIM3 register map and reset values (continued)**

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 19 General-purpose timer (TIM14)

### 19.1 TIM14 introduction

The TIM14 general-purpose timer consists of a 16-bit auto-reload counter driven by a programmable prescaler.

It may be used for a variety of purposes, including measuring the pulse lengths of input signals (input capture) or generating output waveforms (output compare, PWM).

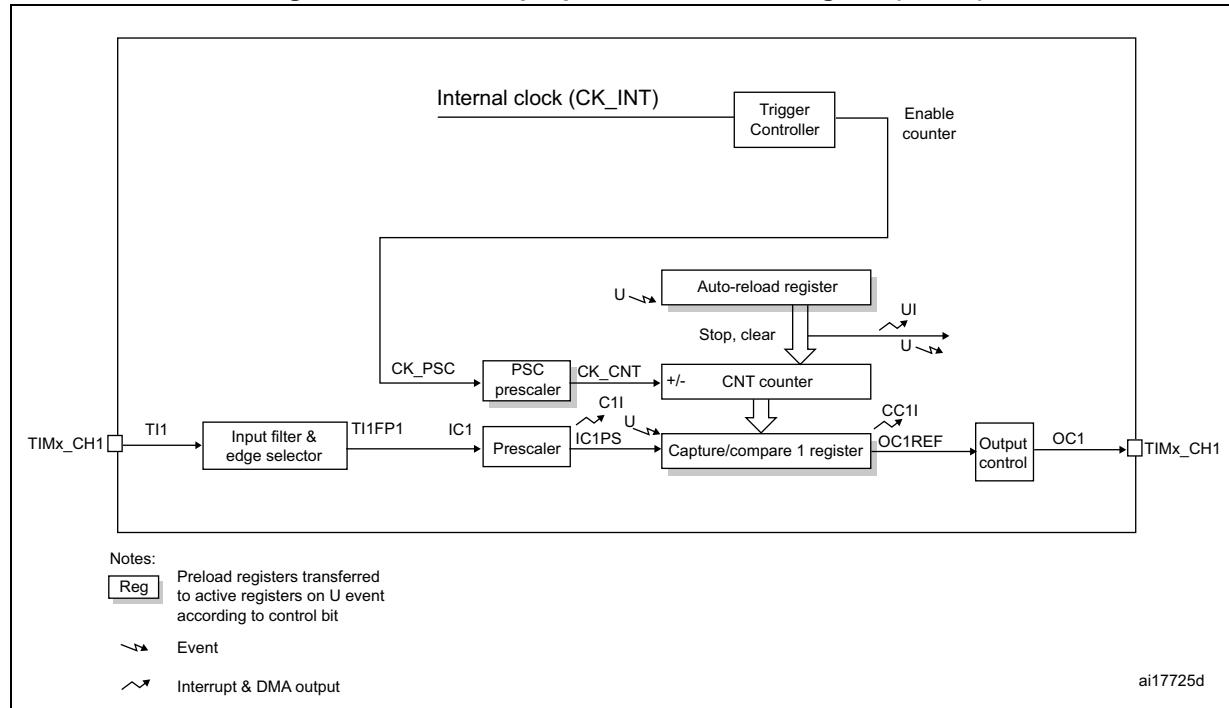
Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the RCC clock controller prescalers.

The TIM14 timer is completely independent, and does not share any resources. It can be synchronized together as described in [Section 18.3.15](#).

### 19.2 TIM14 main features

- 16-bit auto-reload upcounter
- 16-bit programmable prescaler used to divide the counter clock frequency by any factor between 1 and 65535 (can be changed “on the fly”)
- independent channel for:
  - Input capture
  - Output compare
  - PWM generation (edge-aligned mode)
- Interrupt generation on the following events:
  - Update: counter overflow, counter initialization (by software)
  - Input capture
  - Output compare

Figure 157. General-purpose timer block diagram (TIM14)



## 19.3 TIM14 functional description

### 19.3.1 Time-base unit

The main block of the programmable general-purpose timer is a 16-bit upcounter with its related auto-reload register. The counter clock can be divided by a prescaler.

The counter, the auto-reload register and the prescaler register can be written or read by software. This is true even when the counter is running.

The time-base unit includes:

- Counter register (TIMx\_CNT)
- Prescaler register (TIMx\_PSC)
- Auto-reload register (TIMx\_ARR)

The auto-reload register is preloaded. Writing to or reading from the auto-reload register accesses the preload register. The content of the preload register are transferred into the shadow register permanently or at each update event (UEV), depending on the auto-reload preload enable bit (ARPE) in TIMx\_CR1 register. The update event is sent when the counter reaches the overflow and if the UDIS bit equals 0 in the TIMx\_CR1 register. It can also be generated by software. The generation of the update event is described in detailed for each configuration.

The counter is clocked by the prescaler output CK\_CNT, which is enabled only when the counter enable bit (CEN) in TIMx\_CR1 register is set (refer also to the slave mode controller description to get more details on counter enabling).

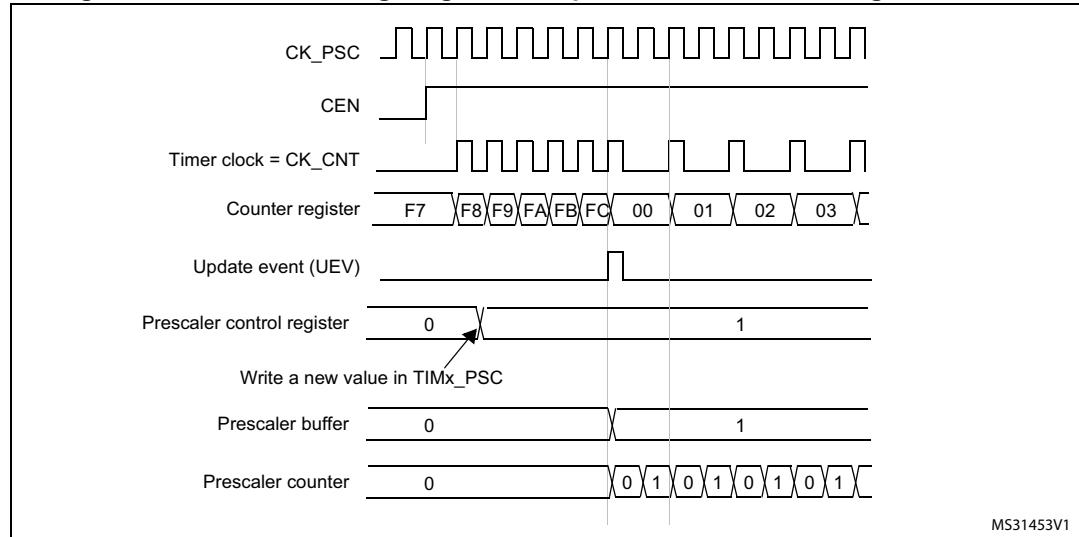
Note that the counter starts counting 1 clock cycle after setting the CEN bit in the TIMx\_CR1 register.

### Prescaler description

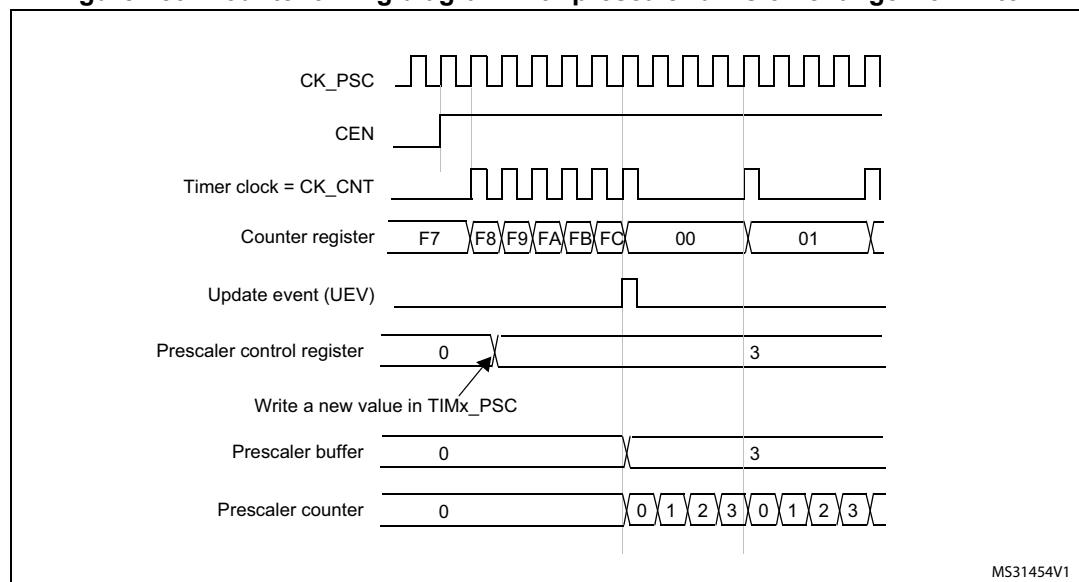
The prescaler can divide the counter clock frequency by any factor between 1 and 65536. It is based on a 16-bit counter controlled through a 16-bit register (in the TIMx\_PSC register). It can be changed on the fly as this control register is buffered. The new prescaler ratio is taken into account at the next update event.

[Figure 159](#) and [Figure 160](#) give some examples of the counter behavior when the prescaler ratio is changed on the fly.

**Figure 158. Counter timing diagram with prescaler division change from 1 to 2**



**Figure 159. Counter timing diagram with prescaler division change from 1 to 4**



### 19.3.2 Counter operation

The counter counts from 0 to the auto-reload value (content of the TIMx\_ARR register), then restarts from 0 and generates a counter overflow event.

Setting the UG bit in the TIMx\_EGR register also generates an update event.

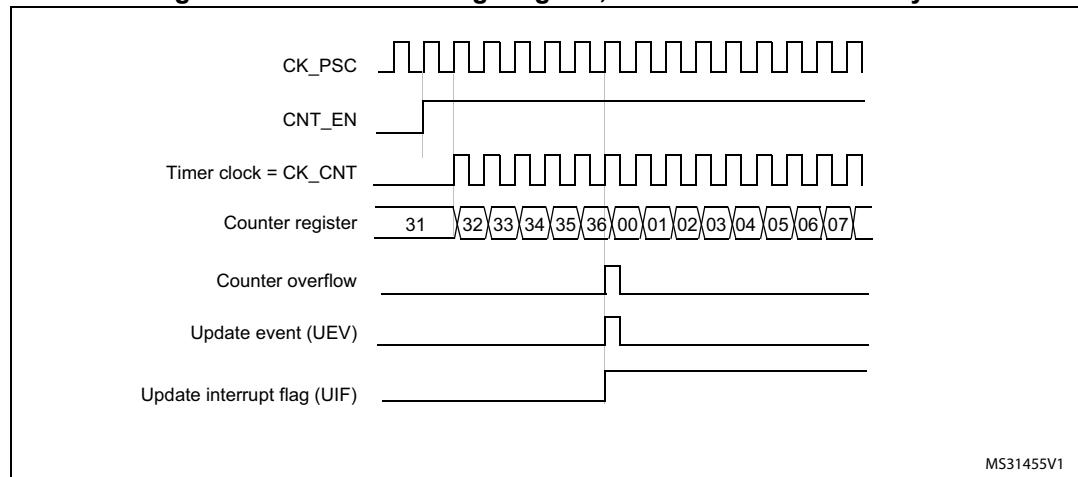
The UEV event can be disabled by software by setting the UDIS bit in the TIMx\_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until the UDIS bit has been written to 0. However, the counter restarts from 0, as well as the counter of the prescaler (but the prescale rate does not change). In addition, if the URS bit (update request selection) in TIMx\_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

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

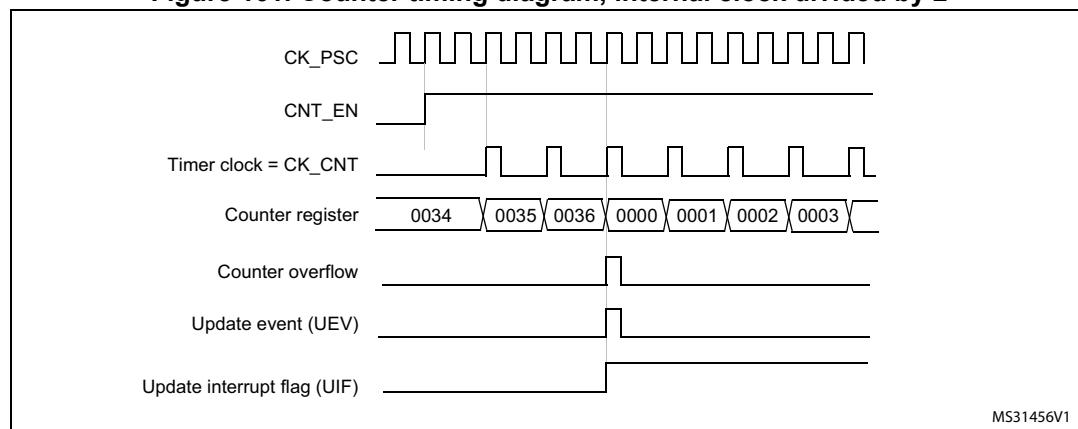
- The auto-reload shadow register is updated with the preload value (TIMx\_ARR),
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx\_PSC register).

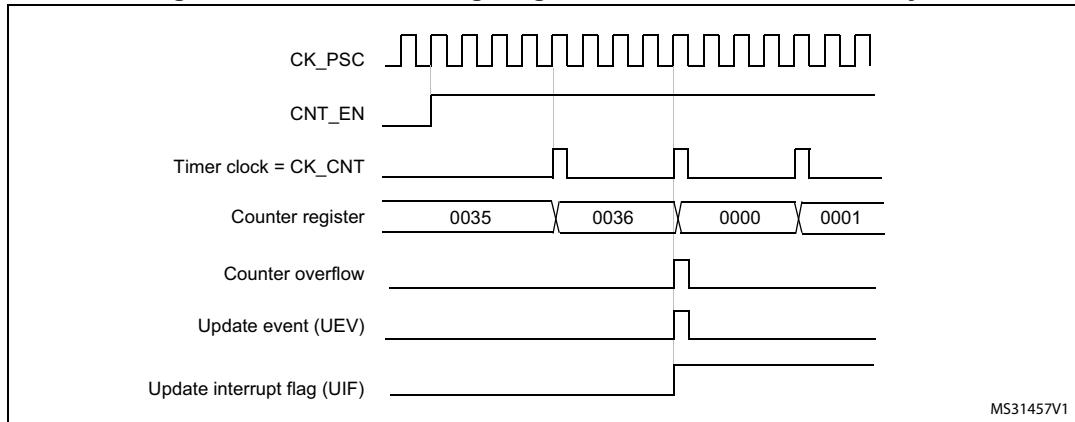
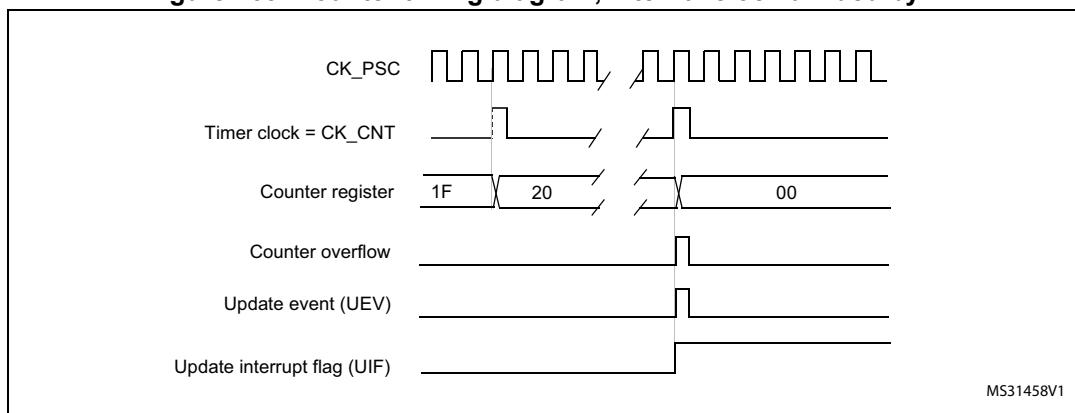
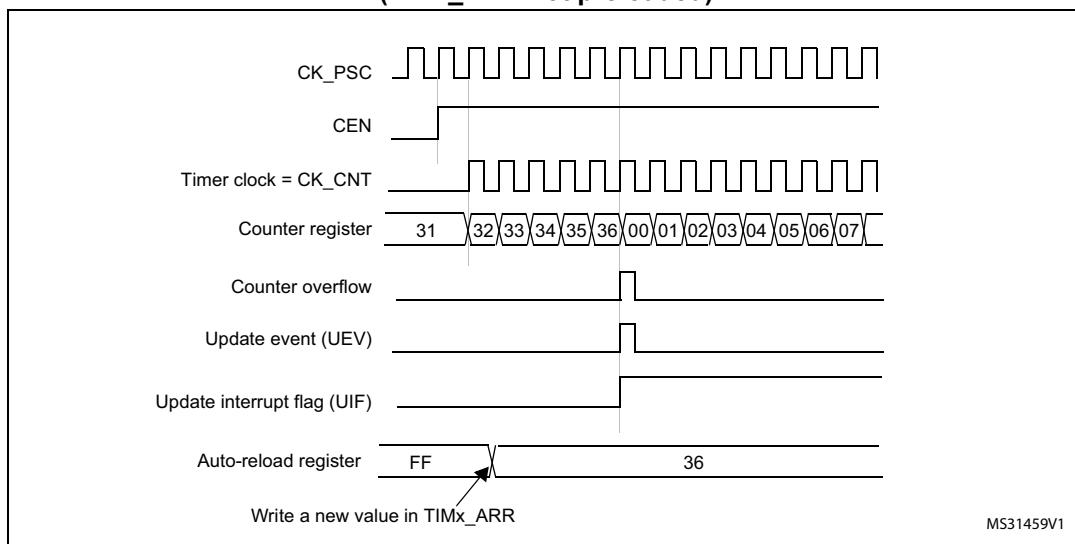
The following figures show some examples of the counter behavior for different clock frequencies when TIMx\_ARR=0x36.

**Figure 160. Counter timing diagram, internal clock divided by 1**

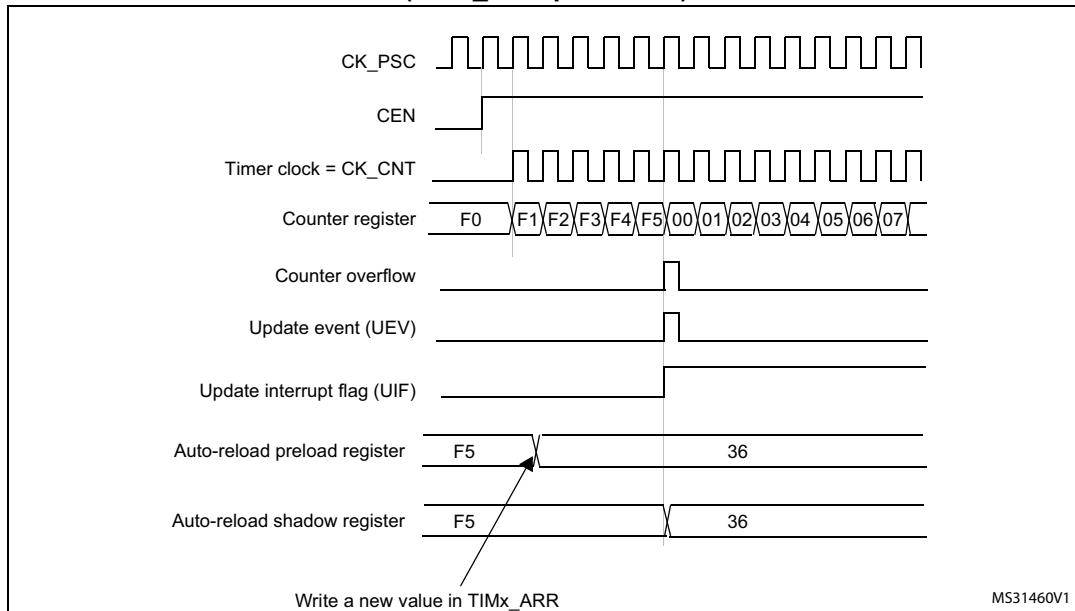


**Figure 161. Counter timing diagram, internal clock divided by 2**



**Figure 162. Counter timing diagram, internal clock divided by 4****Figure 163. Counter timing diagram, internal clock divided by N****Figure 164. Counter timing diagram, update event when ARPE=0  
(TIMx\_ARR not preloaded)**

**Figure 165. Counter timing diagram, update event when ARPE=1  
(TIMx\_ARR preloaded)**



MS31460V1

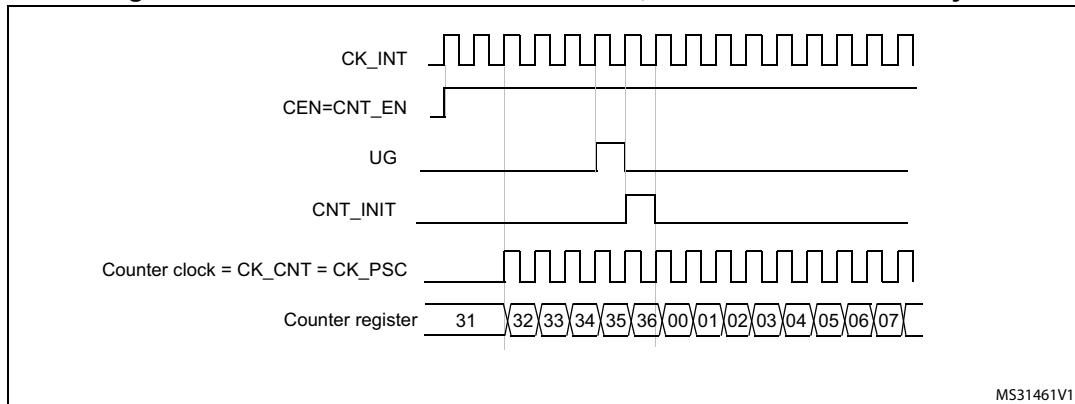
### 19.3.3 Clock source

The counter clock is provided by the Internal clock (CK\_INT) source.

The CEN (in the TIMx\_CR1 register) and UG bits (in the TIMx\_EGR register) are actual control bits and can be changed only by software (except for UG that remains cleared automatically). As soon as the CEN bit is written to 1, the prescaler is clocked by the internal clock CK\_INT.

*Figure 166* shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

**Figure 166. Control circuit in normal mode, internal clock divided by 1**



MS31461V1

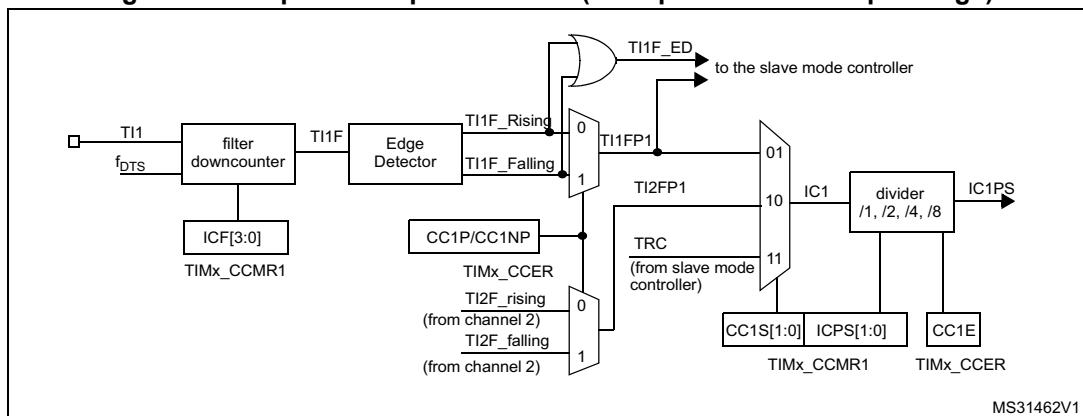
### 19.3.4 Capture/compare channels

Each Capture/Compare channel is built around a capture/compare register (including a shadow register), a input stage for capture (with digital filter, multiplexing and prescaler) and an output stage (with comparator and output control).

*Figure 167* to *Figure 169* give an overview of one capture/compare channel.

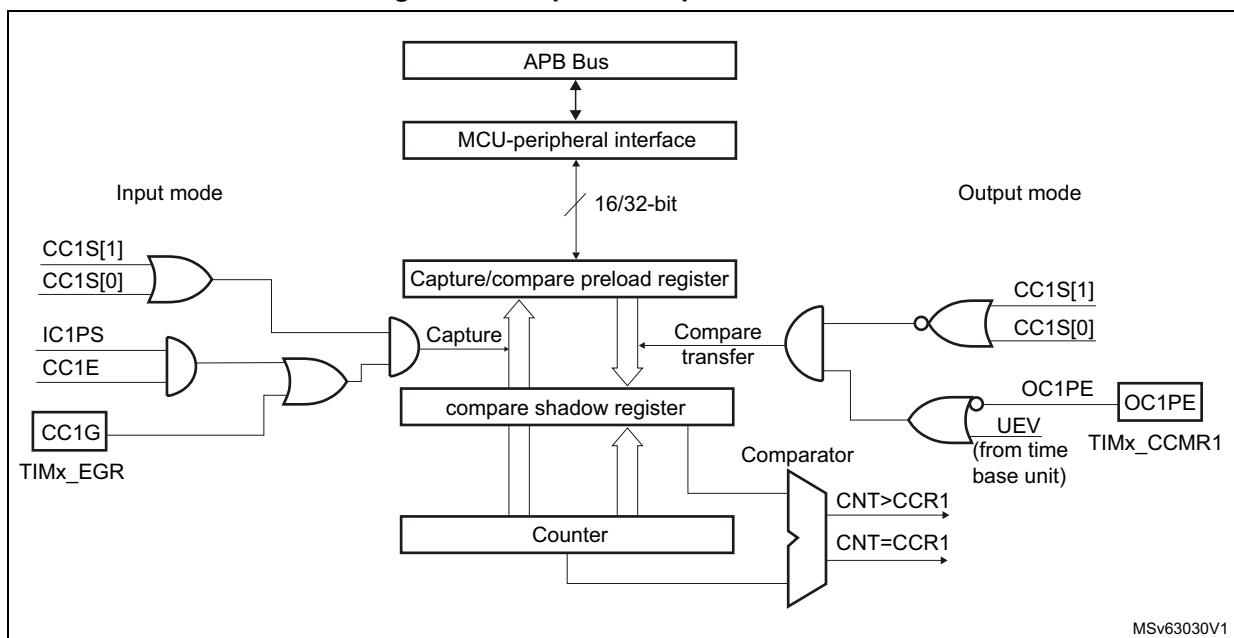
The input stage samples the corresponding TIx input to generate a filtered signal TIxF. Then, an edge detector with polarity selection generates a signal (TIxFPx) which can be used as trigger input by the slave mode controller or as the capture command. It is prescaled before the capture register (ICxPS).

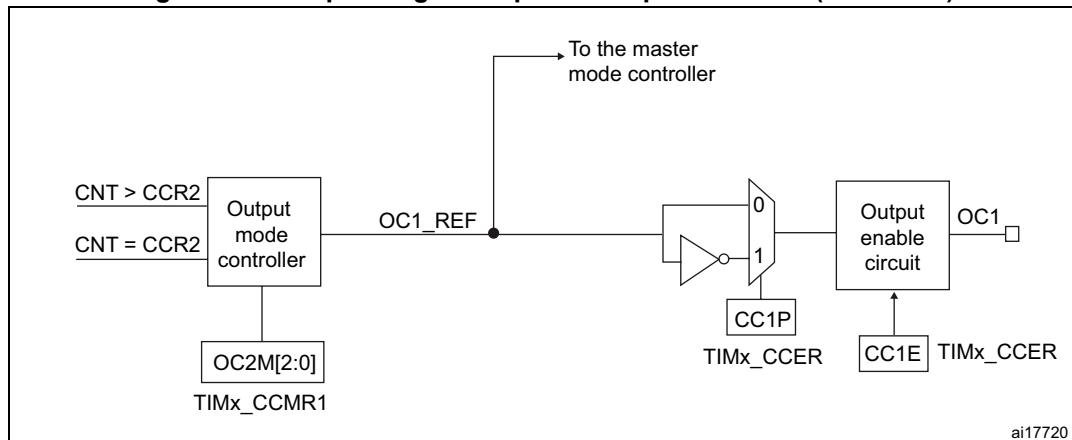
**Figure 167. Capture/compare channel (example: channel 1 input stage)**



The output stage generates an intermediate waveform which is then used for reference: OCxRef (active high). The polarity acts at the end of the chain.

**Figure 168. Capture/compare channel 1 main circuit**



**Figure 169. Output stage of capture/compare channel (channel 1)**

ai17720

The capture/compare block is made of one preload register and one shadow register. Write and read always access the preload register.

In capture mode, captures are actually done in the shadow register, which is copied into the preload register.

In compare mode, the content of the preload register is copied into the shadow register which is compared to the counter.

### 19.3.5 Input capture mode

In Input capture mode, the Capture/Compare Registers (TIMx\_CCRx) are used to latch the value of the counter after a transition detected by the corresponding IC<sub>x</sub> signal. When a capture occurs, the corresponding CC<sub>x</sub>IF flag (TIMx\_SR register) is set and an interrupt or a DMA request can be sent if they are enabled. If a capture occurs while the CC<sub>x</sub>IF flag was already high, then the over-capture flag CC<sub>x</sub>OF (TIMx\_SR register) is set. CC<sub>x</sub>IF can be cleared by software by writing it to '0' or by reading the captured data stored in the TIMx\_CCRx register. CC<sub>x</sub>OF is cleared when it is written with 0.

The following example shows how to capture the counter value in TIMx\_CCR1 when TI1 input rises. To do this, use the following procedure:

1. Select the active input: TIMx\_CCR1 must be linked to the TI1 input, so write the CC1S bits to '01' in the TIMx\_CCMR1 register. As soon as CC1S becomes different from '00', the channel is configured in input mode and the TIMx\_CCR1 register becomes read-only.
2. Program the appropriate input filter duration in relation with the signal connected to the timer (when the input is one of the TI<sub>x</sub> (IC<sub>x</sub>F bits in the TIMx\_CCMRx register). Let us imagine that, when toggling, the input signal is not stable during at most 5 internal clock cycles. We must program a filter duration longer than these 5 clock cycles. We can validate a transition on TI1 when 8 consecutive samples with the new level have been

- detected (sampled at  $f_{DTS}$  frequency). Then write IC1F bits to '0011' in the TIMx\_CCMR1 register.
3. Select the edge of the active transition on the TI1 channel by programming CC1P and CC1NP bits to '00' in the TIMx\_CCER register (rising edge in this case).
  4. Program the input prescaler. In our example, we wish the capture to be performed at each valid transition, so the prescaler is disabled (write IC1PS bits to '00' in the TIMx\_CCMR1 register).
  5. Enable capture from the counter into the capture register by setting the CC1E bit in the TIMx\_CCER register.
  6. If needed, enable the related interrupt request by setting the CC1IE bit in the TIMx\_DIER register.

For code example refer to the Appendix section [A.9.3: Input capture configuration code example](#).

When an input capture occurs:

- The TIMx\_CCR1 register gets the value of the counter on the active transition.
- CC1IF flag is set (interrupt flag). CC1OF is also set if at least two consecutive captures occurred whereas the flag was not cleared.
- An interrupt is generated depending on the CC1IE bit.

For code example refer to the Appendix section [A.9.4: Input capture data management code example](#).

In order to handle the overcapture, it is recommended to read the data before the overcapture flag. This is to avoid missing an overcapture which could happen after reading the flag and before reading the data.

*Note:* IC interrupt requests can be generated by software by setting the corresponding CCxG bit in the TIMx\_EGR register.

### 19.3.6 Forced output mode

In output mode (CCxS bits = '00' in the TIMx\_CCMRx register), each output compare signal (OCxREF and then OCx) can be forced to active or inactive level directly by software, independently of any comparison between the output compare register and the counter.

To force an output compare signal (OCXREF/OCx) to its active level, one just needs to write '101' in the OCxM bits in the corresponding TIMx\_CCMRx register. Thus OCXREF is forced high (OCxREF is always active high) and OCx get opposite value to CCxP polarity bit.

For example: CCxP='0' (OCx active high) => OCx is forced to high level.

The OCxREF signal can be forced low by writing the OCxM bits to '100' in the TIMx\_CCMRx register.

The comparison between the TIMx\_CCRx shadow register and the counter is still performed and allows the flag to be set. Interrupt requests can be sent accordingly. This is described in the output compare mode section below.

### 19.3.7 Output compare mode

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

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

1. Assigns the corresponding output pin to a programmable value defined by the output compare mode (OCxM bits in the TIMx\_CCMRx register) and the output polarity (CCxP bit in the TIMx\_CCER register). The output pin can keep its level (OCXM='000'), be set active (OCXM='001'), be set inactive (OCXM='010') or can toggle (OCXM='011') on match.
2. Sets a flag in the interrupt status register (CCxIF bit in the TIMx\_SR register).
3. Generates an interrupt if the corresponding interrupt mask is set (CCXIE bit in the TIMx\_DIER register).

The TIMx\_CCRx registers can be programmed with or without preload registers using the OCxPE bit in the TIMx\_CCMRx register.

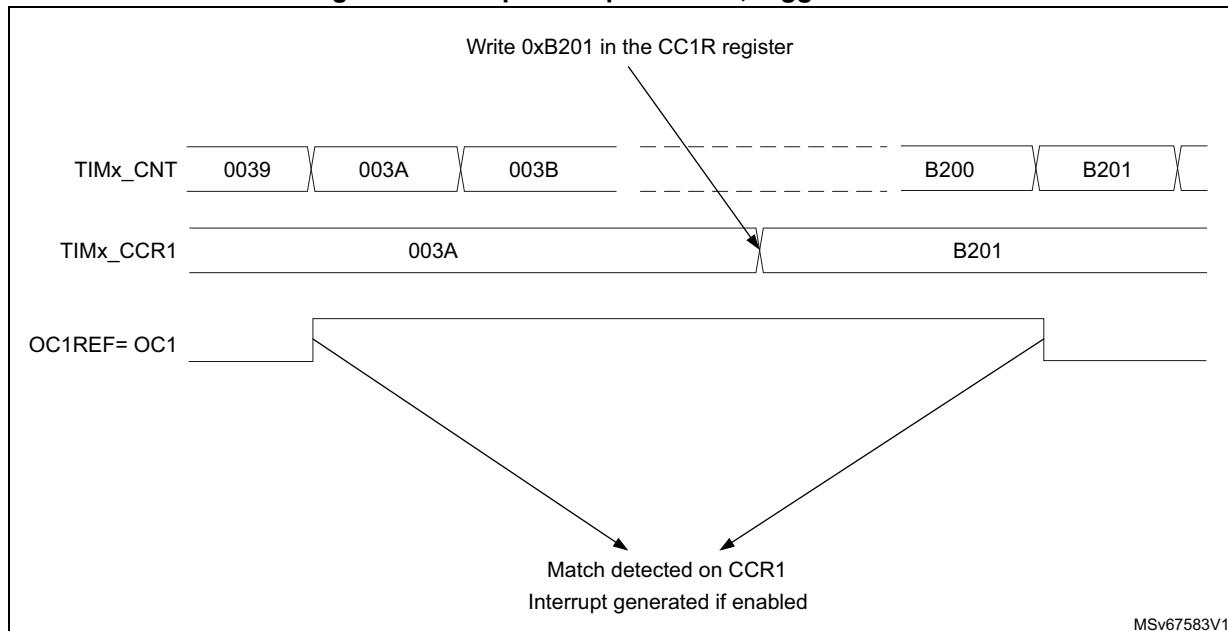
In output compare mode, the update event UEV has no effect on OCxREF and OCx output. The timing resolution is one count of the counter. Output compare mode can also be used to output a single pulse (in One-pulse mode).

Procedure:

1. Select the counter clock (internal, external, prescaler).
2. Write the desired data in the TIMx\_ARR and TIMx\_CCRx registers.
3. Set the CCxIE bit if an interrupt request is to be generated.
4. Select the output mode. For example:
  - Write OCxM = '011' to toggle OCx output pin when CNT matches CCRx
  - Write OCxPE = '0' to disable preload register
  - Write CCxP = '0' to select active high polarity
  - Write CCxE = '1' to enable the output
5. Enable the counter by setting the CEN bit in the TIMx\_CR1 register.

For code example refer to the Appendix section [A.9.7: Output compare configuration code example](#).

The TIMx\_CCRx register can be updated at any time by software to control the output waveform, provided that the preload register is not enabled (OCxPE='0', else TIMx\_CCRx shadow register is updated only at the next update event UEV). An example is given in [Figure 170](#).

**Figure 170. Output compare mode, toggle on OC1**

### 19.3.8 PWM mode

Pulse Width Modulation mode allows to generate a signal with a frequency determined by the value of the TIMx\_ARR register and a duty cycle determined by the value of the TIMx\_CCRx register.

The PWM mode can be selected independently on each channel (one PWM per OCx output) by writing ‘110’ (PWM mode 1) or ‘111’ (PWM mode 2) in the OCxM bits in the TIMx\_CCMRx register. The corresponding preload register must be enabled by setting the OCxPE bit in the TIMx\_CCMRx register, and eventually the auto-reload preload register by setting the ARPE bit in the TIMx\_CR1 register.

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

The OCx polarity is software programmable using the CCxP bit in the TIMx\_CCER register. It can be programmed as active high or active low. The OCx output is enabled by the CCxE bit in the TIMx\_CCER register. Refer to the TIMx\_CCERx register description for more details.

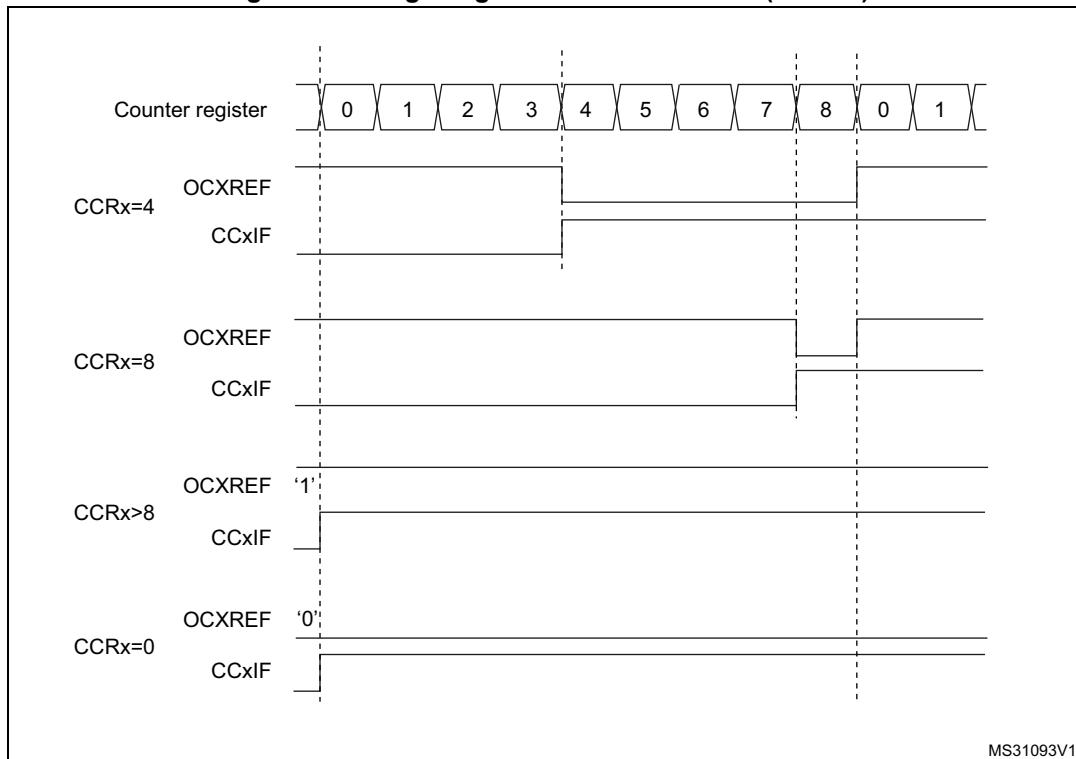
In PWM mode (1 or 2), TIMx\_CNT and TIMx\_CCRx are always compared to determine whether  $\text{TIMx\_CNT} \leq \text{TIMx\_CCRx}$ .

The timer is able to generate PWM in edge-aligned mode only since the counter is upcounting.

#### PWM edge-aligned mode

In the following example, we consider PWM mode 1. The reference PWM signal OCxREF is high as long as  $\text{TIMx\_CNT} < \text{TIMx\_CCRx}$  else it becomes low. If the compare value in TIMx\_CCRx is greater than the auto-reload value (in TIMx\_ARR) then OCxREF is held at ‘1’. If the compare value is 0 then OCxRef is held at ‘0’. [Figure 171](#) shows some edge-aligned PWM waveforms in an example where TIMx\_ARR=8.

Figure 171. Edge-aligned PWM waveforms (ARR=8)



For code example refer to the Appendix section [A.9.8: Edge-aligned PWM configuration example](#).

### 19.3.9 Debug mode

When the microcontroller enters debug mode (Cortex™-M0 core halted), the TIMx counter either continues to work normally or stops, depending on DBG\_TIMx\_STOP configuration bit in DBG module.

## 19.4 TIM14 registers

### 19.4.1 TIM14 control register 1 (TIM14\_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	CKD[1:0]	ARPE	Res.	Res.	Res.	Res.	URS	UDIS	CEN	

Bits 15:10 Reserved, must be kept at reset value.

**Bits 9:8 CKD[1:0]: Clock division**

This bit-field indicates the division ratio between the timer clock (CK\_INT) frequency and sampling clock used by the digital filters (ETR, TIx),

- 00:  $t_{DTS} = t_{CK\_INT}$
- 01:  $t_{DTS} = 2 \times t_{CK\_INT}$
- 10:  $t_{DTS} = 4 \times t_{CK\_INT}$
- 11: Reserved

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

- 0: TIMx\_ARR register is not buffered
- 1: TIMx\_ARR register is buffered

Bits 6:3 Reserved, must be kept at reset value.

**Bit 2 URS: Update request source**

This bit is set and cleared by software to select the update interrupt (UEV) sources.

0: Any of the following events generate an UEV if enabled:

- Counter overflow
- Setting the UG bit

1: Only counter overflow generates an UEV if enabled.

**Bit 1 UDIS: Update disable**

This bit is set and cleared by software to enable/disable update interrupt (UEV) event generation.

0: UEV enabled. An UEV is generated by one of the following events:

- Counter overflow
- Setting the UG bit.

Buffered registers are then loaded with their preload values.

1: UEV disabled. No UEV is generated, shadow registers keep their value (ARR, PSC, CCRx). The counter and the prescaler are reinitialized if the UG bit is set.

**Bit 0 CEN: Counter enable**

- 0: Counter disabled
- 1: Counter enabled

## 19.4.2 TIM14 interrupt enable register (TIM14\_DIER)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CC1IE	UIE													

Bits 15:2 Reserved, must be kept at reset value.

**Bit 1 CC1IE: Capture/Compare 1 interrupt enable**

- 0: CC1 interrupt disabled
- 1: CC1 interrupt enabled

**Bit 0 UIE: Update interrupt enable**

- 0: Update interrupt disabled
- 1: Update interrupt enabled

### 19.4.3 TIM14 status register (TIM14\_SR)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	CC1OF	Res.	CC1IF	UIF						
						rc_w0								rc_w0	rc_w0

Bits 15:10 Reserved, must be kept at reset value.

Bit 9 **CC1OF**: Capture/Compare 1 overcapture flag

This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to '0'.

0: No overcapture has been detected.

1: The counter value has been captured in TIMx\_CCR1 register while CC1IF flag was already set

Bits 8:2 Reserved, must be kept at reset value.

Bit 1 **CC1IF**: Capture/compare 1 interrupt flag

**Condition: channel CC1 is configured as output**

This flag is set by hardware when the counter matches the compare value. It is cleared by software.

0: No match.

1: The content of the counter TIMx\_CNT matches the content of the TIMx\_CCR1 register. When the contents of TIMx\_CCR1 are greater than the contents of TIMx\_ARR, the CC1IF bit goes high on the counter overflow.

**Condition: channel CC1 is configured as input**

This bit is set by hardware on a capture. It is cleared by software or by reading the TIMx\_CCR1 register.

0: No input capture occurred.

1: The counter value has been captured in TIMx\_CCR1 register (an edge has been detected on IC1 which matches the selected polarity).

Bit 0 **UIF**: Update interrupt flag

This bit is set by hardware on an update event. It is cleared by software.

0: No update occurred.

1: Update interrupt pending. This bit is set by hardware when the registers are updated:

- At overflow and if UDIS='0' in the TIMx\_CR1 register.
- When CNT is reinitialized by software using the UG bit in TIMx\_EGR register, if URS='0' and UDIS='0' in the TIMx\_CR1 register.

### 19.4.4 TIM14 event generation register (TIM14\_EGR)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CC1G	UG													
														w	w

Bits 15:2 Reserved, must be kept at reset value.

Bit 1 **CC1G**: Capture/compare 1 generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: A capture/compare event is generated on channel 1:

**Condition: channel CC1 is configured as output**

CC1IF flag is set, Corresponding interrupt or is sent if enabled.

**Condition: channel CC1 is configured as input**

The current value of the counter is captured in TIMx\_CCR1 register. The CC1IF flag is set, the corresponding interrupt is sent if enabled. The CC1OF flag is set if the CC1IF flag was already high.

Bit 0 **UG**: Update generation

This bit can be set by software, it is automatically cleared by hardware.

0: No action

1: Re-initialize the counter and generates an update of the registers. Note that the prescaler counter is cleared too (anyway the prescaler ratio is not affected). The counter is cleared.

#### 19.4.5 TIM14 capture/compare mode register 1 [alternate] (TIM14\_CCMR1)

Address offset: 0x18

Reset value: 0x0000

The same register can be used for input capture mode (this section) or for output compare mode (next section).. The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OCxx describes its function when the channel is configured in output, ICxx describes its function when the channel is configured in input. So one must take care that the same bit can have a different meaning for the input stage and for the output stage

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]								
								rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:8 Reserved, must be kept at reset value.

Bits 7:4 **IC1F[3:0]**: Input capture 1 filter

This bit-field defines the frequency used to sample TI1 input and the length of the digital filter applied to TI1. The digital filter is made of an event counter in which N events are needed to validate a transition on the output:

- 0000: No filter, sampling is done at  $f_{DTS}$
- 0001:  $f_{SAMPLING} = f_{CK\_INT}$ ,  $N = 2$
- 0010:  $f_{SAMPLING} = f_{CK\_INT}$ ,  $N = 4$
- 0011:  $f_{SAMPLING} = f_{CK\_INT}$ ,  $N = 8$
- 0100:  $f_{SAMPLING} = f_{DTS} / 2$ ,  $N = 6$
- 0101:  $f_{SAMPLING} = f_{DTS} / 2$ ,  $N = 8$
- 0110:  $f_{SAMPLING} = f_{DTS} / 4$ ,  $N = 6$
- 0111:  $f_{SAMPLING} = f_{DTS} / 4$ ,  $N = 8$
- 1000:  $f_{SAMPLING} = f_{DTS} / 8$ ,  $N = 6$
- 1001:  $f_{SAMPLING} = f_{DTS} / 8$ ,  $N = 8$
- 1010:  $f_{SAMPLING} = f_{DTS} / 16$ ,  $N = 5$
- 1011:  $f_{SAMPLING} = f_{DTS} / 16$ ,  $N = 6$
- 1100:  $f_{SAMPLING} = f_{DTS} / 16$ ,  $N = 8$
- 1101:  $f_{SAMPLING} = f_{DTS} / 32$ ,  $N = 5$
- 1110:  $f_{SAMPLING} = f_{DTS} / 32$ ,  $N = 6$
- 1111:  $f_{SAMPLING} = f_{DTS} / 32$ ,  $N = 8$

*Note: Care must be taken that  $f_{DTS}$  is replaced in the formula by  $CK\_INT$  when  $ICxF[3:0] = 1$ , 2 or 3.*

Bits 3:2 **IC1PSC[1:0]**: Input capture 1 prescaler

This bit-field defines the ratio of the prescaler acting on CC1 input (IC1).

The prescaler is reset as soon as  $CC1E='0'$  (TIMx\_CCER register).

00: no prescaler, capture is done each time an edge is detected on the capture input

01: capture is done once every 2 events

10: capture is done once every 4 events

11: capture is done once every 8 events

Bits 1:0 **CC1S[1:0]**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output

01: CC1 channel is configured as input, IC1 is mapped on TI1

Other: Reserved

*Note: CC1S bits are writable only when the channel is OFF ( $CC1E = 0$  in TIMx\_CCER).*

**19.4.6 TIM14 capture/compare mode register 1 [alternate] (TIM14\_CCMR1)**

Address offset: 0x18

Reset value: 0x0000

The same register can be used for output compare mode (this section) or for input capture mode (previous section). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OCxx describes its function when the channel is configured in output, ICxx describes its function when the channel is configured in input. So one must take care that the same bit can have a different meaning for the input stage and for the output stage.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	OC1M[2:0]	OC1PE	OC1FE	CC1S[1:0]											
									rw	rw	rw	rw	rw	rw	rw

Bits 15:7 Reserved, must be kept at reset value.

Bits 6:4 **OC1M[2:0]**: Output compare 1 mode

These bits define the behavior of the output reference signal OC1REF from which OC1 is derived. OC1REF is active high whereas OC1 active level depends on CC1P bit.

000: Frozen. The comparison between the output compare register TIMx\_CCR1 and the counter TIMx\_CNT has no effect on the outputs.

001: Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

011: Toggle - OC1REF toggles when TIMx\_CNT = TIMx\_CCR1.

100: Force inactive level - OC1REF is forced low.

101: Force active level - OC1REF is forced high.

110: PWM mode 1 - Channel 1 is active as long as TIMx\_CNT < TIMx\_CCR1 else inactive.

111: PWM mode 2 - Channel 1 is inactive as long as TIMx\_CNT < TIMx\_CCR1 else active.

*Note: In PWM mode 1 or 2, the OCREF level changes when the result of the comparison changes or when the output compare mode switches from frozen to PWM mode.*

Bit 3 **OC1PE**: Output compare 1 preload enable

0: Preload register on TIMx\_CCR1 disabled. TIMx\_CCR1 can be written at anytime, the new value is taken in account immediately.

1: Preload register on TIMx\_CCR1 enabled. Read/Write operations access the preload register. TIMx\_CCR1 preload value is loaded in the active register at each update event.

*Note: The PWM mode can be used without validating the preload register only in one pulse mode (OPM bit set in TIMx\_CR1 register). Else the behavior is not guaranteed.*

Bit 2 **OC1FE**: Output compare 1 fast enable

This bit is used to accelerate the effect of an event on the trigger in input on the CC output.

0: CC1 behaves normally depending on counter and CCR1 values even when the trigger is ON. The minimum delay to activate CC1 output when an edge occurs on the trigger input is 5 clock cycles.

1: An active edge on the trigger input acts like a compare match on CC1 output. OC is then set to the compare level independently of the result of the comparison. Delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. OC1FE acts only if the channel is configured in PWM1 or PWM2 mode.

Bits 1:0 **CC1S[1:0]**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output.

01: CC1 channel is configured as input, IC1 is mapped on TI1.

10: Reserved

11: Reserved

*Note: CC1S bits are writable only when the channel is OFF (CC1E = 0 in TIMx\_CCER).*

## 19.4.7 TIM14 capture/compare enable register (TIM14\_CCER)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CC1NP	Res.	CC1P	CC1E											

Bits 15:4 Reserved, must be kept at reset value.

Bit 3 **CC1NP**: Capture/Compare 1 complementary output Polarity.

CC1 channel configured as output: CC1NP must be kept cleared.

CC1 channel configured as input: CC1NP bit is used in conjunction with CC1P to define TI1FP1 polarity (refer to CC1P description).

Bit 2 Reserved, must be kept at reset value.

Bit 1 **CC1P**: Capture/Compare 1 output Polarity.

**Condition: CC1 channel configured as output**

0: OC1 active high

1: OC1 active low

**Condition: CC1 channel configured as input**

The CC1P bit selects TI1FP1 and TI2FP1 polarity for trigger or capture operations.

00: noninverted/rising edge

Circuit is sensitive to TI1FP1 rising edge (capture mode), TI1FP1 is not inverted.

01: inverted/falling edge

Circuit is sensitive to TI1FP1 falling edge (capture mode), TI1FP1 is inverted.

10: reserved, do not use this configuration.

11: noninverted/both edges

Circuit is sensitive to both TI1FP1 rising and falling edges (capture mode), TI1FP1 is not inverted.

Bit 0 **CC1E**: Capture/Compare 1 output enable.

**Condition: CC1 channel configured as output**:

0: Off - OC1 is not active

1: On - OC1 signal is output on the corresponding output pin

**Condition: CC1 channel configured as input**:

This bit determines if a capture of the counter value can actually be done into the input capture/compare register 1 (TIMx\_CCR1) or not.

0: Capture disabled

1: Capture enabled

**Table 69. Output control bit for standard OCx channels**

CCxE bit	OCx output state
0	Output Disabled (OCx='0', OCx_EN='0')
1	OCx=OCxREF + Polarity, OCx_EN='1'

**Note:** The state of the external I/O pins connected to the standard OCx channels depends on the OCx channel state and the GPIO registers.

#### 19.4.8 TIM14 counter (TIM14\_CNT)

Address offset: 0x24

Reset value: 0x0000

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

Bits 15:0 **CNT[15:0]**: Counter value

### 19.4.9 TIM14 prescaler (TIM14\_PSC)

Address offset: 0x28

Reset value: 0x0000

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

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK\_CNT is equal to  $f_{CK\_PSC} / (PSC[15:0] + 1)$ .

PSC contains the value to be loaded in the active prescaler register at each update event.

### 19.4.10 TIM14 auto-reload register (TIM14\_ARR)

Address offset: 0x2C

Reset value: 0xFFFF

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

Bits 15:0 **ARR[15:0]**: Auto-reload value

ARR is the value to be loaded in the actual auto-reload register.

Refer to the [Section 19.3.1: Time-base unit on page 472](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

### 19.4.11 TIM14 capture/compare register 1 (TIM14\_CCR1)

Address offset: 0x34

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR1[15:0]**: Capture/Compare 1 value

**Condition: channel CC1 is configured as output**

CCR1 is the value to be loaded in the actual capture/compare 1 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR1 register (bit OC1PE). Else the preload value is copied in the active capture/compare 1 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signaled on OC1 output.

**Condition: channel CC1 is configured as input**

CCR1 is the counter value transferred by the last input capture 1 event (IC1).

### 19.4.12 TIM14 option register (TIM14\_OR)

Address offset: 0x50

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TI1_RMP[1:0]	rw rw													

Bits 15:2 Reserved, must be kept at reset value.

Bits 1:0 **TI1\_RMP[1:0]:** Timer Input 1 remap

Set and cleared by software.

00: TIM14 Channel1 is connected to the GPIO. Refer to the alternate function mapping in the device datasheets.

01: TIM14 Channel1 is connected to the RTCCLK.

10: TIM14 Channel1 is connected to the HSE/32 Clock.

11: TIM14 Channel1 is connected to the microcontroller clock output (MCO), this selection is controlled by the MCO[2:0] bits of the Clock configuration register (RCC\_CFGR) (see [Section 6.4.2: Clock configuration register \(RCC\\_CFGR\)](#)).

### 19.4.13 TIM14 register map

TIM14 registers are mapped as 16-bit addressable registers as described in the table below:

**Table 70. TIM14 register map and reset values**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
0x00	<b>TIM14_CR1</b>	Res.																																		
	Reset value																																			
0x08	Reserved																																			
0x0C	<b>TIM14_DIER</b>	Res.																																		
	Reset value																																			
0x10	<b>TIM14_SR</b>	Res.																																		
	Reset value																																			
0x14	<b>TIM14_EGR</b>	Res.																																		
	Reset value																																			
0x18	<b>TIM14_CCMR1</b> Output compare mode	Res.																																		
	Reset value																																			
	<b>TIM14_CCMR1</b> Input capture mode	Res.																																		
	Reset value																																			
0x1C	<b>Reserved</b>	Res.																																		
	Reset value																																			

**Table 70. TIM14 register map and reset values (continued)**

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 20 General-purpose timers (TIM15/16/17)

TIM15 is not available on STM32F03x devices.

### 20.1 TIM15/16/17 introduction

The TIM15/16/17 timers consist of a 16-bit auto-reload counter driven by a programmable prescaler.

They may be used for a variety of purposes, including measuring the pulse lengths of input signals (input capture) or generating output waveforms (output compare, PWM, complementary PWM with dead-time insertion).

Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the RCC clock controller prescalers.

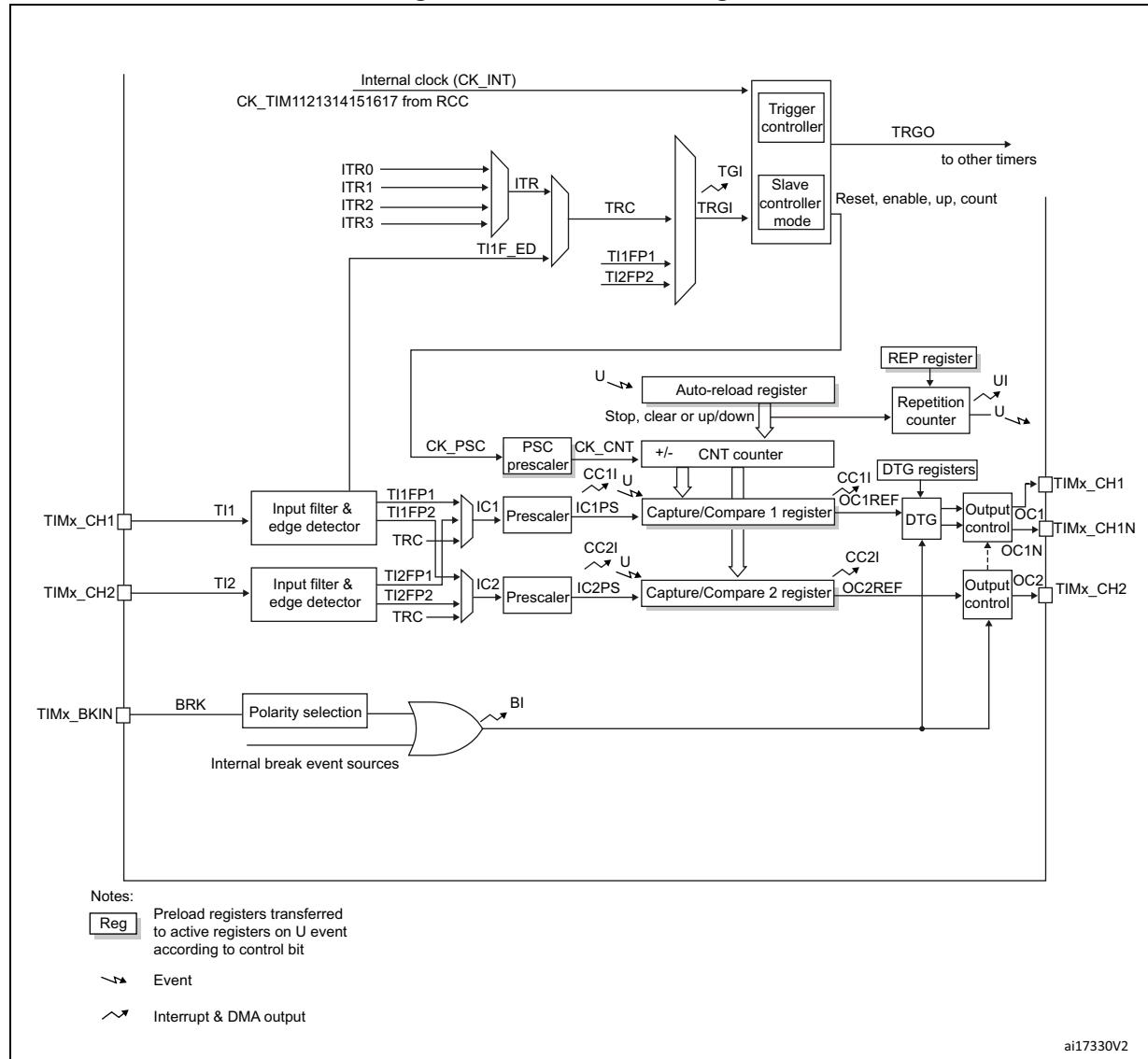
The TIM15/16/17 timers are completely independent, and do not share any resources. The TIM15 can be synchronized with other timers.

### 20.2 TIM15 main features

TIM15 includes the following features:

- 16-bit auto-reload upcounter
- 16-bit programmable prescaler used to divide (also “on the fly”) the counter clock frequency by any factor between 1 and 65535
- Up to 2 independent channels for:
  - Input capture
  - Output compare
  - PWM generation (Edge-aligned mode)
  - One-pulse mode output
- Complementary outputs with programmable dead-time (for channel 1 only)
- Synchronization circuit to control the timer with external signals and to interconnect several timers together
- Repetition counter to update the timer registers only after a given number of cycles of the counter
- Break input to put the timer’s output signals in the reset state or a known state
- Interrupt/DMA generation on the following events:
  - Update: counter overflow, counter initialization (by software or internal/external trigger)
  - Trigger event (counter start, stop, initialization or count by internal/external trigger)
  - Input capture
  - Output compare
  - Break input (interrupt request)

Figure 172. TIM15 block diagram

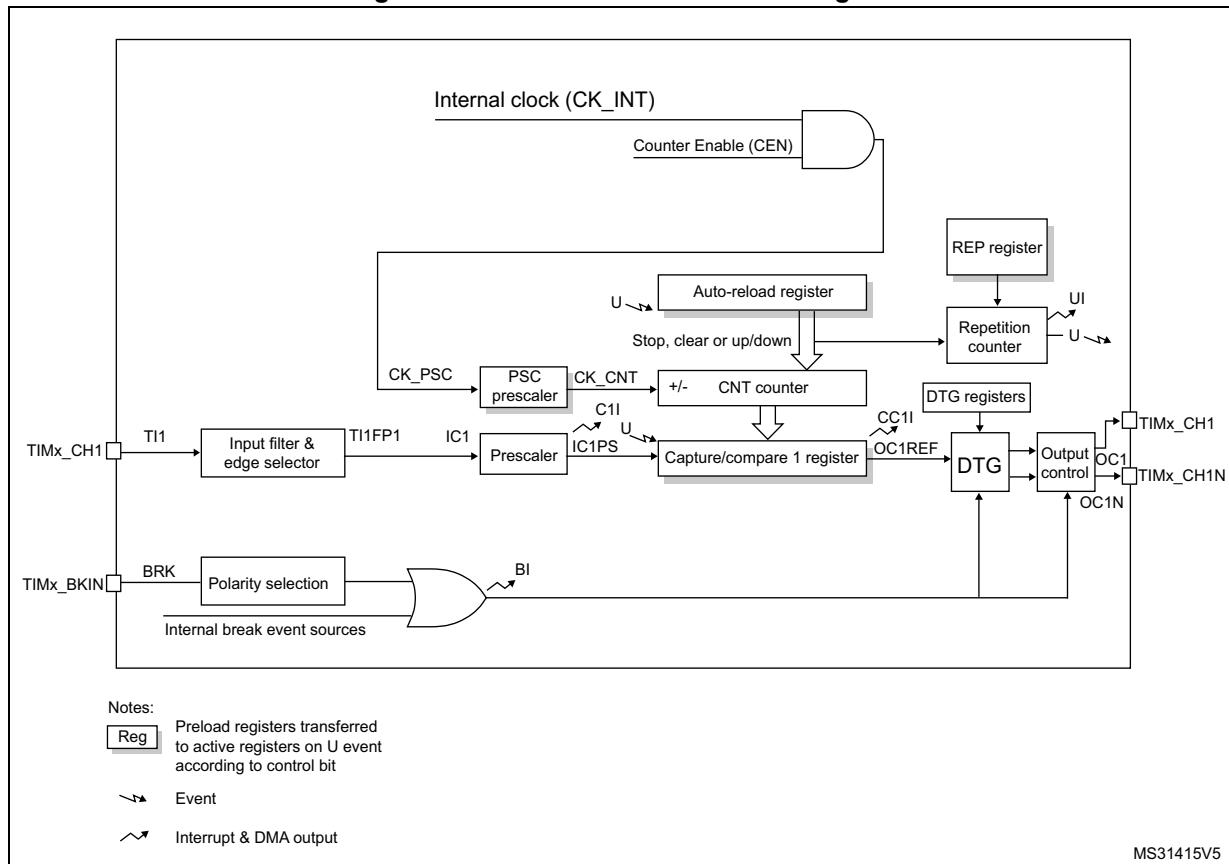


## 20.3 TIM16 and TIM17 main features

The TIM16 and TIM17 timers include the following features:

- 16-bit auto-reload upcounter
- 16-bit programmable prescaler used to divide (also “on the fly”) the counter clock frequency by any factor between 1 and 65535
- One channel for:
  - Input capture
  - Output compare
  - PWM generation (Edge-aligned mode)
  - One-pulse mode output
- Complementary outputs with programmable dead-time
- Repetition counter to update the timer registers only after a given number of cycles of the counter
- Break input to put the timer’s output signals in the reset state or a known state
- Interrupt/DMA generation on the following events:
  - Update: counter overflow
  - Input capture
  - Output compare
  - Break input

Figure 173. TIM16 and TIM17 block diagram



## 20.4 TIM15/16/17 functional description

### 20.4.1 Time-base unit

The main block of the programmable general purpose timer is a 16-bit upcounter with its related auto-reload register. The counter clock can be divided by a prescaler.

The counter, the auto-reload register and the prescaler register can be written or read by software. This is true even when the counter is running.

The time-base unit includes:

- Counter register (TIMx\_CNT)
- Prescaler register (TIMx\_PSC)
- Auto-reload register (TIMx\_ARR)
- Repetition counter register (TIMx\_RCR)

The auto-reload register is preloaded. Writing to or reading from the auto-reload register accesses the preload register. The content of the preload register are transferred into the shadow register permanently or at each update event (UEV), depending on the auto-reload preload enable bit (ARPE) in TIMx\_CR1 register. The update event is sent when the counter reaches the overflow (or underflow when downcounting) and if the UDIS bit equals 0 in the

**TIMx\_CR1** register. It can also be generated by software. The generation of the update event is described in detailed for each configuration.

The counter is clocked by the prescaler output CK\_CNT, which is enabled only when the counter enable bit (CEN) in TIMx\_CR1 register is set (refer also to the slave mode controller description to get more details on counter enabling).

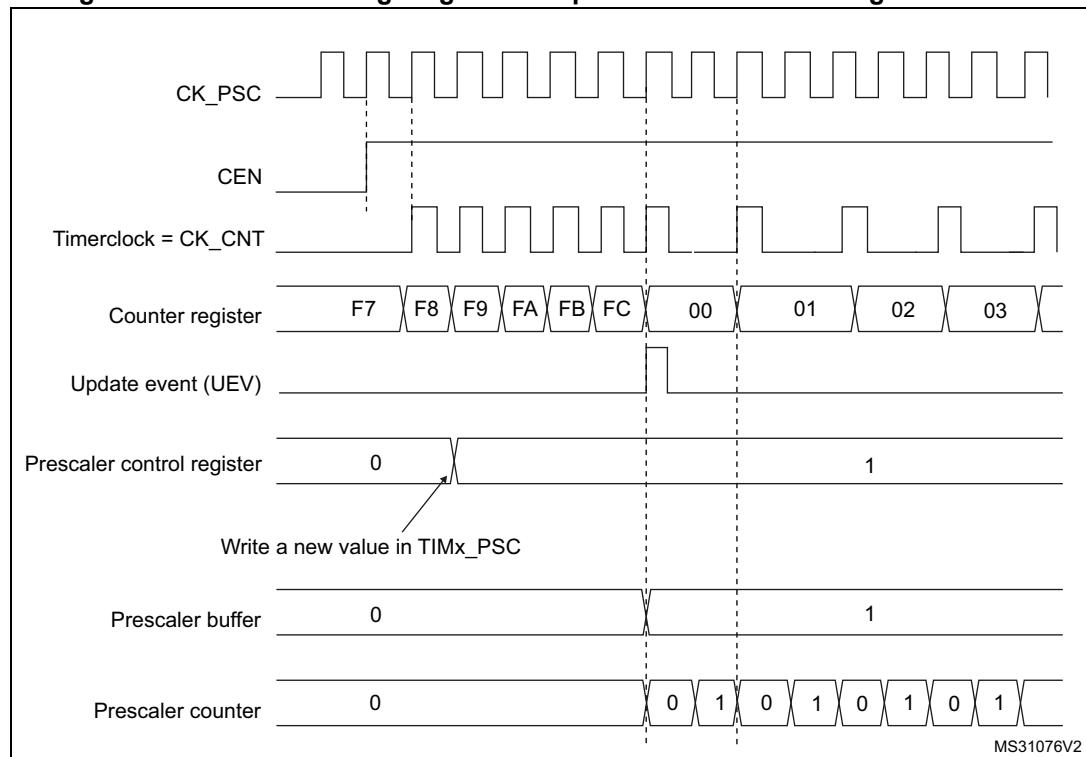
Note that the counter starts counting 1 clock cycle after setting the CEN bit in the TIMx\_CR1 register.

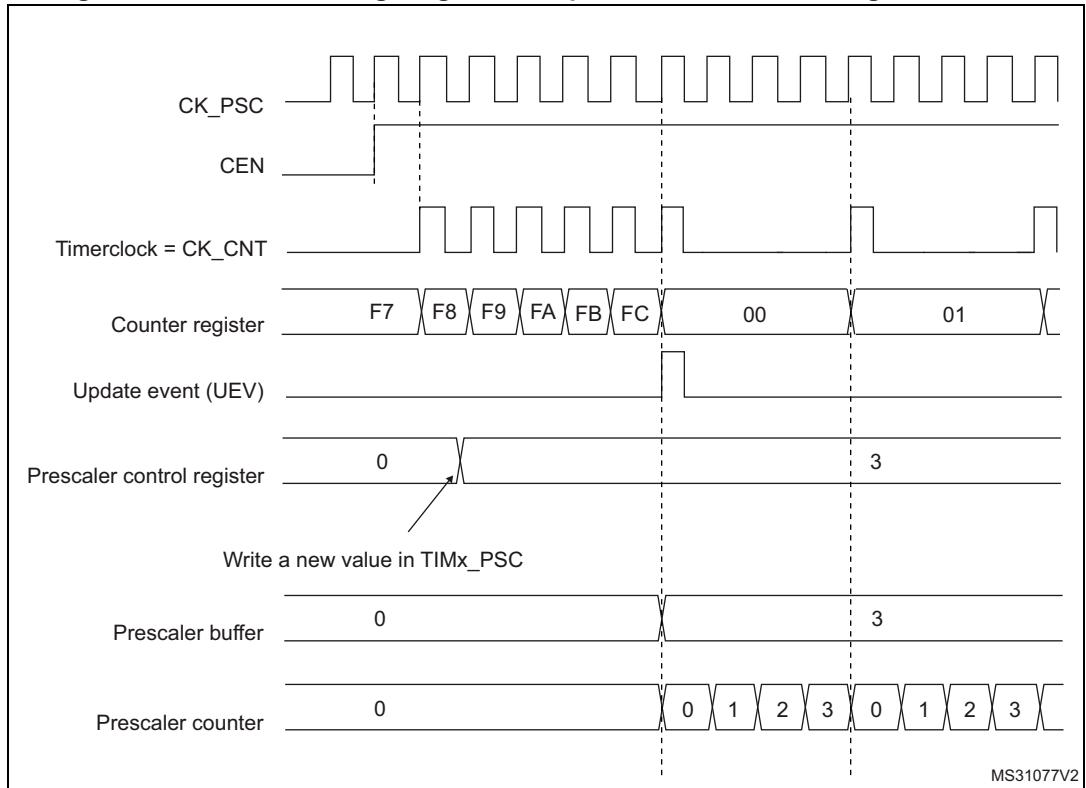
## Prescaler description

The prescaler can divide the counter clock frequency by any factor between 1 and 65536. It is based on a 16-bit counter controlled through a 16-bit register (in the TIMx\_PSC register). It can be changed on the fly as this control register is buffered. The new prescaler ratio is taken into account at the next update event.

[Figure 159](#) and [Figure 160](#) give some examples of the counter behavior when the prescaler ratio is changed on the fly:

**Figure 174.** Counter timing diagram with prescaler division change from 1 to 2



**Figure 175. Counter timing diagram with prescaler division change from 1 to 4**

## 20.4.2 Counter operation

The counter counts from 0 to the auto-reload value (content of the TIMx\_ARR register), then restarts from 0 and generates a counter overflow event.

If the repetition counter is used, the update event (UEV) is generated after upcounting is repeated for the number of times programmed in the repetition counter register (TIMx\_RCR). Else the update event is generated at each counter overflow.

Setting the UG bit in the TIMx\_EGR register (by software or by using the slave mode controller) also generates an update event.

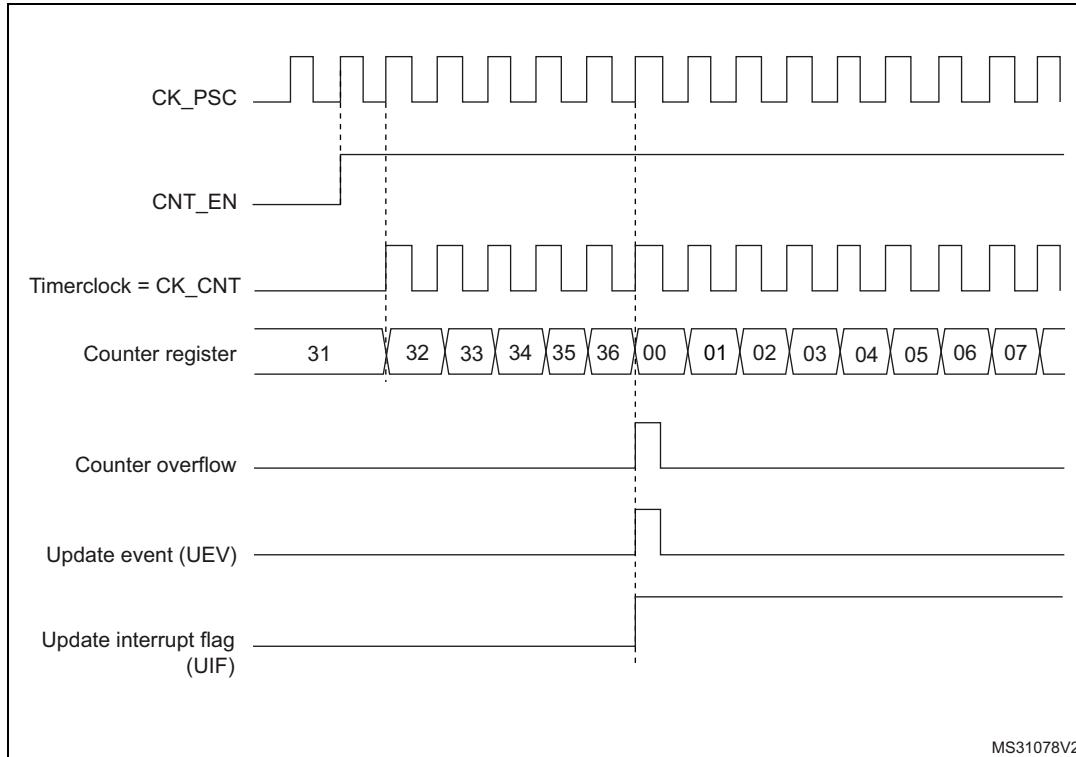
The UEV event can be disabled by software by setting the UDIS bit in the TIMx\_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until the UDIS bit has been written to 0. However, the counter restarts from 0, as well as the counter of the prescaler (but the prescale rate does not change). In addition, if the URS bit (update request selection) in TIMx\_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

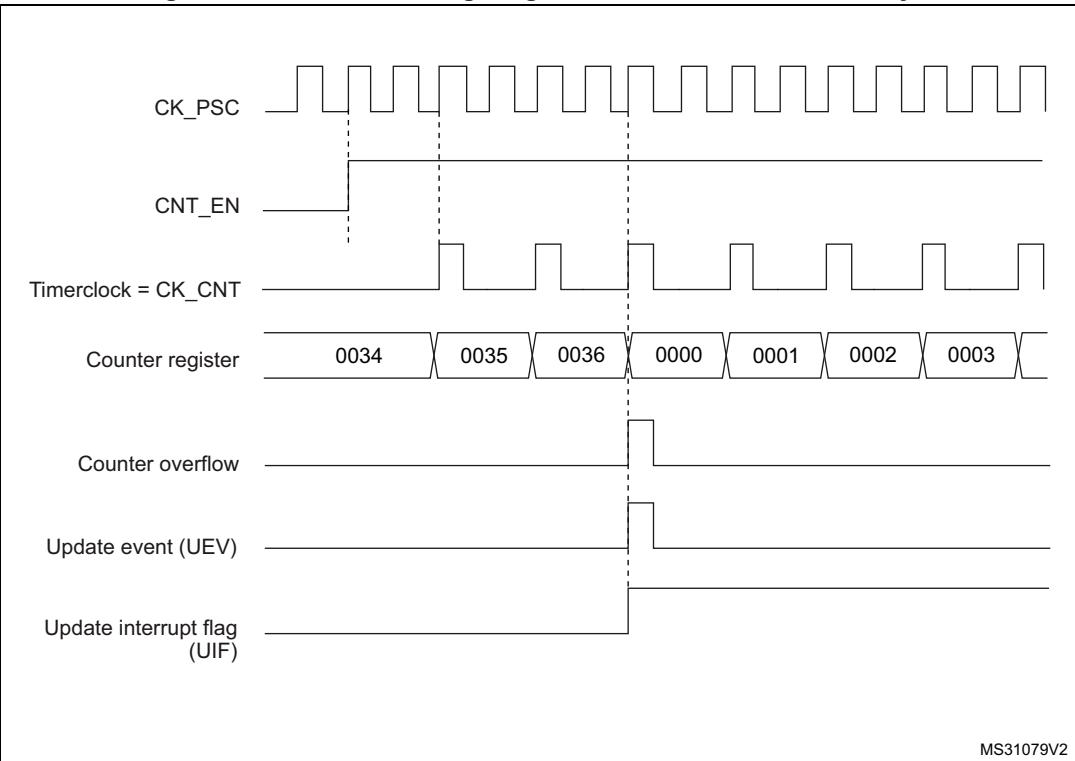
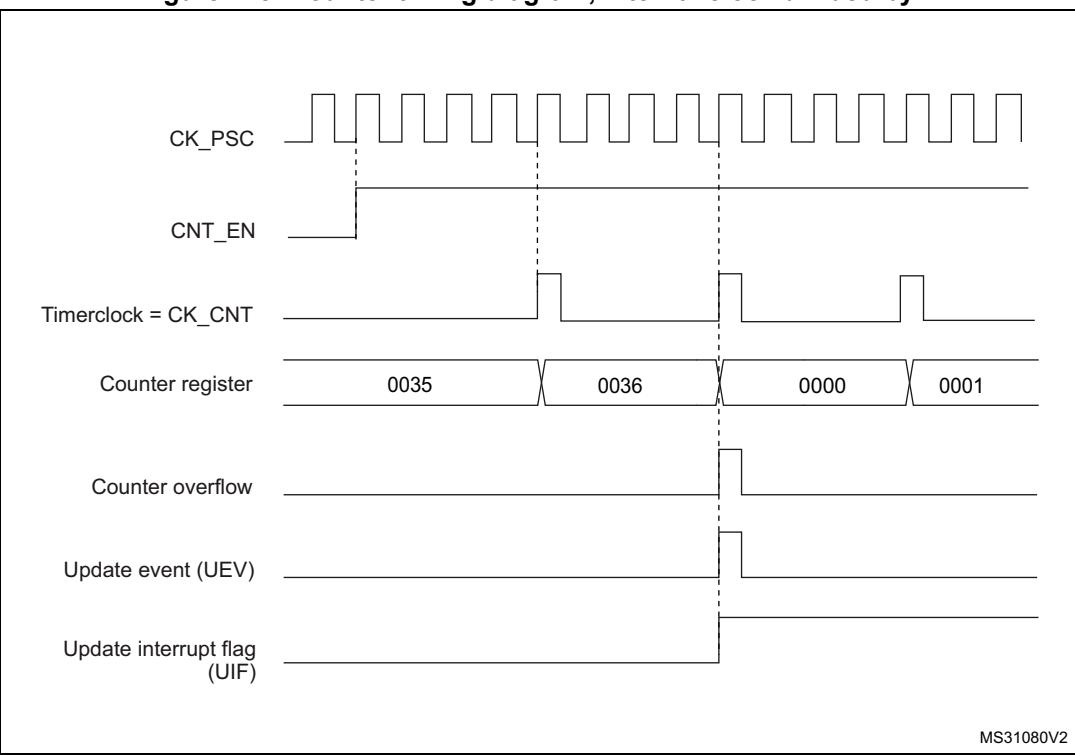
When an update event occurs, all the registers are updated and the update flag (UIF bit in TIMx\_SR register) is set (depending on the URS bit):

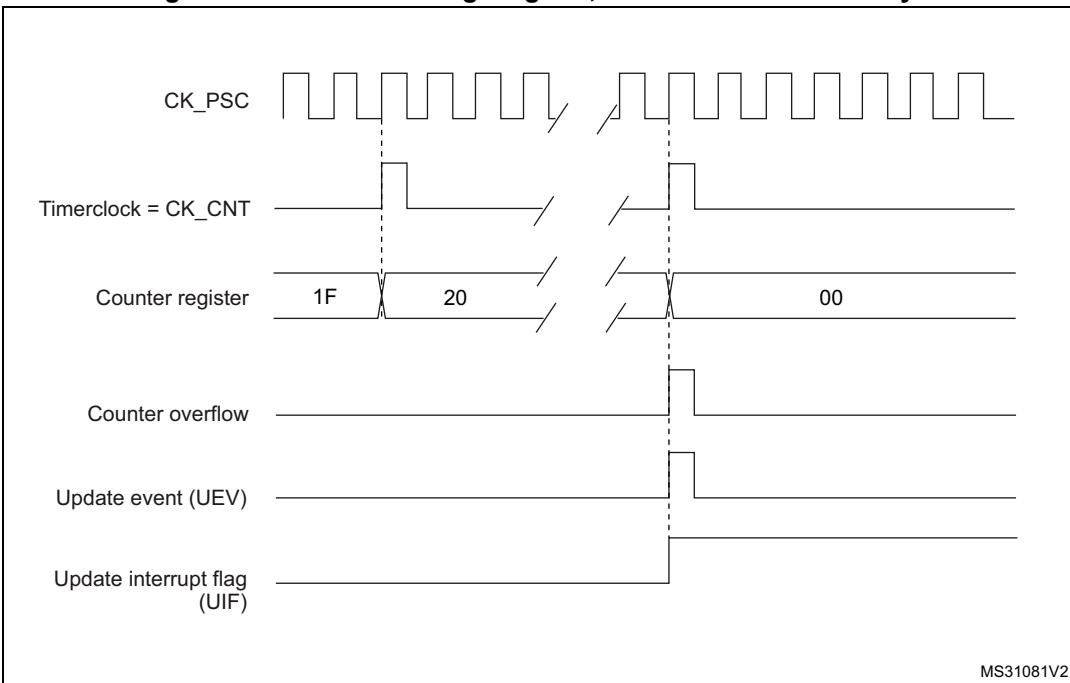
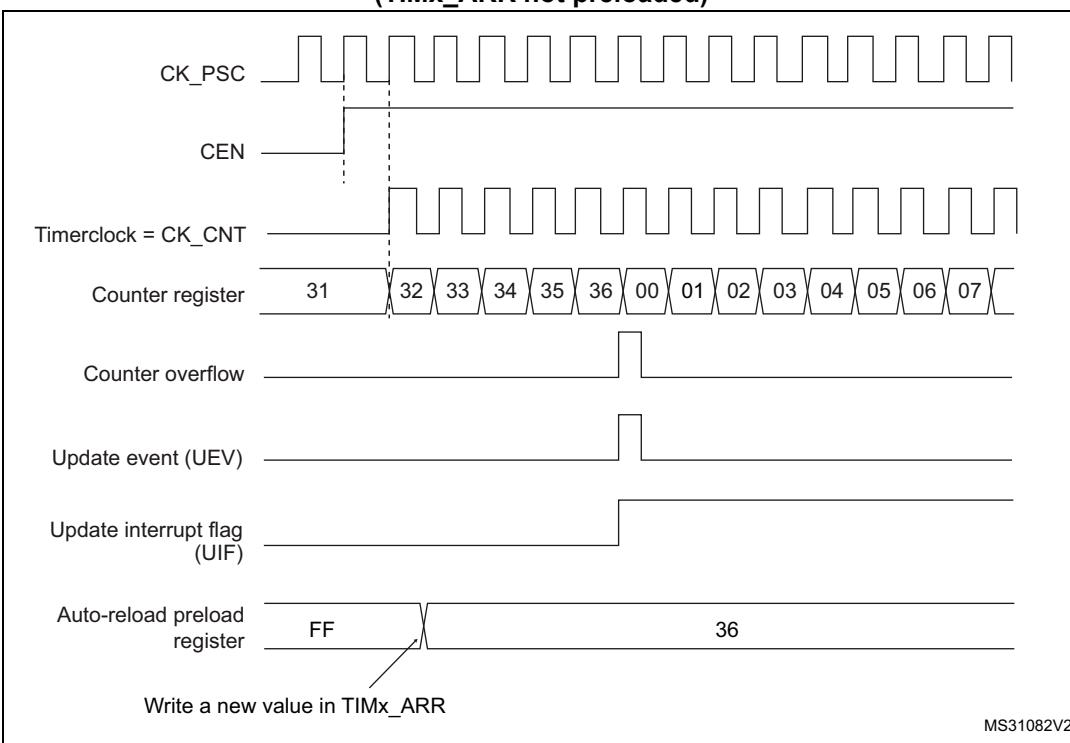
- The repetition counter is reloaded with the content of TIMx\_RCR register,
- The auto-reload shadow register is updated with the preload value (TIMx\_ARR),
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx\_PSC register).

The following figures show some examples of the counter behavior for different clock frequencies when TIMx\_ARR=0x36.

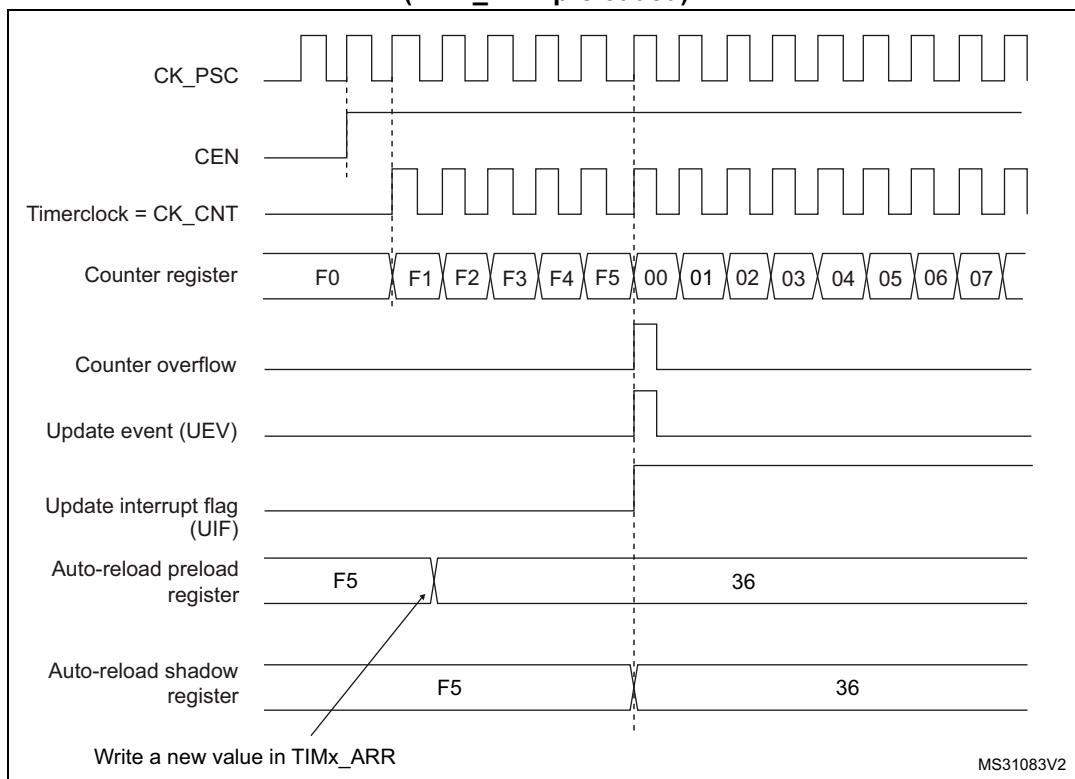
**Figure 176. Counter timing diagram, internal clock divided by 1**



**Figure 177. Counter timing diagram, internal clock divided by 2****Figure 178. Counter timing diagram, internal clock divided by 4**

**Figure 179. Counter timing diagram, internal clock divided by N****Figure 180. Counter timing diagram, update event when ARPE=0  
(TIMx\_ARR not preloaded)**

**Figure 181. Counter timing diagram, update event when ARPE=1  
(TIMx\_ARR preloaded)**



#### 20.4.3 Repetition counter

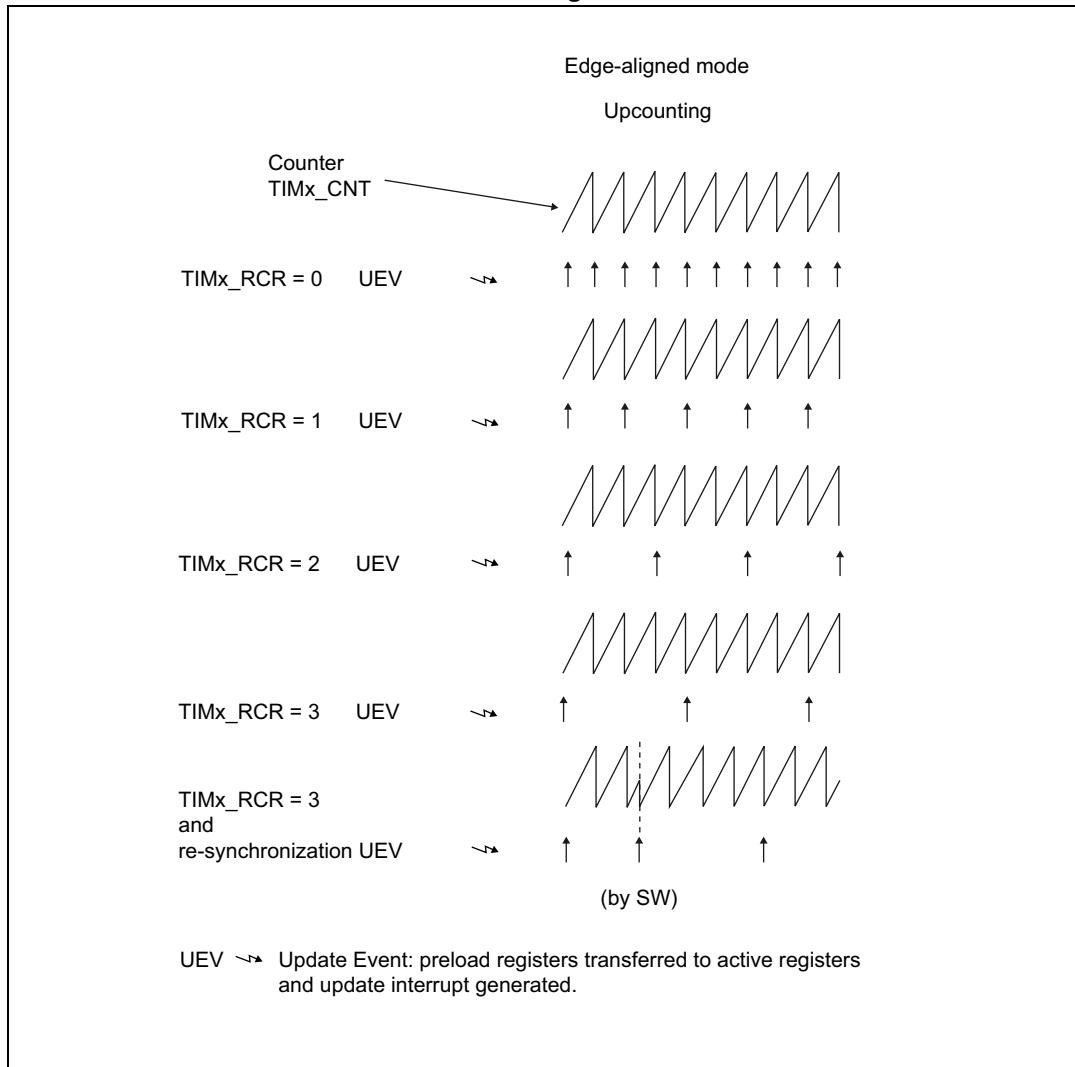
[Section 19.3.1: Time-base unit](#) describes how the update event (UEV) is generated with respect to the counter overflows/underflows. It is actually generated only when the repetition counter has reached zero. This can be useful when generating PWM signals.

This means that data are transferred from the preload registers to the shadow registers (TIMx\_ARR auto-reload register, TIMx\_PSC prescaler register, but also TIMx\_CCRx capture/compare registers in compare mode) every N counter overflows or underflows, where N is the value in the TIMx\_RCR repetition counter register.

The repetition counter is decremented at each counter overflow in upcounting mode.

The repetition counter is an auto-reload type; the repetition rate is maintained as defined by the TIMx\_RCR register value (refer to [Figure 182](#)). When the update event is generated by software (by setting the UG bit in TIMx\_EGR register) or by hardware through the slave mode controller, it occurs immediately whatever the value of the repetition counter is and the repetition counter is reloaded with the content of the TIMx\_RCR register.

**Figure 182. Update rate examples depending on mode and TIMx\_RCR register settings**



#### 20.4.4 Clock sources

The counter clock can be provided by the following clock sources:

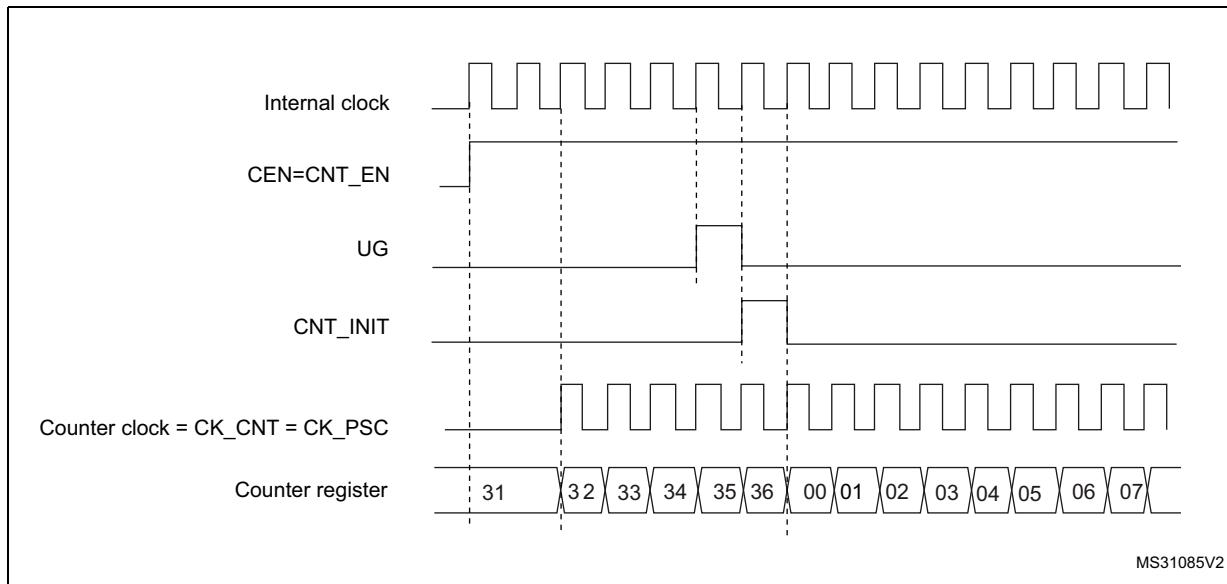
- Internal clock (CK\_INT)
- External clock mode1: external input pin (only for TIM15)
- Internal trigger inputs (ITRx) (only for TIM15): using one timer as the prescaler for another timer, for example, TIM1 can be configured to act as a prescaler for TIM15. Refer to [Using one timer as prescaler for another](#) for more details.

##### Internal clock source (CK\_INT)

For TIM15 if the slave mode controller is disabled (SMS=000), then the CEN and UG bits (in the TIMx\_EGR register) are actual control bits and can be changed only by software (except UG which remains cleared automatically). As soon as the CEN bit is written to 1, the prescaler is clocked by the internal clock CK\_INT.

*Figure 19.3.4* shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

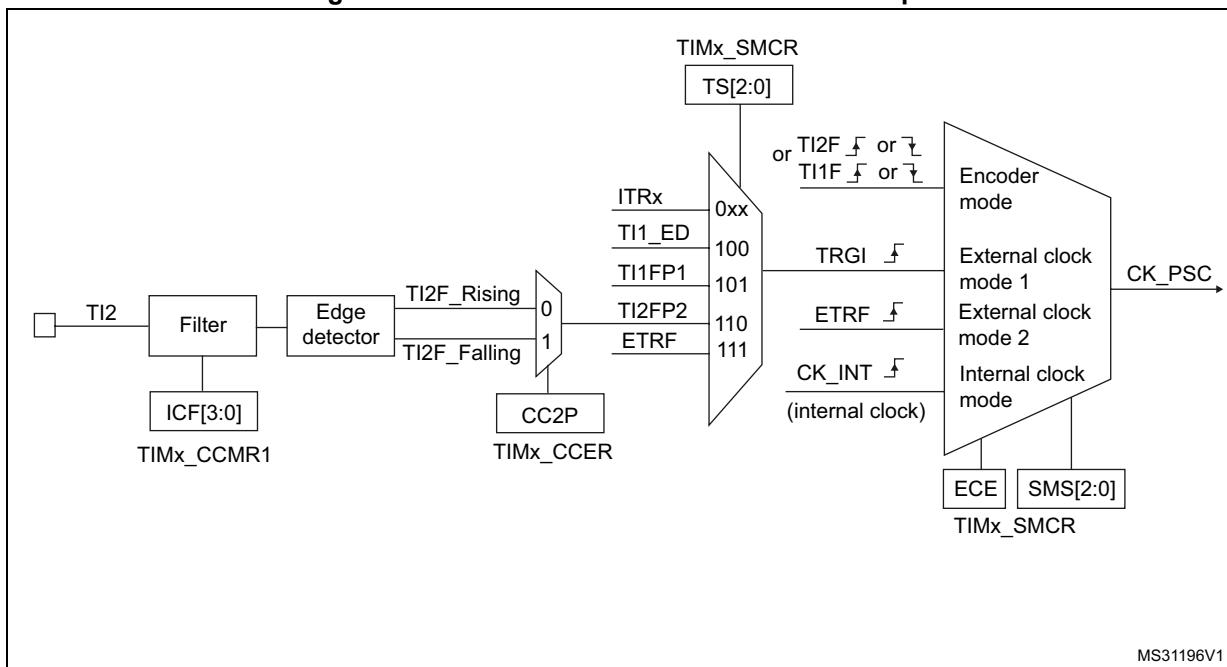
**Figure 183. Control circuit in normal mode, internal clock divided by 1**



### External clock source mode 1

This mode is selected when SMS=111 in the TIMx\_SMCR register. The counter can count at each rising or falling edge on a selected input.

**Figure 184. TI2 external clock connection example**



For example, to configure the upcounter to count in response to a rising edge on the TI2 input, use the following procedure:

1. Configure channel 2 to detect rising edges on the TI2 input by writing CC2S = '01' in the TIMx\_CCMR1 register.
2. Configure the input filter duration by writing the IC2F[3:0] bits in the TIMx\_CCMR1 register (if no filter is needed, keep IC2F=0000).
3. Select rising edge polarity by writing CC2P=0 in the TIMx\_CCER register.
4. Configure the timer in external clock mode 1 by writing SMS=111 in the TIMx\_SMCR register.
5. Select TI2 as the trigger input source by writing TS=110 in the TIMx\_SMCR register.
6. Enable the counter by writing CEN=1 in the TIMx\_CR1 register.

*Note:*

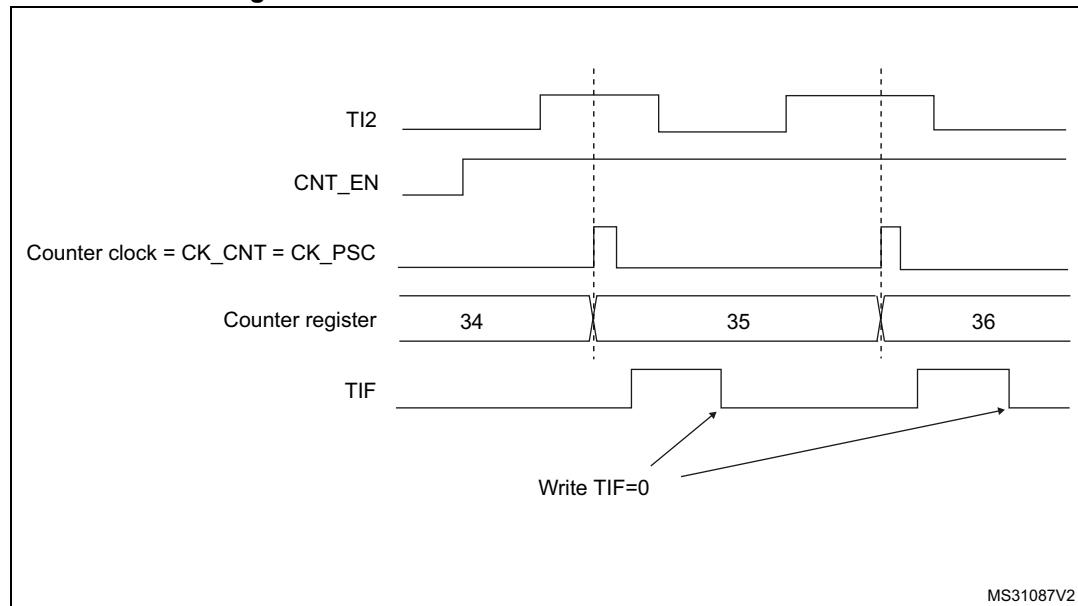
*The capture prescaler is not used for triggering, so it does not need to be configured.*

For code example refer to the Appendix section [A.9.1: Upcounter on TI2 rising edge code example](#).

When a rising edge occurs on TI2, the counter counts once and the TIF flag is set.

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

**Figure 185. Control circuit in external clock mode 1**



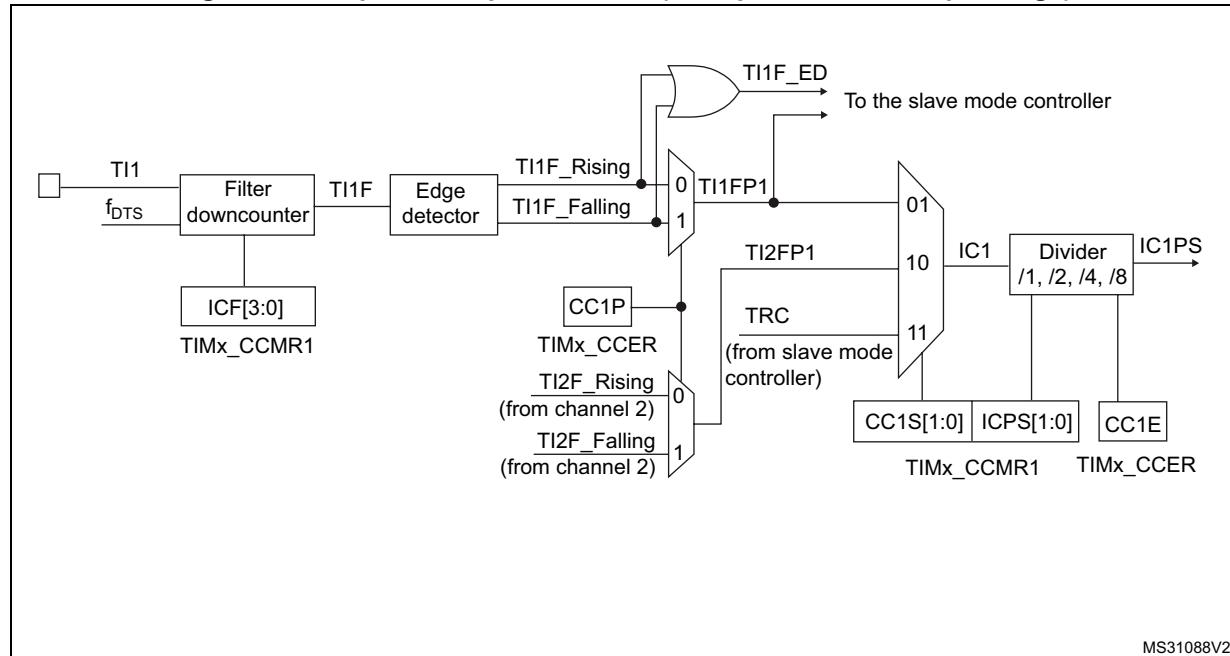
#### 20.4.5 Capture/compare channels

Each Capture/Compare channel is built around a capture/compare register (including a shadow register), a input stage for capture (with digital filter, multiplexing and prescaler) and an output stage (with comparator and output control).

[Figure 167](#) to [Figure 189](#) give an overview of one Capture/Compare channel.

The input stage samples the corresponding TIx input to generate a filtered signal TIxF. Then, an edge detector with polarity selection generates a signal (TIxFPx) which can be used as trigger input by the slave mode controller or as the capture command. It is prescaled before the capture register (ICxPS).

Figure 186. Capture/compare channel (example: channel 1 input stage)



The output stage generates an intermediate waveform which is then used for reference: OCxRef (active high). The polarity acts at the end of the chain.

Figure 187. Capture/compare channel 1 main circuit

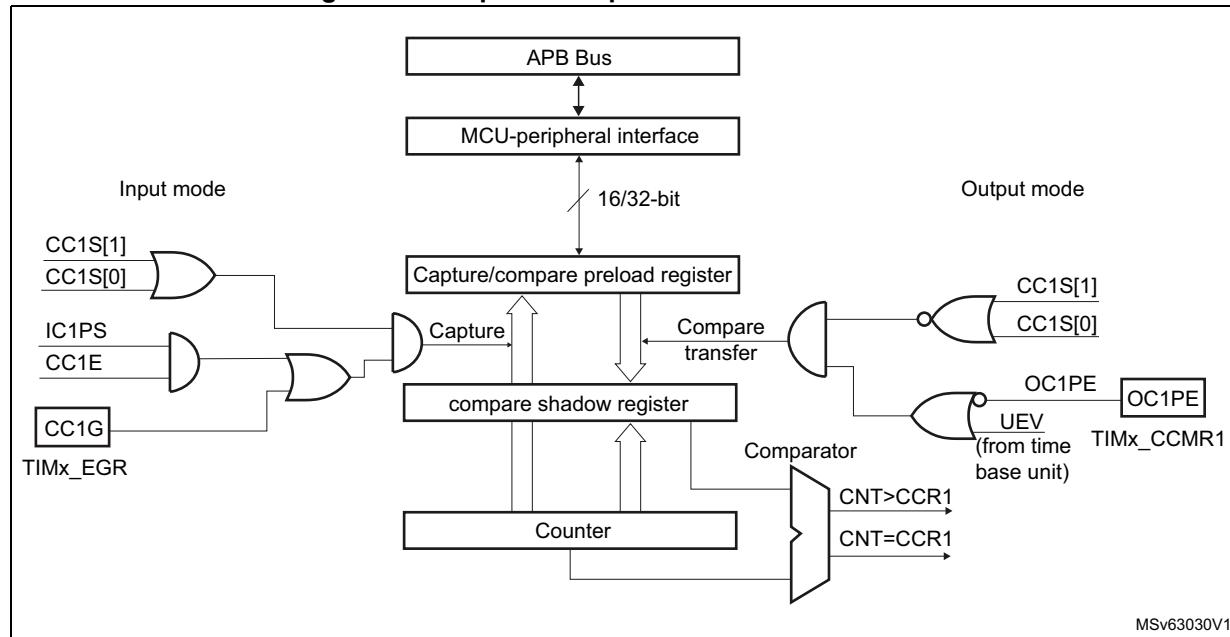


Figure 188. Output stage of capture/compare channel (channel 1)

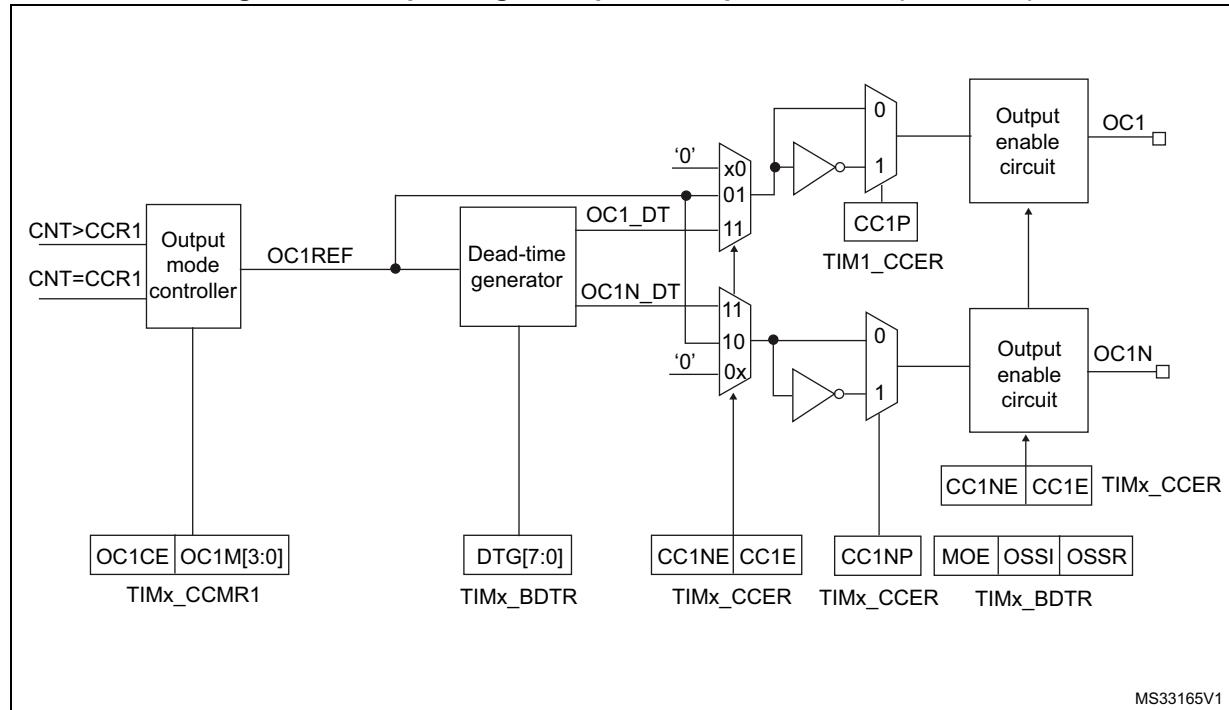
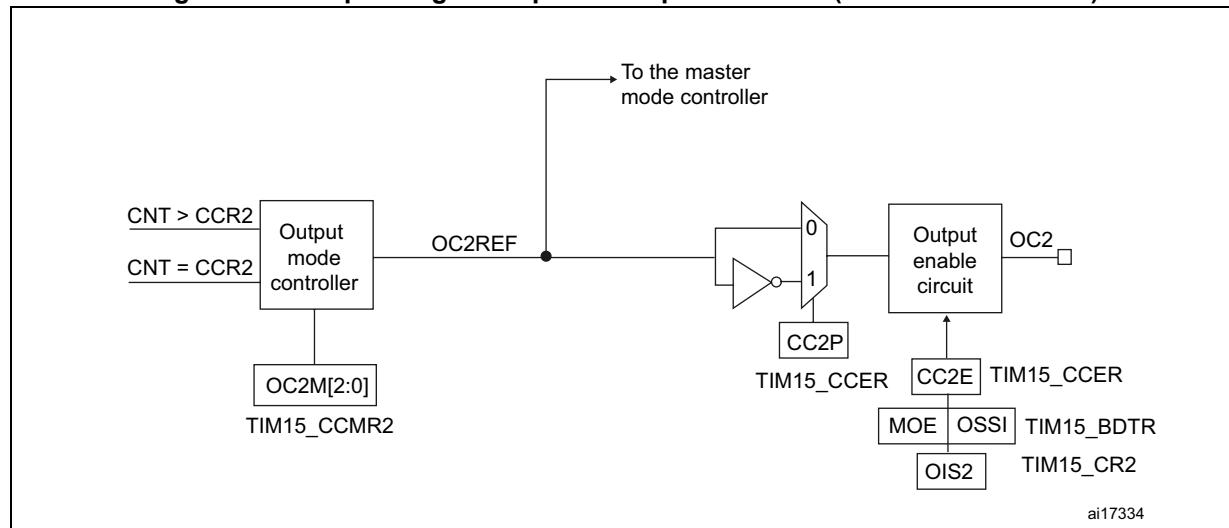


Figure 189. Output stage of capture/compare channel (channel 2 for TIM15)



The capture/compare block is made of one preload register and one shadow register. Write and read always access the preload register.

In capture mode, captures are actually done in the shadow register, which is copied into the preload register.

In compare mode, the content of the preload register is copied into the shadow register which is compared to the counter.

## 20.4.6 Input capture mode

In Input capture mode, the Capture/Compare Registers (TIMx\_CCRx) are used to latch the value of the counter after a transition detected by the corresponding ICx signal. When a capture occurs, the corresponding CCxIF flag (TIMx\_SR register) is set and an interrupt or a DMA request can be sent if they are enabled. If a capture occurs while the CCxIF flag was already high, then the over-capture flag CCxOF (TIMx\_SR register) is set. CCxIF can be cleared by software by writing it to '0' or by reading the captured data stored in the TIMx\_CCRx register. CCxOF is cleared when it is written with 0.

The following example shows how to capture the counter value in TIMx\_CCR1 when TI1 input rises. To do this, use the following procedure:

- Select the active input: TIMx\_CCR1 must be linked to the TI1 input, so write the CC1S bits to 01 in the TIMx\_CCMR1 register. As soon as CC1S becomes different from 00, the channel is configured in input and the TIMx\_CCR1 register becomes read-only.
- Program the appropriate input filter duration in relation with the signal connected to the timer (when the input is one of the TIx (ICxF bits in the TIMx\_CCMRx register). Let's imagine that, when toggling, the input signal is not stable during at most 5 internal clock cycles. We must program a filter duration longer than these 5 clock cycles. We can validate a transition on TI1 when 8 consecutive samples with the new level have been detected (sampled at  $f_{DTS}$  frequency). Then write IC1F bits to 0011 in the TIMx\_CCMR1 register.
- Select the edge of the active transition on the TI1 channel by writing CC1P bit to 0 in the TIMx\_CCER register (rising edge in this case).
- Program the input prescaler. In our example, we wish the capture to be performed at each valid transition, so the prescaler is disabled (write IC1PS bits to '00' in the TIMx\_CCMR1 register).
- Enable capture from the counter into the capture register by setting the CC1E bit in the TIMx\_CCER register.
- If needed, enable the related interrupt request by setting the CC1IE bit in the TIMx\_DIER register, and/or the DMA request by setting the CC1DE bit in the TIMx\_DIER register.

For code example refer to the Appendix section [A.9.3: Input capture configuration code example](#).

When an input capture occurs:

- The TIMx\_CCR1 register gets the value of the counter on the active transition.
- CC1IF flag is set (interrupt flag). CC1OF is also set if at least two consecutive captures occurred whereas the flag was not cleared.
- An interrupt is generated depending on the CC1IE bit.
- A DMA request is generated depending on the CC1DE bit.

For code example refer to the Appendix section [A.9.4: Input capture data management code example](#).

In order to handle the overcapture, it is recommended to read the data before the overcapture flag. This is to avoid missing an overcapture which could happen after reading the flag and before reading the data.

**Note:** *IC interrupt and/or DMA requests can be generated by software by setting the corresponding CCxG bit in the TIMx\_EGR register.*

### 20.4.7 PWM input mode (only for TIM15)

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

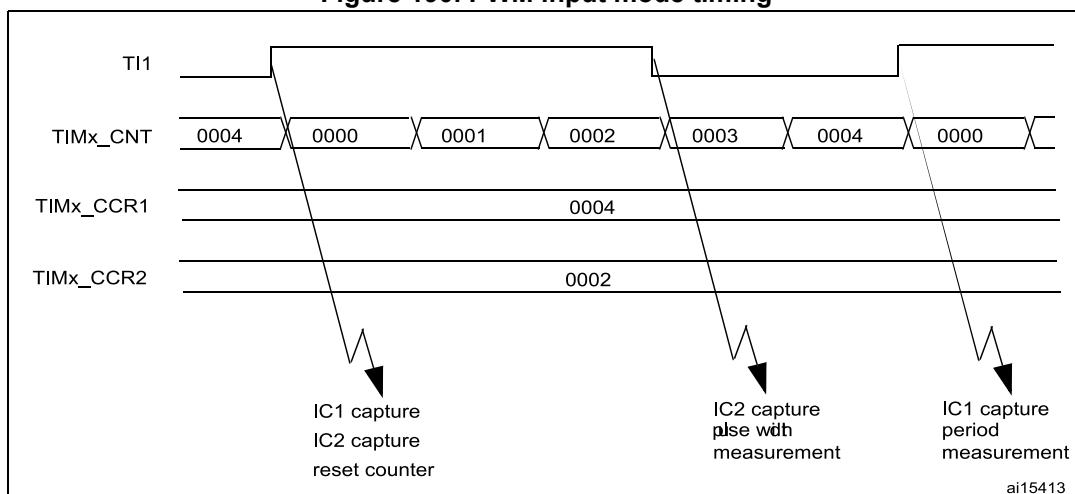
- Two ICx signals are mapped on the same TIx input.
- These 2 ICx signals are active on edges with opposite polarity.
- One of the two TIxFP signals is selected as trigger input and the slave mode controller is configured in reset mode.

For example, one can measure the period (in TIMx\_CCR1 register) and the duty cycle (in TIMx\_CCR2 register) of the PWM applied on TI1 using the following procedure (depending on CK\_INT frequency and prescaler value):

- Select the active input for TIMx\_CCR1: write the CC1S bits to 01 in the TIMx\_CCMR1 register (TI1 selected).
- Select the active polarity for TI1FP1 (used both for capture in TIMx\_CCR1 and counter clear): write the CC1P bit to '0' (active on rising edge).
- Select the active input for TIMx\_CCR2: write the CC2S bits to 10 in the TIMx\_CCMR1 register (TI1 selected).
- Select the active polarity for TI1FP2 (used for capture in TIMx\_CCR2): write the CC2P bit to '1' and the CC2NP bit to '0' (active on falling edge).
- Select the valid trigger input: write the TS bits to 101 in the TIMx\_SMCR register (TI1FP1 selected).
- Configure the slave mode controller in reset mode: write the SMS bits to 100 in the TIMx\_SMCR register.
- Enable the captures: write the CC1E and CC2E bits to '1' in the TIMx\_CCER register.

For code example refer to the Appendix section [A.9.5: PWM input configuration code example](#).

**Figure 190. PWM input mode timing**



1. The PWM input mode can be used only with the TIMx\_CH1/TIMx\_CH2 signals due to the fact that only TI1FP1 and TI2FP2 are connected to the slave mode controller.

#### 20.4.8 Forced output mode

In output mode (CCxS bits = 00 in the TIMx\_CCMRx register), each output compare signal (OCxREF and then OCx/OCxN) can be forced to active or inactive level directly by software, independently of any comparison between the output compare register and the counter.

To force an output compare signal (OCXREF/OCx) to its active level, one just needs to write 101 in the OCxM bits in the corresponding TIMx\_CCMRx register. Thus OCXREF is forced high (OCxREF is always active high) and OCx get opposite value to CCxP polarity bit.

For example: CCxP=0 (OCx active high) => OCx is forced to high level.

The OCxREF signal can be forced low by writing the OCxM bits to 100 in the TIMx\_CCMRx register.

Anyway, the comparison between the TIMx\_CCRx shadow register and the counter is still performed and allows the flag to be set. Interrupt and DMA requests can be sent accordingly. This is described in the output compare mode section below.

#### 20.4.9 Output compare mode

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

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

- Assigns the corresponding output pin to a programmable value defined by the output compare mode (OCxM bits in the TIMx\_CCMRx register) and the output polarity (CCxP bit in the TIMx\_CCER register). The output pin can keep its level (OCXM=000), be set active (OCXM=001), be set inactive (OCXM=010) or can toggle (OCXM=011) on match.
- Sets a flag in the interrupt status register (CCxIF bit in the TIMx\_SR register).
- Generates an interrupt if the corresponding interrupt mask is set (CCXIE bit in the TIMx\_DIER register).
- Sends a DMA request if the corresponding enable bit is set (CCxDE bit in the TIMx\_DIER register, CCDS bit in the TIMx\_CR2 register for the DMA request selection).

The TIMx\_CCRx registers can be programmed with or without preload registers using the OCxPE bit in the TIMx\_CCMRx register.

In output compare mode, the update event UEV has no effect on OCxREF and OCx output. The timing resolution is one count of the counter. Output compare mode can also be used to output a single pulse (in One-pulse mode).

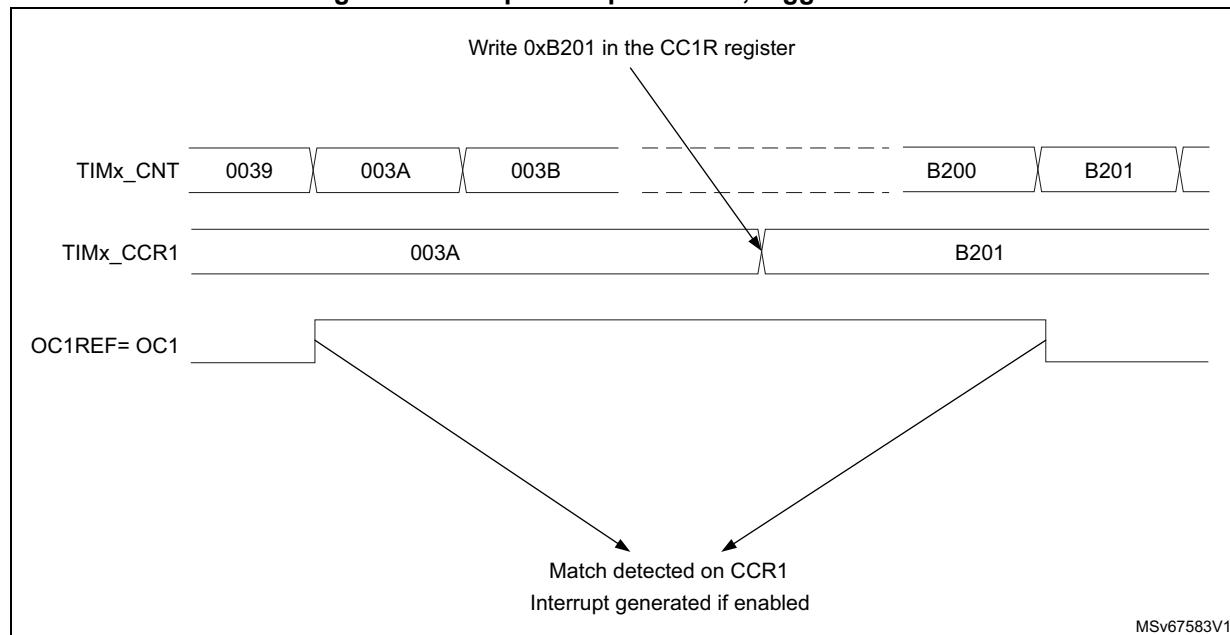
**Procedure:**

1. Select the counter clock (internal, external, prescaler).
2. Write the desired data in the TIMx\_ARR and TIMx\_CCRx registers.
3. Set the CCxIE bit if an interrupt request is to be generated.
4. Select the output mode. For example:
  - Write OCxM = 011 to toggle OCx output pin when CNT matches CCRx
  - Write OCxPE = 0 to disable preload register
  - Write CCxP = 0 to select active high polarity
  - Write CCxE = 1 to enable the output
5. Enable the counter by setting the CEN bit in the TIMx\_CR1 register.

For code example refer to the Appendix section [A.9.2: Up counter on each 2 ETR rising edges code example](#).

The TIMx\_CCRx register can be updated at any time by software to control the output waveform, provided that the preload register is not enabled (OCxPE='0', else TIMx\_CCRx shadow register is updated only at the next update event UEV). An example is given in [Figure 170](#).

**Figure 191. Output compare mode, toggle on OC1**



#### 20.4.10 PWM mode

Pulse Width Modulation mode allows to generate a signal with a frequency determined by the value of the TIMx\_ARR register and a duty cycle determined by the value of the TIMx\_CCRx register.

The PWM mode can be selected independently on each channel (one PWM per OCx output) by writing '110' (PWM mode 1) or '111' (PWM mode 2) in the OCxM bits in the TIMx\_CCMRx register. The corresponding preload register must be enabled by setting the OCxPE bit in the TIMx\_CCMRx register, and eventually the auto-reload preload register by setting the ARPE bit in the TIMx\_CR1 register.

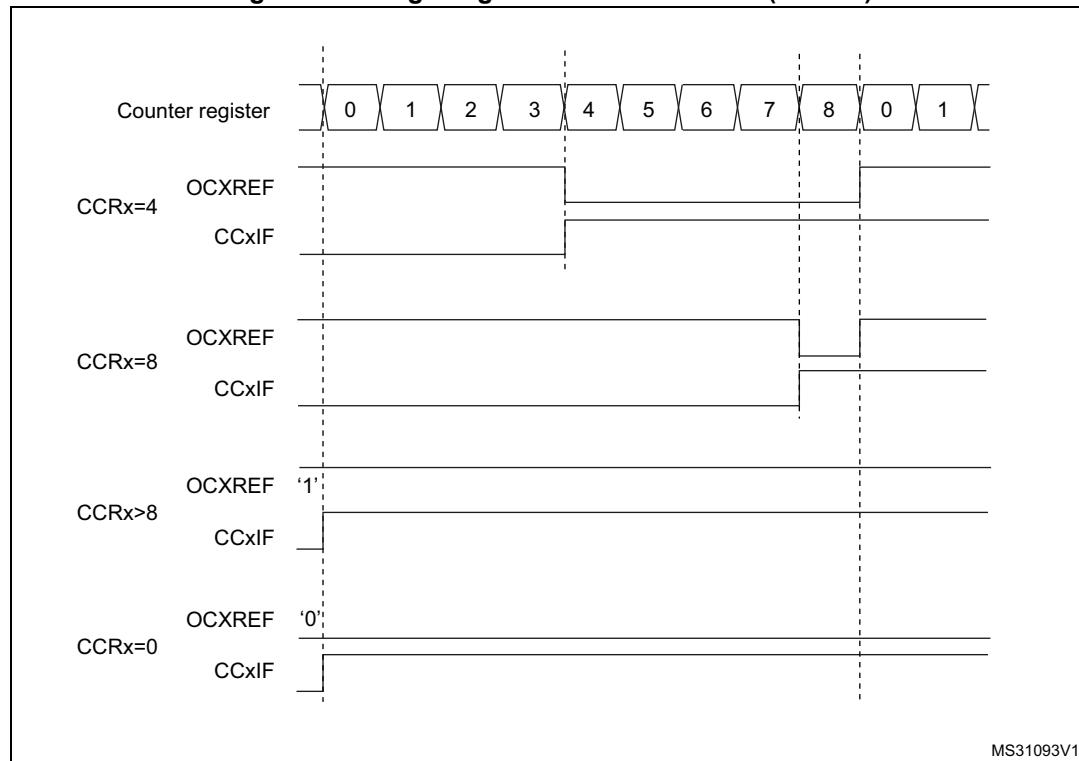
As the preload registers are transferred to the shadow registers only when an update event occurs, before starting the counter, all registers must be initialized by setting the UG bit in the TIMx\_EGR register.

OCx polarity is software programmable using the CCxP bit in the TIMx\_CCER register. It can be programmed as active high or active low. OCx output is enabled by a combination of the CCxE, CCxNE, MOE, OSS1 and OSSR bits (TIMx\_CCER and TIMx\_BDTR registers). Refer to the TIMx\_CCER register description for more details.

In PWM mode (1 or 2), TIMx\_CNT and TIMx\_CCRx are always compared to determine whether  $\text{TIMx\_CCR}_x \leq \text{TIMx\_CNT}$ .

In the following example, we consider PWM mode 1. The reference PWM signal OCxREF is high as long as  $\text{TIMx\_CNT} < \text{TIMx\_CCR}_x$  else it becomes low. If the compare value in TIMx\_CCRx is greater than the auto-reload value (in TIMx\_ARR) then OCxREF is held at '1'. If the compare value is 0 then OCxRef is held at '0'. [Figure 171](#) shows some edge-aligned PWM waveforms in an example where TIMx\_ARR=8.

**Figure 192. Edge-aligned PWM waveforms (ARR=8)**



#### 20.4.11 Complementary outputs and dead-time insertion

The TIM15/16/17 general-purpose timers can output one complementary signal and manage the switching-off and switching-on of the outputs.

This time is generally known as dead-time and it has to be adjusted depending on the devices that are connected to the outputs and their characteristics (intrinsic delays of level-shifters, delays due to power switches...)

The polarity of the outputs (main output OC<sub>x</sub> or complementary OC<sub>xN</sub>) can be selected independently for each output. This is done by writing to the CC<sub>xP</sub> and CC<sub>xNP</sub> bits in the TIM<sub>x</sub>\_CCER register.

The complementary signals OC<sub>x</sub> and OC<sub>xN</sub> are activated by a combination of several control bits: the CC<sub>xE</sub> and CC<sub>xNE</sub> bits in the TIM<sub>x</sub>\_CCER register and the MOE, OIS<sub>x</sub>, OIS<sub>xN</sub>, OSS<sub>I</sub> and OSS<sub>R</sub> bits in the TIM<sub>x</sub>\_BDTR and TIM<sub>x</sub>\_CR2 registers. Refer to [Table 72: Output control bits for complementary OC<sub>x</sub> and OC<sub>xN</sub> channels with break feature on page 535](#) for more details. In particular, the dead-time is activated when switching to the IDLE state (MOE falling down to 0).

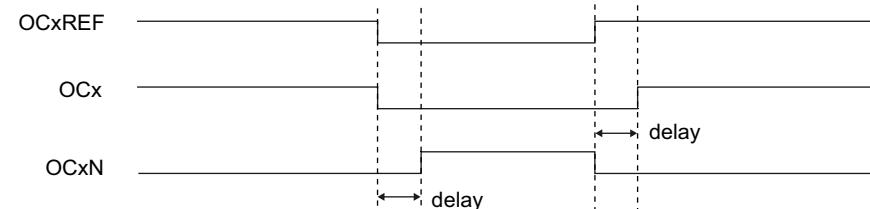
Dead-time insertion is enabled by setting both CC<sub>xE</sub> and CC<sub>xNE</sub> bits, and the MOE bit if the break circuit is present. There is one 10-bit dead-time generator for each channel. From a reference waveform OC<sub>xREF</sub>, it generates 2 outputs OC<sub>x</sub> and OC<sub>xN</sub>. If OC<sub>x</sub> and OC<sub>xN</sub> are active high:

- The OC<sub>x</sub> 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<sub>xN</sub> output signal is the opposite of the reference signal except for the rising edge, which is delayed relative to the reference falling edge.

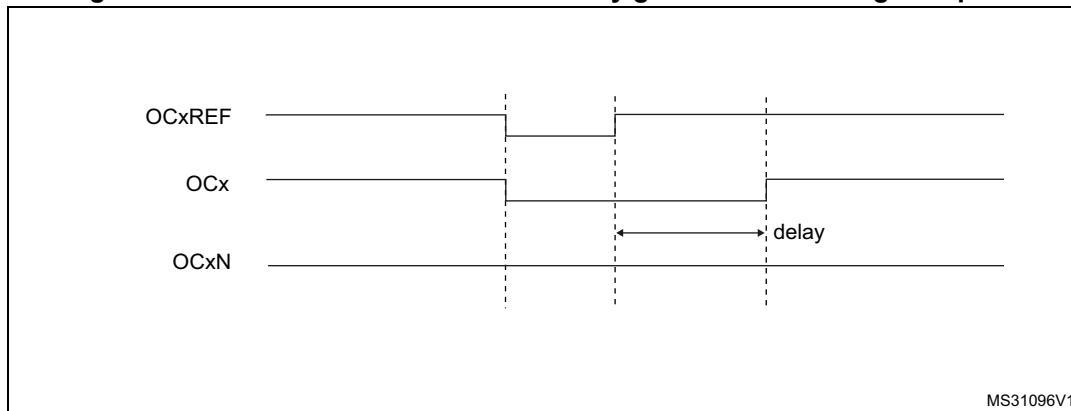
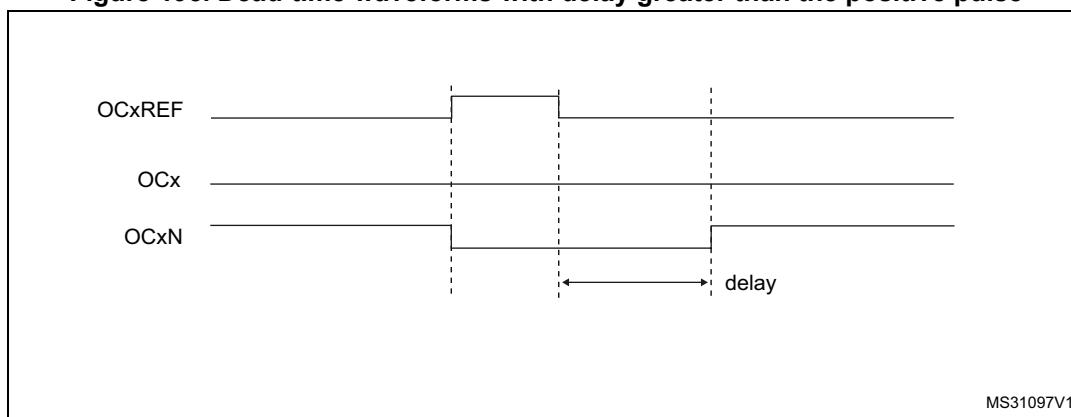
If the delay is greater than the width of the active output (OC<sub>x</sub> or OC<sub>xN</sub>) then the corresponding pulse is not generated.

The following figures show the relationships between the output signals of the dead-time generator and the reference signal OC<sub>xREF</sub>. (we suppose CC<sub>xP</sub>=0, CC<sub>xNP</sub>=0, MOE=1, CC<sub>xE</sub>=1 and CC<sub>xNE</sub>=1 in these examples)

**Figure 193. Complementary output with dead-time insertion**



MS31095V1

**Figure 194. Dead-time waveforms with delay greater than the negative pulse****Figure 195. Dead-time waveforms with delay greater than the positive pulse**

The dead-time delay is the same for each of the channels and is programmable with the DTG bits in the TIMx\_BDTR register. Refer to [Section 20.5.16: TIM15 break and dead-time register \(TIM15\\_BDTR\) on page 538](#) for delay calculation.

#### Re-directing OCxREF to OCx or OCxN

In output mode (forced, output compare or PWM), OCxREF can be re-directed to the OCx output or to OCxN output by configuring the CCxE and CCxNE bits in the TIMx\_CCER register.

This allows to send a specific waveform (such as PWM or static active level) on one output while the complementary remains at its inactive level. Other alternative possibilities are to have both outputs at inactive level or both outputs active and complementary with dead-time.

**Note:** When only OCxN is enabled (CCxE=0, CCxNE=1), it is not complemented and becomes active as soon as OCxREF is high. For example, if CCxNP=0 then OCxN=OCxRef. On the other hand, when both OCx and OCxN are enabled (CCxE=CCxNE=1) OCx becomes active when OCxREF is high whereas OCxN is complemented and becomes active when OCxREF is low.

### 20.4.12 Using the break function

When using the break function, the output enable signals and inactive levels are modified according to additional control bits (MOE, OSS1 and OSSR bits in the TIMx\_BDTR register, OISx and OISxN bits in the TIMx\_CR2 register). In any case, the OCx and OCxN outputs cannot be set both to active level at a given time. Refer to [Table 72: Output control bits for complementary OC<sub>x</sub> and OC<sub>xN</sub> channels with break feature on page 535](#) for more details.

The source for break (BRK) channel can be an external source connected to the BKIN pin or one of the following internal sources:

- the core LOCKUP output
- the PVD output
- the SRAM parity error signal
- a clock failure event generated by the CSS detector

When exiting from reset, the break circuit is disabled and the MOE bit is low. The break function can be enabled by setting the BKE bit in the TIMx\_BDTR register. The break input polarity can be selected by configuring the BKP bit in the same register. BKE and BKP can be modified at the same time. When the BKE and BKP bits are written, a delay of 1 APB clock cycle is applied before the writing is effective. Consequently, it is necessary to wait 1 APB clock period to correctly read back the bit after the write operation.

Because MOE falling edge can be asynchronous, a resynchronization circuit has been inserted between the actual signal (acting on the outputs) and the synchronous control bit (accessed in the TIMx\_BDTR register). It results in some delays between the asynchronous and the synchronous signals. In particular, if MOE is set to 1 whereas it was low, a delay must be inserted (dummy instruction) before reading it correctly. This is because the write acts on the asynchronous signal whereas the read reflects the synchronous signal.

When a break occurs (selected level on the break input):

- The MOE bit is cleared asynchronously, putting the outputs in inactive state, idle state or in reset state (selected by the OSS1 bit). This feature functions even if the MCU oscillator is off.
- Each output channel is driven with the level programmed in the OISx bit in the TIMx\_CR2 register as soon as MOE=0. If OSS1=0 then the timer releases the enable output else the enable output remains high.
- When complementary outputs are used:
  - The outputs are first put in reset state inactive state (depending on the polarity). This is done asynchronously so that it works even if no clock is provided to the timer.
  - If the timer clock is still present, then the dead-time generator is reactivated in order to drive the outputs with the level programmed in the OISx and OISxN bits after a dead-time. Even in this case, OCx and OCxN cannot be driven to their active level together. Note that because of the resynchronization on MOE, the dead-time duration is a bit longer than usual (around 2 ck\_tim clock cycles).
  - If OSS1=0 then the timer releases the enable outputs else the enable outputs remain or become high as soon as one of the CCxE or CCxNE bits is high.
- The break status flag (BIF bit in the TIMx\_SR register) is set. An interrupt can be generated if the BIE bit in the TIMx\_DIER register is set.
- If the AOE bit in the TIMx\_BDTR register is set, the MOE bit is automatically set again at the next update event UEV. This can be used to perform a regulation, for instance.

Else, MOE remains low until it is written with 1 again. In this case, it can be used for security and the break input can be connected to an alarm from power drivers, thermal sensors or any security components.

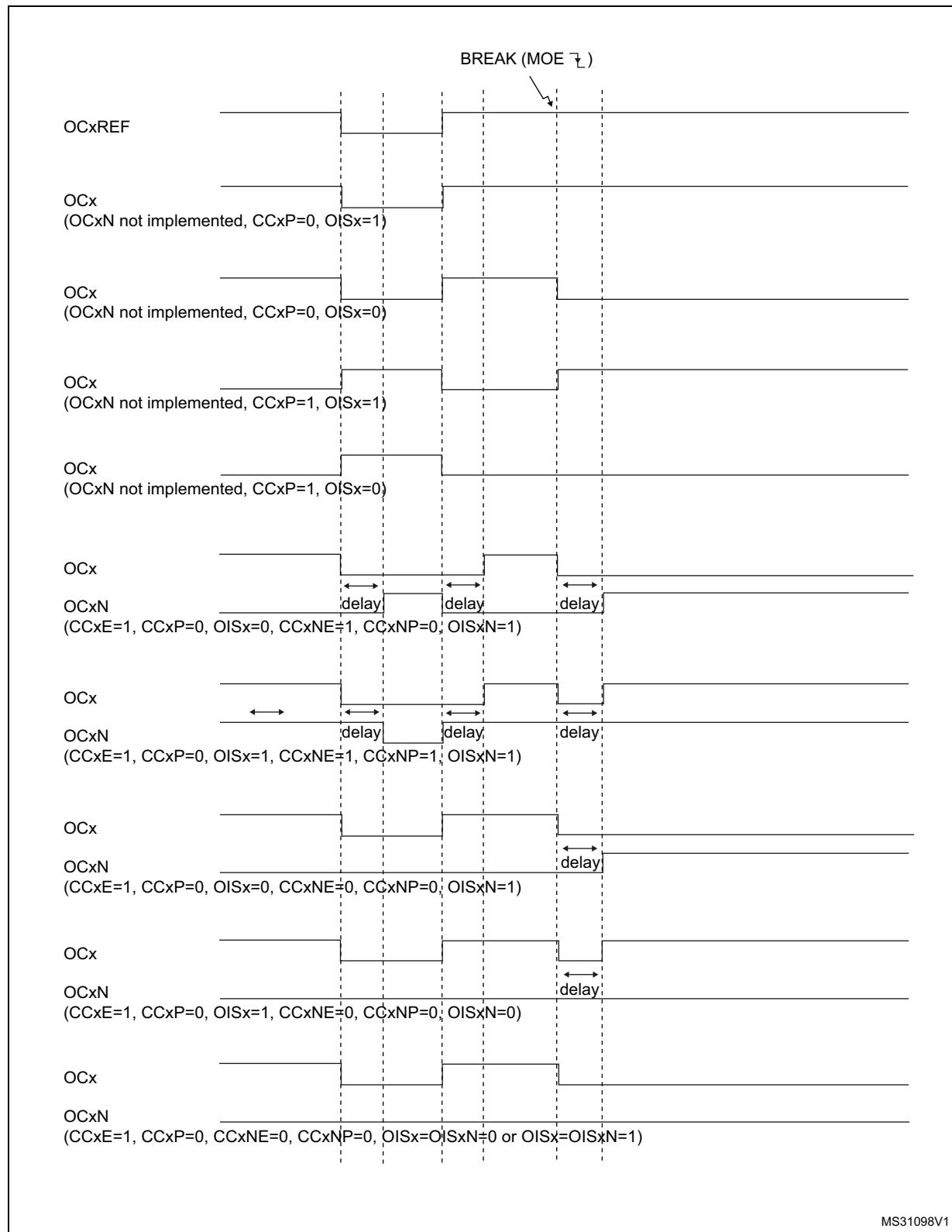
**Note:** *The break inputs is acting on level. Thus, the MOE cannot be set while the break input is active (neither automatically nor by software). In the meantime, the status flag BIF cannot be cleared.*

The break can be generated by the BRK input which has a programmable polarity and an enable bit BKE in the TIMx\_BDTR Register.

In addition to the break input and the output management, a write protection has been implemented inside the break circuit to safeguard the application. It allows to freeze the configuration of several parameters (dead-time duration, OCx/OCxN polarities and state when disabled, OCxM configurations, break enable and polarity). The protection can be selected among 3 levels with the LOCK bits in the TIMx\_BDTR register. Refer to [Section 20.5.16: TIM15 break and dead-time register \(TIM15\\_BDTR\) on page 538](#). The LOCK bits can be written only once after an MCU reset.

The [Figure 196](#) shows an example of behavior of the outputs in response to a break.

Figure 196. Output behavior in response to a break



MS31098V1

### 20.4.13 One-pulse mode

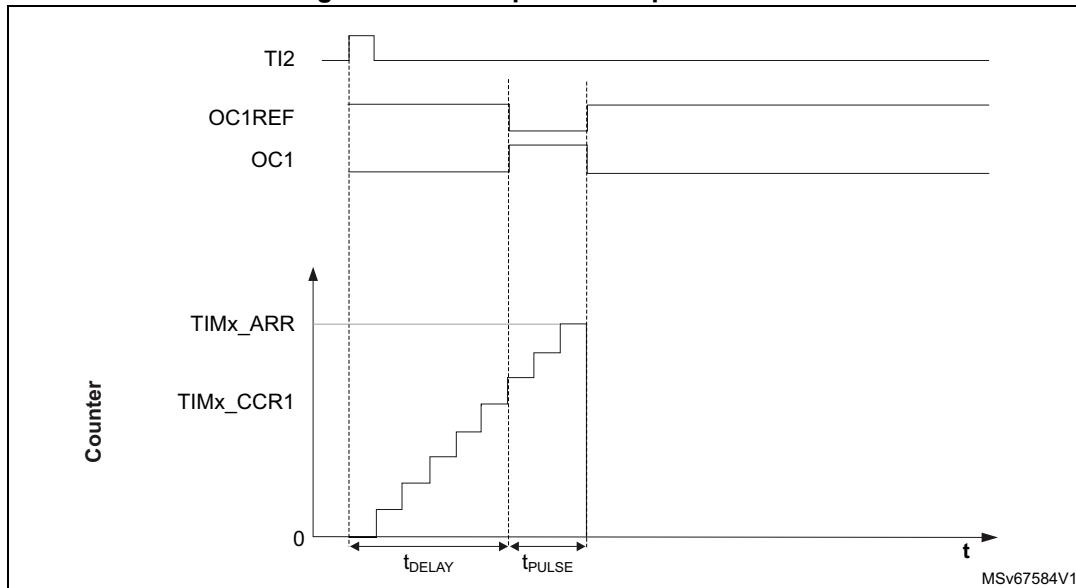
One-pulse mode (OPM) is a particular case of the previous modes. It allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length after a programmable delay.

Starting the counter can be controlled through the slave mode controller. Generating the waveform can be done in output compare mode or PWM mode. One-pulse mode is selected by setting the OPM bit in the TIMx\_CR1 register. This makes the counter stop automatically at the next update event UEV.

A pulse can be correctly generated only if the compare value is different from the counter initial value. Before starting (when the timer is waiting for the trigger), the configuration must be:

- In upcounting: CNT < CCRx  $\leq$  ARR (in particular, 0 < CCRx)
- In downcounting: CNT > CCRx

**Figure 197. Example of One-pulse mode**



For example one may want to generate a positive pulse on OC1 with a length of  $t_{PULSE}$  and after a delay of  $t_{DELAY}$  as soon as a positive edge is detected on the TI2 input pin.

Let's use TI2FP2 as trigger 1:

- Map TI2FP2 to TI2 by writing CC2S='01' in the TIMx\_CCMR1 register.
- TI2FP2 must detect a rising edge, write CC2P='0' in the TIMx\_CCER register.
- Configure TI2FP2 as trigger for the slave mode controller (TRGI) by writing TS='110' in the TIMx\_SMCR register.
- TI2FP2 is used to start the counter by writing SMS to '110' in the TIMx\_SMCR register (trigger mode).

For code example refer to the Appendix section [A.9.16: One-Pulse mode code example](#).

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

- The  $t_{DELAY}$  is defined by the value written in the TIMx\_CCR1 register.
- The  $t_{PULSE}$  is defined by the difference between the auto-reload value and the compare value (TIMx\_ARR - TIMx\_CCR1).
- Let's say one want to build a waveform with a transition from '0' to '1' when a compare match occurs and a transition from '1' to '0' when the counter reaches the auto-reload value. To do this PWM mode 2 must be enabled by writing OC1M=111 in the TIMx\_CCMR1 register. Optionally the preload registers can be enabled by writing OC1PE='1' in the TIMx\_CCMR1 register and ARPE in the TIMx\_CR1 register. In this case one has to write the compare value in the TIMx\_CCR1 register, the auto-reload value in the TIMx\_ARR register, generate an update by setting the UG bit and wait for external trigger event on TI2. CC1P is written to '0' in this example.

Since only 1 pulse is needed, a 1 must be written in the OPM bit in the TIMx\_CR1 register to stop the counter at the next update event (when the counter rolls over from the auto-reload value back to 0).

#### Particular case: OCx fast enable

In One-pulse mode, the edge detection on TIx input set the CEN bit which enables the counter. Then the comparison between the counter and the compare value makes the output toggle. But several clock cycles are needed for these operations and it limits the minimum delay  $t_{DELAY}$  min we can get.

If one wants to output a waveform with the minimum delay, the OCxFE bit can be set in the TIMx\_CCMRx register. Then OCxRef (and OCx) are forced in response to the stimulus, without taking in account the comparison. Its new level is the same as if a compare match had occurred. OCxFE acts only if the channel is configured in PWM1 or PWM2 mode.

For code example refer to the part of code conditioned by PULSE\_WITHOUT\_DELAY > 0 in the Appendix section [A.9.16: One-Pulse mode code example](#).

### 20.4.14 TIM15 external trigger synchronization

This section applies to STM32F05x, STM32F07x and STM32F09x devices only.

The TIM15 timer can be synchronized with an external trigger in several modes: Reset mode, Gated mode and Trigger mode.

#### Slave mode: Reset mode

The counter and its prescaler can be reinitialized in response to an event on a trigger input. Moreover, if the URS bit from the TIMx\_CR1 register is low, an update event UEV is generated. Then all the preloaded registers (TIMx\_ARR, TIMx\_CCRx) are updated.

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

- Configure the channel 1 to detect rising edges on TI1. Configure the input filter duration (in this example, we do not need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. The CC1S bits

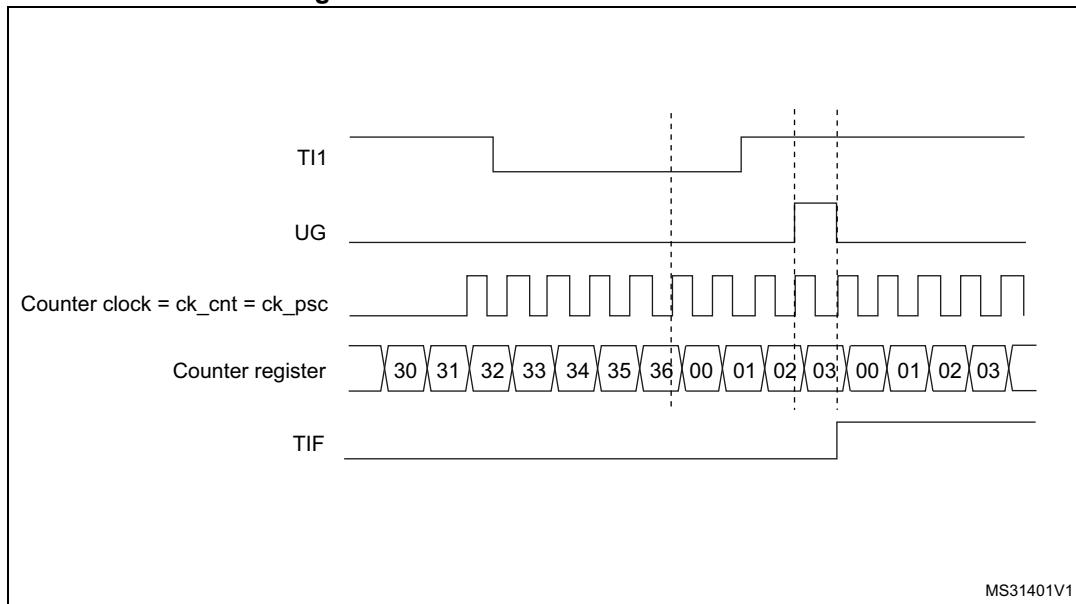
- select the input capture source only, CC1S = 01 in the TIMx\_CCMR1 register. Write CC1P=0 in TIMx\_CCER register to validate the polarity (and detect rising edges only).
- Configure the timer in reset mode by writing SMS=100 in TIMx\_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx\_SMCR register.
- Start the counter by writing CEN=1 in the TIMx\_CR1 register.

For code example refer to the Appendix section [A.9.12: Reset mode code example](#).

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

The following figure shows this behavior when the auto-reload register TIMx\_ARR=0x36. The delay between the rising edge on TI1 and the actual reset of the counter is due to the resynchronization circuit on TI1 input.

**Figure 198. Control circuit in reset mode**



### Slave mode: Gated mode

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

In the following example, the upcounter counts only when TI1 input is low:

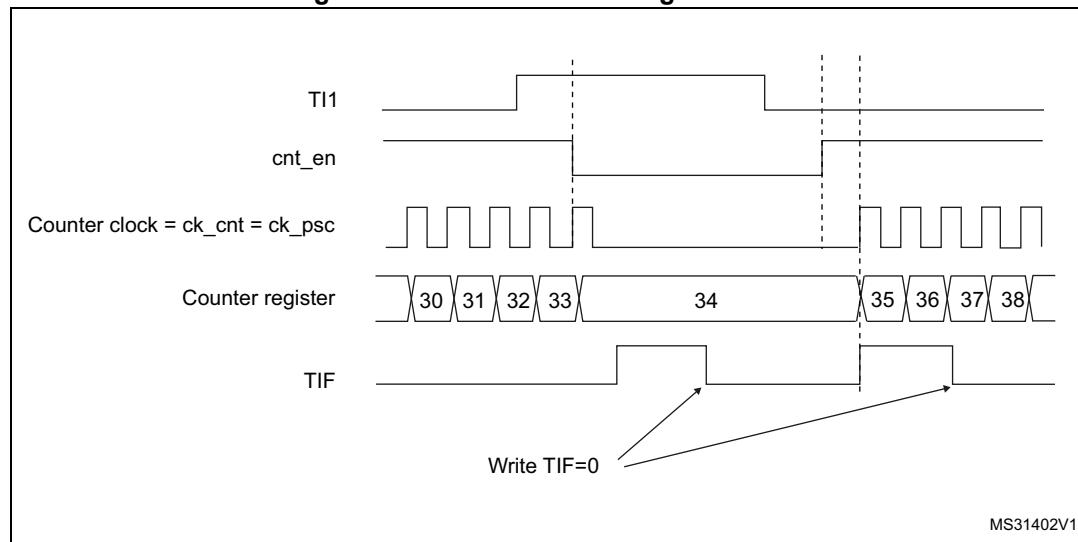
- Configure the channel 1 to detect low levels on TI1. Configure the input filter duration (in this example, we do not need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. The CC1S bits select the input capture source only, CC1S=01 in TIMx\_CCMR1 register. Write CC1P=1 in TIMx\_CCER register to validate the polarity (and detect low level only).
- Configure the timer in gated mode by writing SMS=101 in TIMx\_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx\_SMCR register.
- Enable the counter by writing CEN=1 in the TIMx\_CR1 register (in gated mode, the counter doesn't start if CEN=0, whatever is the trigger input level).

For code example refer to the Appendix section [A.9.13: Gated mode code example](#).

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

The delay between the rising edge on TI1 and the actual stop of the counter is due to the resynchronization circuit on TI1 input.

**Figure 199. Control circuit in gated mode**



### Slave mode: Trigger mode

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

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

- Configure the channel 2 to detect rising edges on TI2. Configure the input filter duration (in this example, we do not need any filter, so we keep IC2F=0000). The capture prescaler is not used for triggering, so it does not need to be configured. The CC2S bits are configured to select the input capture source only, CC2S=01 in TIMx\_CCMR1

register. Write CC2P=1 in TIMx\_CCER register to validate the polarity (and detect low level only).

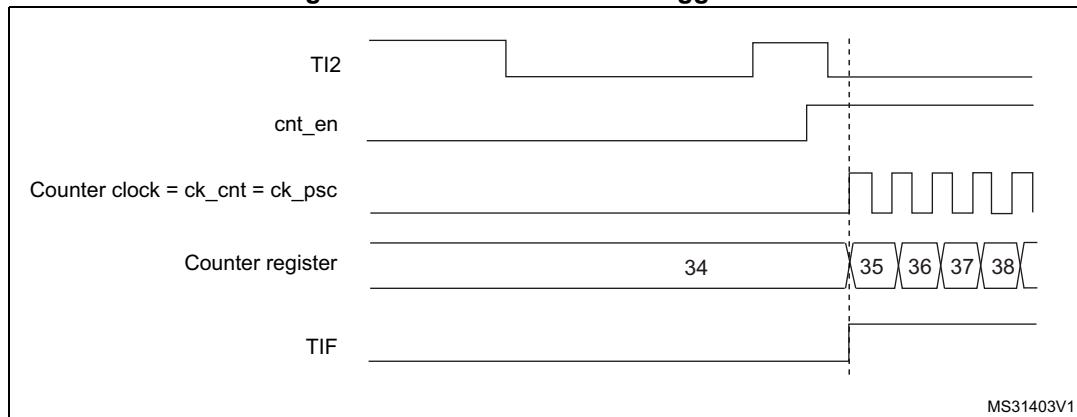
- Configure the timer in trigger mode by writing SMS=110 in TIMx\_SMCR register. Select TI2 as the input source by writing TS=110 in TIMx\_SMCR register.

For code example refer to the Appendix section [A.9.14: Trigger mode code example](#).

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

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

**Figure 200. Control circuit in trigger mode**



The TIM timers are linked together internally for timer synchronization or chaining. Refer to [Section 18.3.15: Timer synchronization on page 439](#) for details.

#### 20.4.15 Timer synchronization (TIM15)

This section applies to STM32F05x, STM32F07x and STM32F09x devices only.

The TIM timers are linked together internally for timer synchronization or chaining. Refer to [Section 18.3.15: Timer synchronization on page 439](#) for details.

#### 20.4.16 Debug mode

When the microcontroller enters debug mode (Cortex™-M0 core halted), the TIMx counter either continues to work normally or stops, depending on DBG\_TIMx\_STOP configuration bit in DBG module.

## 20.5 TIM15 registers

Refer to [Section 1.2 on page 42](#) for a list of abbreviations used in register descriptions.

### 20.5.1 TIM15 control register 1 (TIM15\_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	CKD[1:0]		ARPE	Res.	Res.	Res.	OPM	URS	UDIS	CEN

Bits 15:10 Reserved, must be kept at reset value.

Bits 9:8 **CKD[1:0]**: Clock division

This bit field indicates the division ratio between the timer clock (CK\_INT) frequency and the dead-time and sampling clock ( $t_{DTS}$ ) used by the dead-time generators and the digital filters (Tlx)

- 00:  $t_{DTS} = t_{CK\_INT}$
- 01:  $t_{DTS} = 2*t_{CK\_INT}$
- 10:  $t_{DTS} = 4*t_{CK\_INT}$
- 11: Reserved, do not program this value

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

- 0: TIMx\_ARR register is not buffered
- 1: TIMx\_ARR register is buffered

Bits 6:4 Reserved, must be kept at reset value.

Bit 3 **OPM**: One-pulse mode

- 0: Counter is not stopped at update event
- 1: Counter stops counting at the next update event (clearing the bit CEN)

Bit 2 **URS**: Update request source

This bit is set and cleared by software to select the UEV event sources.

0: Any of the following events generate an update interrupt if enabled. These events can be:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

1: Only counter overflow/underflow generates an update interrupt if enabled

Bit 1 **UDIS**: Update disable

This bit is set and cleared by software to enable/disable UEV event generation.

0: UEV enabled. The Update (UEV) event is generated by one of the following events:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

Buffered registers are then loaded with their preload values.

1: UEV disabled. The Update event is not generated, shadow registers keep their value (ARR, PSC, CCRx). However the counter and the prescaler are reinitialized if the UG bit is set or if a hardware reset is received from the slave mode controller.

Bit 0 **CEN**: Counter enable

0: Counter disabled

1: Counter enabled

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

**20.5.2 TIM15 control register 2 (TIM15\_CR2)**

Address offset: 0x04

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	OIS2	OIS1N	OIS1	Res.	MMS[2:0]			CCDS	CCUS	Res.	CCPC

Bits 15:11 Reserved, must be kept at reset value.

Bit 10 **OIS2**: Output idle state 2 (OC2 output)

- 0: OC2=0 when MOE=0
- 1: OC2=1 when MOE=0

*Note: This bit cannot be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in the TIMx\_BKR register).*

Bit 9 **OIS1N**: Output Idle state 1 (OC1N output)

- 0: OC1N=0 after a dead-time when MOE=0
- 1: OC1N=1 after a dead-time when MOE=0

*Note: This bit can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx\_BKR register).*

Bit 8 **OIS1**: Output Idle state 1 (OC1 output)

- 0: OC1=0 (after a dead-time if OC1N is implemented) when MOE=0
- 1: OC1=1 (after a dead-time if OC1N is implemented) when MOE=0

*Note: This bit can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx\_BKR register).*

Bit 7 Reserved, must be kept at reset value.

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

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

000: **Reset** - the UG bit from the TIMx\_EGR register is used as trigger output (TRGO). If the reset is generated by the trigger input (slave mode controller configured in reset mode) then the signal on TRGO is delayed compared to the actual reset.

001: **Enable** - the Counter Enable signal CNT\_EN is used as trigger output (TRGO). It is useful to start several timers at the same time or to control a window in which a slave timer is enable. The Counter Enable signal is generated by a logic OR between CEN control bit and the trigger input when configured in gated mode. When the Counter Enable signal is controlled by the trigger input, there is a delay on TRGO, except if the master/slave mode is selected (see the MSM bit description in TIMx\_SMCR register).

010: **Update** - The update event is selected as trigger output (TRGO). For instance a master timer can then be used as a prescaler for a slave timer.

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

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

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

Bit 3 **CCDS**: Capture/compare DMA selection

0: CCx DMA request sent when CCx event occurs

1: CCx DMA requests sent when update event occurs

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

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

1: When capture/compare control bits are preloaded (CCPC=1), they are updated by setting the COMG bit or when an rising edge occurs on TRGI.

*Note: This bit acts only on channels that have a complementary output.*

Bit 1 Reserved, must be kept at reset value.

Bit 0 **CCPC**: Capture/compare preloaded control

0: CCxE, CCxNE and OCxM bits are not preloaded

1: CCxE, CCxNE and OCxM bits are preloaded, after having been written, they are updated only when COM bit is set.

*Note: This bit acts only on channels that have a complementary output.*

### 20.5.3 TIM15 slave mode control register (TIM15\_SMCR)

Address offset: 0x08

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	MSM	TS[2:0]			Res.	SMS[2:0]									

Bits 15:8 Reserved, must be kept at reset value.

Bit 7 **MSM:** Master/slave mode

0: No action

1: The effect of an event on the trigger input (TRGI) is delayed to allow a perfect synchronization between the current timer and its slaves (through TRGO). It is useful if we want to synchronize several timers on a single external event.

Bits 6:4 **TS[2:0]:** Trigger selection

This bit field selects the trigger input to be used to synchronize the counter.

000: Internal Trigger 0 (ITR0)

001: Internal Trigger 1 (ITR1)

010: Internal Trigger 2 (ITR2)

011: Internal Trigger 3 (ITR3)

100: TI1 Edge Detector (TI1F\_ED)

101: Filtered Timer Input 1 (TI1FP1)

110: Filtered Timer Input 2 (TI2FP2)

See [Table 71: TIMx Internal trigger connection on page 525](#) for more details on ITRx meaning for each Timer.

*Note: These bits must be changed only when they are not used (e.g. when SMS=000) to avoid wrong edge detections at the transition.*

Bit 3 Reserved, must be kept at reset value.

Bits 2:0 **SMS[2:0]:** Slave mode selection

When external signals are selected the active edge of the trigger signal (TRGI) is linked to the polarity selected on the external input (see Input Control register and Control Register description).

000: Slave mode disabled - if CEN = '1' then the prescaler is clocked directly by the internal clock.

100: Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter and generates an update of the registers.

101: Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high. The counter stops (but is not reset) as soon as the trigger becomes low. Both start and stop of the counter are controlled.

110: Trigger Mode - The counter starts at a rising edge of the trigger TRGI (but it is not reset). Only the start of the counter is controlled.

111: External Clock Mode 1 - Rising edges of the selected trigger (TRGI) clock the counter.

*Note: The gated mode must not be used if TI1F\_ED is selected as the trigger input (TS='100'). Indeed, TI1F\_ED outputs 1 pulse for each transition on TI1F, whereas the gated mode checks the level of the trigger signal.*

**Table 71. TIMx Internal trigger connection**

Slave TIM	ITR0 (TS = 000)	ITR1 (TS = 001)	ITR2 (TS = 010)	ITR3 (TS = 011)
TIM15	TIM2	TIM3	TIM16_OC	TIM17_OC

## 20.5.4 TIM15 DMA/interrupt enable register (TIM15\_DIER)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res.	Res.	Res.	CC2DE	CC1DE	UDE	BIE	TIE	COMIE	Res.	Res.	CC2IE	CC1IE	UIE
	rw				rw	rw	rw	rw	rw	rw			rw	rw	rw

Bit 15 Reserved, must be kept at reset value.

Bit 14 **TDE**: Trigger DMA request enable

- 0: Trigger DMA request disabled
- 1: Trigger DMA request enabled

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

Bit 10 **CC2DE**: Capture/Compare 2 DMA request enable

- 0: CC2 DMA request disabled
- 1: CC2 DMA request enabled

Bit 9 **CC1DE**: Capture/Compare 1 DMA request enable

- 0: CC1 DMA request disabled
- 1: CC1 DMA request enabled

Bit 8 **UDE**: Update DMA request enable

- 0: Update DMA request disabled
- 1: Update DMA request enabled

Bit 7 **BIE**: Break interrupt enable

- 0: Break interrupt disabled
- 1: Break interrupt enabled

Bit 6 **TIE**: Trigger interrupt enable

- 0: Trigger interrupt disabled
- 1: Trigger interrupt enabled

Bit 5 **COMIE**: COM interrupt enable

- 0: COM interrupt disabled
- 1: COM interrupt enabled

Bits 4:3 Reserved, must be kept at reset value.

Bit 2 **CC2IE**: Capture/Compare 2 interrupt enable

- 0: CC2 interrupt disabled
- 1: CC2 interrupt enabled

Bit 1 **CC1IE**: Capture/Compare 1 interrupt enable

- 0: CC1 interrupt disabled
- 1: CC1 interrupt enabled

Bit 0 **UIE**: Update interrupt enable

- 0: Update interrupt disabled
- 1: Update interrupt enabled

### 20.5.5 TIM15 status register (TIM15\_SR)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	CC2OF	CC1OF	Res.	BIF	TIF	COMIF	Res.	Res.	CC2IF	CC1IF	UIF
					rc_w0	rc_w0		rc_w0	rc_w0	rc_w0			rc_w0	rc_w0	rc_w0

Bits 15:11 Reserved, must be kept at reset value.

Bit 10 **CC2OF**: Capture/Compare 2 overcapture flag

Refer to CC1OF description.

Bit 9 **CC1OF**: Capture/Compare 1 overcapture flag

This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to '0'.

0: No overcapture has been detected

1: The counter value has been captured in TIMx\_CCR1 register while CC1IF flag was already set

Bit 8 Reserved, must be kept at reset value.

Bit 7 **BIF**: Break interrupt flag

This flag is set by hardware as soon as the break input goes active. It can be cleared by software if the break input is not active.

0: No break event occurred

1: An active level has been detected on the break input

Bit 6 **TIF**: Trigger interrupt flag

This flag is set by hardware on trigger event (active edge detected on TRGI input when the slave mode controller is enabled in all modes but gated mode, both edges in case gated mode is selected). It is cleared by software.

0: No trigger event occurred

1: Trigger interrupt pending

Bit 5 **COMIF**: COM interrupt flag

This flag is set by hardware on a COM event (once the capture/compare control bits –CCxE, CCxNE, OCxM– have been updated). It is cleared by software by writing it to '0'.

0: No COM event occurred

1: COM interrupt pending

Bits 4:3 Reserved, must be kept at reset value.

Bit 2 **CC2IF**: Capture/Compare 2 interrupt flag  
refer to CC1IF description

Bit 1 **CC1IF**: Capture/Compare 1 interrupt flag

**Condition: channel CC1 is configured as output**

This flag is set by hardware when the counter matches the compare value, it is cleared by software.

0: No match.

1: The content of the counter TIMx\_CNT matches the content of the TIMx\_CCR1 register.  
When the contents of TIMx\_CCR1 are greater than the contents of TIMx\_ARR, the CC1IF bit goes high on the counter overflow (in upcounting and up/down-counting modes) or underflow (in downcounting mode)

**Condition: If channel CC1 is configured as input**

This bit is set by hardware on a capture. It is cleared by software or by reading the TIMx\_CCR1 register.

0: No input capture occurred

1: The counter value has been captured in TIMx\_CCR1 register (An edge has been detected on IC1 which matches the selected polarity)

Bit 0 **UIF**: Update interrupt flag

This bit is set by hardware on an update event. It is cleared by software.

0: No update occurred.

1: Update interrupt pending. This bit is set by hardware when the registers are updated:  
–At overflow regarding the repetition counter value (update if repetition counter = 0) and if the UDIS=0 in the TIMx\_CR1 register.  
–When CNT is reinitialized by software using the UG bit in TIMx\_EGR register, if URS=0 and UDIS=0 in the TIMx\_CR1 register.  
–When CNT is reinitialized by a trigger event (refer to [Section 20.5.3: TIM15 slave mode control register \(TIM15\\_SMCR\)](#)), if URS=0 and UDIS=0 in the TIMx\_CR1 register.

## 20.5.6 TIM15 event generation register (TIM15\_EGR)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	BG	TG	COMG	Res.	Res.	CC2G	CC1G	UG							
								w	w	rw			w	w	w

Bits 15:8 Reserved, must be kept at reset value.

**Bit 7 BG:** Break generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: A break event is generated. MOE bit is cleared and BIF flag is set. Related interrupt or DMA transfer can occur if enabled.

**Bit 6 TG:** Trigger generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: The TIF flag is set in TIMx\_SR register. Related interrupt or DMA transfer can occur if enabled

**Bit 5 COMG:** Capture/Compare control update generation

This bit can be set by software, it is automatically cleared by hardware.

0: No action

1: When the CCPC bit is set, it is possible to update the CCxE, CCxNE and OCxM bits

*Note: This bit acts only on channels that have a complementary output.*

Bits 4:3 Reserved, must be kept at reset value.

**Bit 2 CC2G:** Capture/Compare 2 generation

Refer to CC1G description

**Bit 1 CC1G:** Capture/Compare 1 generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: A capture/compare event is generated on channel 1:

**Condition: channel CC1 is configured as output**

CC1IF flag is set, Corresponding interrupt or DMA request is sent if enabled.

**Condition: channel CC1 is configured as input**

The current value of the counter is captured in TIMx\_CCR1 register. The CC1IF flag is set, the corresponding interrupt or DMA request is sent if enabled. The CC1OF flag is set if the CC1IF flag was already high.

**Bit 0 UG:** Update generation

This bit can be set by software, it is automatically cleared by hardware.

0: No action.

1: Reinitialize the counter and generates an update of the registers. Note that the prescaler counter is cleared too (anyway the prescaler ratio is not affected).

### 20.5.7 TIM15 capture/compare mode register 1 [alternate] (TIM15\_CCMR1)

Address offset: 0x18

Reset value: 0x0000

The same register can be used for input capture mode (this section) or for output compare mode (next section). The direction of a channel is defined by configuring the corresponding CC<sub>x</sub>S bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OC<sub>xx</sub> describes its function when the channel is configured in output, IC<sub>xx</sub> describes its function when the channel is configured in input. So one must take care that the same bit can have a different meaning for the input stage and for the output stage.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IC2F[3:0]				IC2PSC[1:0]		CC2S[1:0]		IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:12 **IC2F[3:0]**: Input capture 2 filter

Bits 11:10 **IC2PSC[1:0]**: Input capture 2 prescaler

Bits 9:8 **CC2S[1:0]**: Capture/Compare 2 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output

01: CC2 channel is configured as input, IC2 is mapped on TI2

10: CC2 channel is configured as input, IC2 is mapped on TI1

11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note:* CC2S bits are writable only when the channel is OFF (CC2E = '0' in TIMx\_CCER).

Bits 7:4 **IC1F[3:0]**: Input capture 1 filter

This bit-field defines the frequency used to sample TI1 input and the length of the digital filter applied to TI1. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output:

- 0000: No filter, sampling is done at  $f_{DTS}$
- 0001:  $f_{SAMPLING} = f_{CK\_INT}$ ,  $N = 2$
- 0010:  $f_{SAMPLING} = f_{CK\_INT}$ ,  $N = 4$
- 0011:  $f_{SAMPLING} = f_{CK\_INT}$ ,  $N = 8$
- 0100:  $f_{SAMPLING} = f_{DTS} / 2$ ,  $N = 6$
- 0101:  $f_{SAMPLING} = f_{DTS} / 2$ ,  $N = 8$
- 0110:  $f_{SAMPLING} = f_{DTS} / 4$ ,  $N = 6$
- 0111:  $f_{SAMPLING} = f_{DTS} / 4$ ,  $N = 8$
- 1000:  $f_{SAMPLING} = f_{DTS} / 8$ ,  $N = 6$
- 1001:  $f_{SAMPLING} = f_{DTS} / 8$ ,  $N = 8$
- 1010:  $f_{SAMPLING} = f_{DTS} / 16$ ,  $N = 5$
- 1011:  $f_{SAMPLING} = f_{DTS} / 16$ ,  $N = 6$
- 1100:  $f_{SAMPLING} = f_{DTS} / 16$ ,  $N = 8$
- 1101:  $f_{SAMPLING} = f_{DTS} / 32$ ,  $N = 5$
- 1110:  $f_{SAMPLING} = f_{DTS} / 32$ ,  $N = 6$
- 1111:  $f_{SAMPLING} = f_{DTS} / 32$ ,  $N = 8$

*Note: Care must be taken that  $f_{DTS}$  is replaced in the formula by  $CK\_INT$  when  $ICxF[3:0] = 1, 2$  or  $3$ .*

Bits 3:2 **IC1PSC[1:0]**: Input capture 1 prescaler

This bit-field defines the ratio of the prescaler acting on CC1 input (IC1).

The prescaler is reset as soon as  $CC1E='0'$  (TIMx\_CCER register).

- 00: no prescaler, capture is done each time an edge is detected on the capture input
- 01: capture is done once every 2 events
- 10: capture is done once every 4 events
- 11: capture is done once every 8 events

Bits 1:0 **CC1S[1:0]**: Capture/Compare 1 Selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output

01: CC1 channel is configured as input, IC1 is mapped on TI1

10: CC1 channel is configured as input, IC1 is mapped on TI2

11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC1S bits are writable only when the channel is OFF ( $CC1E = '0'$  in TIMx\_CCER).*

## 20.5.8 TIM15 capture/compare mode register 1 [alternate] (TIM15\_CCMR1)

Address offset: 0x18

Reset value: 0x0000

The same register can be used for output compare mode (this section) or for input capture mode (previous section). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OCxx describes its function when the channel is configured in output, ICxx describes its function when the channel is configured in input. So one must take care that the same bit can have a different meaning for the input stage and for the output stage.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	OC2M[2:0]			OC2 PE	OC2 FE	CC2S[1:0]		Res.	OC1M[2:0]			OC1 PE	OC1 FE	CC1S[1:0]	
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

Bit 15 Reserved, must be kept at reset value.

Bits 14:12 **OC2M[2:0]**: Output Compare 2 mode

Bit 11 **OC2PE**: Output Compare 2 preload enable

Bit 10 **OC2FE**: Output Compare 2 fast enable

Bits 9:8 **CC2S[1:0]**: Capture/Compare 2 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output.

01: CC2 channel is configured as input, IC2 is mapped on TI2.

10: CC2 channel is configured as input, IC2 is mapped on TI1.

11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode is working only if an internal trigger input is selected through the TS bit (TIMx\_SMCR register)

*Note:* CC2S bits are writable only when the channel is OFF (CC2E = '0' in TIMx\_CCER).

Bit 7 Reserved, must be kept at reset value.

Bits 6:4 **OC1M[2:0]**: Output Compare 1 mode

These bits define the behavior of the output reference signal OC1REF from which OC1 and OC1N are derived. OC1REF is active high whereas OC1 and OC1N active level depends on CC1P and CC1NP bits.

000: Frozen - The comparison between the output compare register TIMx\_CCR1 and the counter TIMx\_CNT has no effect on the outputs.

001: Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

011: Toggle - OC1REF toggles when TIMx\_CNT=TIMx\_CCR1.

100: Force inactive level - OC1REF is forced low.

101: Force active level - OC1REF is forced high.

110: PWM mode 1 - In upcounting, channel 1 is active as long as TIMx\_CNT<TIMx\_CCR1 else inactive. In downcounting, channel 1 is inactive (OC1REF='0') as long as TIMx\_CNT>TIMx\_CCR1 else active (OC1REF='1').

111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIMx\_CNT<TIMx\_CCR1 else active. In downcounting, channel 1 is active as long as TIMx\_CNT>TIMx\_CCR1 else inactive.

*Note:* These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx\_BDTR register) and CC1S='00' (the channel is configured in output).

*Note:* In PWM mode 1 or 2, the OCREF level changes only when the result of the comparison changes or when the output compare mode switches from "frozen" mode to "PWM" mode.

Bit 3 **OC1PE**: Output Compare 1 preload enable

0: Preload register on TIMx\_CCR1 disabled. TIMx\_CCR1 can be written at anytime, the new value is taken in account immediately.

1: Preload register on TIMx\_CCR1 enabled. Read/Write operations access the preload register. TIMx\_CCR1 preload value is loaded in the active register at each update event.

*Note: These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx\_BDTR register) and CC1S='00' (the channel is configured in output).*

Bit 2 **OC1FE**: Output Compare 1 fast enable

This bit is used to accelerate the effect of an event on the trigger in input on the CC output.

0: CC1 behaves normally depending on counter and CCR1 values even when the trigger is ON. The minimum delay to activate CC1 output when an edge occurs on the trigger input is 5 clock cycles.

1: An active edge on the trigger input acts like a compare match on CC1 output. Then, OC is set to the compare level independently of the result of the comparison. Delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. OCFE acts only if the channel is configured in PWM1 or PWM2 mode.

Bits 1:0 **CC1S[1:0]**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output.

01: CC1 channel is configured as input, IC1 is mapped on TI1.

10: CC1 channel is configured as input, IC1 is mapped on TI2.

11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC1S bits are writable only when the channel is OFF (CC1E = '0' in TIMx\_CCER).*

## 20.5.9 TIM15 capture/compare enable register (TIM15\_CCER)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CC2NP	Res.	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E							

Bits 15:8 Reserved, must be kept at reset value.

Bit 7 **CC2NP**: Capture/Compare 2 complementary output polarity

Refer to CC1NP description.

Bit 6 Reserved, must be kept at reset value.

Bit 5 **CC2P**: Capture/Compare 2 output polarity

Refer to CC1P description.

Bit 4 **CC2E**: Capture/Compare 2 output enable

Refer to CC1E description.

Bit 3 **CC1NP**: Capture/Compare 1 complementary output polarity

- 0: OC1N active high
- 1: OC1N active low

*Note: This bit is not writable as soon as LOCK level 2 or 3 has been programmed (LOCK bits in TIMx\_BDTR register) and CC1S="00" (the channel is configured in output).*

Bit 2 **CC1NE**: Capture/Compare 1 complementary output enable

- 0: Off - OC1N is not active. OC1N level is then function of MOE, OSS1, OSSR, OIS1, OIS1N and CC1E bits.
- 1: On - OC1N signal is output on the corresponding output pin depending on MOE, OSS1, OSSR, OIS1, OIS1N and CC1E bits.

Bit 1 **CC1P**: Capture/Compare 1 output polarity

**Condition: CC1 channel configured as output**

- 0: OC1 active high
- 1: OC1 active low

**Condition: CC1 channel configured as input**

The CC1NP/CC1P bits select the polarity of TI1FP1 and TI2FP1 for trigger or capture operations.

- 00: Noninverted/rising edge: circuit is sensitive to TIxFP1's rising edge (capture, trigger in reset or trigger mode), TIxFP1 is not inverted (trigger in gated mode).
- 01: Inverted/falling edge: circuit is sensitive to TIxFP1's falling edge (capture, trigger in reset, or trigger mode), TIxFP1 is inverted (trigger in gated mode).
- 10: Reserved, do not use this configuration.
- 11: Noninverted/both edges: circuit is sensitive to both the rising and falling edges of TIxFP1 (capture, trigger in reset or trigger mode), TIxFP1 is not inverted (trigger in gated mode).

*Note: This bit is not writable as soon as LOCK level 2 or 3 has been programmed (LOCK bits in TIMx\_BDTR register).*

Bit 0 **CC1E**: Capture/Compare 1 output enable

**Condition: CC1 channel configured as output**

- 0: Off - OC1 is not active. OC1 level is then function of MOE, OSS1, OSSR, OIS1, OIS1N and CC1NE bits.

- 1: On - OC1 signal is output on the corresponding output pin depending on MOE, OSS1, OSSR, OIS1, OIS1N and CC1NE bits.

**Condition: CC1 channel configured as input**

This bit determines if a capture of the counter value can actually be done into the input capture/compare register 1 (TIMx\_CCR1) or not.

- 0: Capture disabled
- 1: Capture enabled

**Table 72. Output control bits for complementary OCx and OCxN channels with break feature**

Control bits						Output states <sup>(1)</sup>	
MOE bit	OSSI bit	OSSR bit	CCxE bit	CCxNE bit	OCx output state	OCxN output state	
1	X	0	0	0	Output Disabled (not driven by the timer) OCx=0, OCx_EN=0	Output Disabled (not driven by the timer) OCxN=0, OCxN_EN=0	
		0	0	1	Output Disabled (not driven by the timer) OCx=0, OCx_EN=0	OCxREF + Polarity OCxN=OCxREF xor CCxNP, OCxN_EN=1	
		0	1	0	OCxREF + Polarity OCx=OCxREF xor CCxP, OCx_EN=1	Output Disabled (not driven by the timer) OCxN=0, OCxN_EN=0	
		0	1	1	OCREF + Polarity + dead-time OCx_EN=1	Complementary to OCREF (not OCREF) + Polarity + dead-time OCxN_EN=1	
		1	0	0	Output Disabled (not driven by the timer) OCx=CCxP, OCx_EN=0	Output Disabled (not driven by the timer) OCxN=CCxNP, OCxN_EN=0	
		1	0	1	Off-State (output enabled with inactive state) OCx=CCxP, OCx_EN=1	OCxREF + Polarity OCxN=OCxREF xor CCxNP, OCxN_EN=1	
		1	1	0	OCxREF + Polarity OCx=OCxREF xor CCxP, OCx_EN=1	Off-State (output enabled with inactive state) OCxN=CCxNP, OCxN_EN=1	
		1	1	1	OCREF + Polarity + dead-time OCx_EN=1	Complementary to OCREF (not OCREF) + Polarity + dead-time OCxN_EN=1	
0	X	0	0	0	Output Disabled (not driven by the timer)		
		0	0	1	Asynchronously: OCx=CCxP, OCx_EN=0, OCxN=CCxNP, OCxN_EN=0		
		0	1	0			
		0	1	1	Then if the clock is present: OCx=OISx and OCxN=OISxN after a dead-time, assuming that OISx and OISxN do not correspond to OCX and OCxN both in active state.		
		1	0	0			
		1	0	1	Off-State (output enabled with inactive state)		
		1	1	0	Asynchronously: OCx=CCxP, OCx_EN=1, OCxN=CCxNP, OCxN_EN=1		
		1	1	1	Then if the clock is present: OCx=OISx and OCxN=OISxN after a dead-time, assuming that OISx and OISxN do not correspond to OCX and OCxN both in active state		

1. When both outputs of a channel are not used (CCxE = CCxNE = 0), the OISx, OISxN, CCxP and CCxNP bits must be kept cleared.

**Note:** The state of the external I/O pins connected to the complementary OCx and OCxN channels depends on the OCx and OCxN channel state and the GPIO and AFIO registers.

### 20.5.10 TIM15 counter (TIM15\_CNT)

Address offset: 0x24

Reset value: 0x0000

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

Bits 15:0 **CNT[15:0]**: Counter value

### 20.5.11 TIM15 prescaler (TIM15\_PSC)

Address offset: 0x28

Reset value: 0x0000

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

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency (CK\_CNT) is equal to  $f_{CK\_PSC} / (PSC[15:0] + 1)$ .

PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx\_EGR register or through trigger controller when configured in “reset mode”).

### 20.5.12 TIM15 auto-reload register (TIM15\_ARR)

Address offset: 0x2C

Reset value: 0xFFFF

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

Bits 15:0 **ARR[15:0]**: Auto-reload value

ARR is the value to be loaded in the actual auto-reload register.

Refer to the [Section 19.3.1: Time-base unit on page 472](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

### 20.5.13 TIM15 repetition counter register (TIM15\_RCR)

Address offset: 0x30

Reset value: 0x0000

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

Bits 15:8 Reserved, must be kept at reset value.

Bits 7:0 **REP[7:0]**: Repetition counter value

These bits allow the user to set-up the update rate of the compare registers (i.e. periodic transfers from preload to active registers) when preload registers are enable, as well as the update interrupt generation rate, if this interrupt is enable.

Each time the REP\_CNT related downcounter reaches zero, an update event is generated and it restarts counting from REP value. As REP\_CNT is reloaded with REP value only at the repetition update event U\_RC, any write to the TIMx\_RCR register is not taken in account until the next repetition update event.

It means in PWM mode (REP+1) corresponds to the number of PWM periods in edge-aligned mode.

### 20.5.14 TIM15 capture/compare register 1 (TIM15\_CCR1)

Address offset: 0x34

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR1[15:0]**: Capture/Compare 1 value

**Condition: if channel CC1 is configured as output**

CCR1 is the value to be loaded in the actual capture/compare 1 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR1 register (bit OC1PE). Else the preload value is copied in the active capture/compare 1 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signaled on OC1 output.

**Condition: if channel CC1 is configured as input**

CCR1 is the counter value transferred by the last input capture 1 event (IC1).

### 20.5.15 TIM15 capture/compare register 2 (TIM15\_CCR2)

Address offset: 0x38

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR2[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR2[15:0]**: Capture/Compare 2 value

**Condition: channel CC2 is configured as output:**

CCR2 is the value to be loaded in the actual capture/compare 2 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR2 register (bit OC2PE). Else the preload value is copied in the active capture/compare 2 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signalled on OC2 output.

**Condition: channel CC2 is configured as input:**

CCR2 is the counter value transferred by the last input capture 2 event (IC2).

### 20.5.16 TIM15 break and dead-time register (TIM15\_BDTR)

Address offset: 0x44

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK[1:0]		DTG[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

**Note:** As the bits AOE, BKP, BKE, OSSR and DTG[7:0] can be write-locked depending on the LOCK configuration, it can be necessary to configure all of them during the first write access to the TIMx\_BDTR register.

Bit 15 **MOE**: Main output enable

This bit is cleared asynchronously by hardware as soon as the break input is active. It is set by software or automatically depending on the AOE bit. It is acting only on the channels which are configured in output.

0: OC and OCN outputs are disabled or forced to idle state

1: OC and OCN outputs are enabled if their respective enable bits are set (CCxE, CCxNE in TIMx\_CCER register)

See OC/OCN enable description for more details ([Section 20.5.9: TIM15 capture/compare enable register \(TIM15\\_CCER\) on page 533](#)).

Bit 14 **AOE**: Automatic output enable

0: MOE can be set only by software

1: MOE can be set by software or automatically at the next update event (if the break input is not active)

**Note:** This bit can not be modified as long as LOCK level 1 has been programmed (LOCK bits in TIMx\_BDTR register).

Bit 13 **BKP**: Break polarity

0: Break input BRK is active low

1: Break input BRK is active high

**Note:** This bit can not be modified as long as LOCK level 1 has been programmed (LOCK bits in TIMx\_BDTR register).

**Note:** Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.

Bit 12 **BKE**: Break enable

- 0: Break inputs (BRK and CCS clock failure event) disabled
- 1: Break inputs (BRK and CCS clock failure event) enabled

*Note: This bit cannot be modified when LOCK level 1 has been programmed (LOCK bits in TIMx\_BDTR register).*

*Note: Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.*

Bit 11 **OSSR**: Off-state selection for Run mode

This bit is used when MOE=1 on channels having a complementary output which are configured as outputs. OSSR is not implemented if no complementary output is implemented in the timer.

See OC/OCN enable description for more details ([Section 20.5.9: TIM15 capture/compare enable register \(TIM15\\_CCER\) on page 533](#)).

- 0: When inactive, OC/OCN outputs are disabled (OC/OCN enable output signal=0)
- 1: When inactive, OC/OCN outputs are enabled with their inactive level as soon as CCxE=1 or CCxNE=1. Then, OC/OCN enable output signal=1

*Note: This bit can not be modified as soon as the LOCK level 2 has been programmed (LOCK bits in TIMx\_BDTR register).*

Bit 10 **OSSI**: Off-state selection for Idle mode

This bit is used when MOE=0 on channels configured as outputs.

See OC/OCN enable description for more details ([Section 20.5.9: TIM15 capture/compare enable register \(TIM15\\_CCER\) on page 533](#)).

- 0: When inactive, OC/OCN outputs are disabled (OC/OCN enable output signal=0)
- 1: When inactive, OC/OCN outputs are forced first with their idle level as soon as CCxE=1 or CCxNE=1. OC/OCN enable output signal=1

*Note: This bit can not be modified as soon as the LOCK level 2 has been programmed (LOCK bits in TIMx\_BDTR register).*

Bits 9:8 **LOCK[1:0]**: Lock configuration

These bits offer a write protection against software errors.

00: LOCK OFF - No bit is write protected

01: LOCK Level 1 = DTG bits in TIMx\_BDTR register, OISx and OISxN bits in TIMx\_CR2 register and BKE/BKP/AOE bits in TIMx\_BDTR register can no longer be written

10: LOCK Level 2 = LOCK Level 1 + CC Polarity bits (CCxP/CCxNP bits in TIMx\_CCER register, as long as the related channel is configured in output through the CCxS bits) as well as OSSR and OSSI bits can no longer be written.

11: LOCK Level 3 = LOCK Level 2 + CC Control bits (OCxM and OCxPE bits in TIMx\_CCMRx registers, as long as the related channel is configured in output through the CCxS bits) can no longer be written.

*Note: The LOCK bits can be written only once after the reset. Once the TIMx\_BDTR register has been written, their content is frozen until the next reset.*

Bits 7:0 **DTG[7:0]**: Dead-time generator setup

This bit-field defines the duration of the dead-time inserted between the complementary outputs. DT correspond to this duration.

$DTG[7:5]=0xx \Rightarrow DT=DTG[7:0]x t_{dtg}$  with  $t_{dtg}=t_{DTS}$

$DTG[7:5]=10x \Rightarrow DT=(64+DTG[5:0])x t_{dtg}$  with  $T_{dtg}=2x t_{DTS}$

$DTG[7:5]=110 \Rightarrow DT=(32+DTG[4:0])x t_{dtg}$  with  $T_{dtg}=8x t_{DTS}$

$DTG[7:5]=111 \Rightarrow DT=(32+DTG[4:0])x t_{dtg}$  with  $T_{dtg}=16x t_{DTS}$

Example if  $T_{DTS}=125\text{ns}$  (8MHz), dead-time possible values are:

0 to 15875 ns by 125 ns steps,

16  $\mu\text{s}$  to 31750 ns by 250 ns steps,

32  $\mu\text{s}$  to 63  $\mu\text{s}$  by 1  $\mu\text{s}$  steps,

64  $\mu\text{s}$  to 126  $\mu\text{s}$  by 2  $\mu\text{s}$  steps

*Note: This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx\_BDTR register).*

**20.5.17 TIM15 DMA control register (TIM15\_DCR)**

Address offset: 0x48

Reset value: 0x0000

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

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

Bits 12:8 **DBL[4:0]**: DMA burst length

This 5-bit vector defines the length of DMA transfers (the timer recognizes a burst transfer when a read or a write access is done to the TIMx\_DMAR address).

00000: 1 transfer

00001: 2 transfers

00010: 3 transfers

...

10001: 18 transfers

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

Bits 4:0 **DBA[4:0]**: DMA base address

This 5-bits vector defines the base-address for DMA transfers (when read/write access are done through the TIMx\_DMAR address). DBA is defined as an offset starting from the address of the TIMx\_CR1 register.

Example:

00000: TIMx\_CR1

00001: TIMx\_CR2

00010: TIMx\_SMCR

...

### 20.5.18 TIM15 DMA address for full transfer (TIM15\_DMAR)

Address offset: 0x4C

Reset value: 0x0000

Bits 15:0 **PMAB[15:0]**: DMA register for burst accesses

A read or write operation to the DMAR register accesses the register located at the address  
(TIMx\_CR1 address) + (DBA + DMA index) x 4

where  $\text{TIMx\_CR1}$  address is the address of the control register 1,  $\text{DBA}$  is the DMA base address configured in  $\text{TIMx\_DCR}$  register, DMA index is automatically controlled by the DMA transfer, and ranges from 0 to  $\text{DBL}$  ( $\text{DBL}$  configured in  $\text{TIMx\_DCR}$ ).

### 20.5.19 TIM15 register map

TIM15 registers are mapped as 16-bit addressable registers as described in the table below:

**Table 73. TIM15 register map and reset values**

Table 73. TIM15 register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x24	TIM15_CNT	Res.	CNT[15:0]																														
	Reset value																																
0x28	TIM15_PSC	Res.	PSC[15:0]																														
	Reset value																																
0x2C	TIM15_ARR	Res.	ARR[15:0]																														
	Reset value																																
0x30	TIM15_RCR	Res.	REP[7:0]																														
	Reset value																																
0x34	TIM15_CCR1	Res.	CCR1[15:0]																														
	Reset value																																
0x38	TIM15_CCR2	Res.	CCR2[15:0]																														
	Reset value																																
0x44	TIM15_BDTR	Res.	DT[7:0]																														
	Reset value																																
0x48	TIM15_DCR	Res.	DBL[4:0]																														
	Reset value																																
0x4C	TIM15_DMAR	Res.	DMAB[15:0]																														
	Reset value																																

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 20.6 TIM16/TIM17 registers

Refer to [Section 1.2 on page 42](#) for a list of abbreviations used in register descriptions.

### 20.6.1 TIMx control register 1 (TIMx\_CR1)(x = 16 to 17)

Address offset: 0x000

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	CKD[1:0]	ARPE	Res.	Res.	Res.	OPM	URS	UDIS	CEN	rw

Bits 15:10 Reserved, must be kept at reset value.

**Bits 9:8 CKD[1:0]: Clock division**

This bit-field indicates the division ratio between the timer clock (CK\_INT) frequency and the dead-time and sampling clock ( $t_{DTS}$ ) used by the dead-time generators and the digital filters (TIx),

- 00:  $t_{DTS}=t_{CK\_INT}$
- 01:  $t_{DTS}=2*t_{CK\_INT}$
- 10:  $t_{DTS}=4*t_{CK\_INT}$
- 11: Reserved, do not program this value

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

- 0: TIMx\_ARR register is not buffered
- 1: TIMx\_ARR register is buffered

Bits 6:4 Reserved, must be kept at reset value.

**Bit 3 OPM: One pulse mode**

- 0: Counter is not stopped at update event
- 1: Counter stops counting at the next update event (clearing the bit CEN)

**Bit 2 URS: Update request source**

This bit is set and cleared by software to select the UEV event sources.

- 0: Any of the following events generate an update interrupt or DMA request if enabled.  
These events can be:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

- 1: Only counter overflow/underflow generates an update interrupt or DMA request if enabled.

**Bit 1 UDIS: Update disable**

This bit is set and cleared by software to enable/disable UEV event generation.

- 0: UEV enabled. The Update (UEV) event is generated by one of the following events:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

Buffered registers are then loaded with their preload values.

- 1: UEV disabled. The Update event is not generated, shadow registers keep their value (ARR, PSC, CCRx). However the counter and the prescaler are reinitialized if the UG bit is set or if a hardware reset is received from the slave mode controller.

**Bit 0 CEN: Counter enable**

- 0: Counter disabled
- 1: Counter enabled

## 20.6.2 TIMx control register 2 (TIMx\_CR2)(x = 16 to 17)

Address offset: 0x04

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	OIS1N	OIS1	Res.	Res.	Res.	Res.	CCDS	CCUS	Res.	CCPC
						rw	rw					rw	rw		rw

Bits 15:10 Reserved, must be kept at reset value.

Bit 9 **OIS1N**: Output Idle state 1 (OC1N output)

- 0: OC1N=0 after a dead-time when MOE=0
- 1: OC1N=1 after a dead-time when MOE=0

*Note: This bit can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx\_BKR register).*

Bit 8 **OIS1**: Output Idle state 1 (OC1 output)

- 0: OC1=0 (after a dead-time if OC1N is implemented) when MOE=0
- 1: OC1=1 (after a dead-time if OC1N is implemented) when MOE=0

*Note: This bit can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx\_BKR register).*

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

Bit 3 **CCDS**: Capture/compare DMA selection

- 0: CCx DMA request sent when CCx event occurs
- 1: CCx DMA requests sent when update event occurs

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

- 0: When capture/compare control bits are preloaded (CCPC=1), they are updated by setting the COMG bit only.
- 1: When capture/compare control bits are preloaded (CCPC=1), they are updated by setting the COMG bit or when an rising edge occurs on TRGI.

*Note: This bit acts only on channels that have a complementary output.*

Bit 1 Reserved, must be kept at reset value.

Bit 0 **CCPC**: Capture/compare preloaded control

- 0: CCxE, CCxNE and OCxM bits are not preloaded
- 1: CCxE, CCxNE and OCxM bits are preloaded, after having been written, they are updated only when COM bit is set.

*Note: This bit acts only on channels that have a complementary output.*

## 20.6.3 TIMx DMA/interrupt enable register (TIMx\_DIER)(x = 16 to 17)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	CC1DE	UDE	BIE	Res.	COMIE	Res.	Res.	Res.	CC1IE	UIE
						rw	rw	rw		rw				rw	rw

Bits 15:10 Reserved, must be kept at reset value.

Bit 9 **CC1DE**: Capture/Compare 1 DMA request enable

- 0: CC1 DMA request disabled
- 1: CC1 DMA request enabled

Bit 8 **UDE**: Update DMA request enable

- 0: Update DMA request disabled
- 1: Update DMA request enabled

Bit 7 **BIE**: Break interrupt enable

- 0: Break interrupt disabled
- 1: Break interrupt enabled

Bit 6 Reserved, must be kept at reset value.

Bit 5 **COMIE**: COM interrupt enable

- 0: COM interrupt disabled
- 1: COM interrupt enabled

Bits 4:2 Reserved, must be kept at reset value.

Bit 1 **CC1IE**: Capture/Compare 1 interrupt enable

- 0: CC1 interrupt disabled
- 1: CC1 interrupt enabled

Bit 0 **UIE**: Update interrupt enable

- 0: Update interrupt disabled
- 1: Update interrupt enabled

#### 20.6.4 TIMx status register (TIMx\_SR)(x = 16 to 17)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	Res	Res	Res	Res	Res	CC1OF	Res	BIF	Res	COMIF	Res	Res	Res	CC1IF	UIF
						rc_w0		rc_w0		rc_w0				rc_w0	rc_w0

Bits 15:10 Reserved, must be kept at reset value.

Bit 9 **CC1OF**: Capture/Compare 1 overcapture flag

This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to '0'.

0: No overcapture has been detected

1: The counter value has been captured in TIMx\_CCR1 register while CC1IF flag was already set

Bit 8 Reserved, must be kept at reset value.

Bit 7 **BIF**: Break interrupt flag

This flag is set by hardware as soon as the break input goes active. It can be cleared by software if the break input is not active.

0: No break event occurred

1: An active level has been detected on the break input

Bit 6 Reserved, must be kept at reset value.

Bit 5 **COMIF**: COM interrupt flag

This flag is set by hardware on a COM event (once the capture/compare control bits –CCxE, CCxNE, OCxM– have been updated). It is cleared by software by writing it to '0'.

0: No COM event occurred

1: COM interrupt pending

Bits 4:2 Reserved, must be kept at reset value.

Bit 1 **CC1IF**: Capture/Compare 1 interrupt flag

**Condition: channel CC1 is configured as output:**

This flag is set by hardware when the counter matches the compare value, it is cleared by software.

0: No match.

1: The content of the counter TIMx\_CNT matches the content of the TIMx\_CCR1 register.

When the contents of TIMx\_CCR1 are greater than the contents of TIMx\_ARR, the CC1IF bit goes high on the counter overflow (in upcounting and up/down-counting modes) or underflow (in downcounting mode)

**Condition: channel CC1 is configured as input:**

This bit is set by hardware on a capture. It is cleared by software or by reading the TIMx\_CCR1 register.

0: No input capture occurred

1: The counter value has been captured in TIMx\_CCR1 register (An edge has been detected on IC1 which matches the selected polarity)

Bit 0 **UIF**: Update interrupt flag

This bit is set by hardware on an update event. It is cleared by software.

0: No update occurred.

1: Update interrupt pending. This bit is set by hardware when the registers are updated:

- At overflow regarding the repetition counter value (update if repetition counter = 0) and if the UDIS=0 in the TIMx\_CR1 register.
- When CNT is reinitialized by software using the UG bit in TIMx\_EGR register, if URS=0 and UDIS=0 in the TIMx\_CR1 register.

## 20.6.5 TIMx event generation register (TIMx\_EGR)(x = 16 to 17)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	BG	Res.	COMG	Res.	Res.	Res.	CC1G	UG							

Bits 15:8 Reserved, must be kept at reset value.

**Bit 7 BG:** Break generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action.

1: A break event is generated. MOE bit is cleared and BIF flag is set. Related interrupt or DMA transfer can occur if enabled.

Bit 6 Reserved, must be kept at reset value.

**Bit 5 COMG:** Capture/Compare control update generation

This bit can be set by software, it is automatically cleared by hardware.

0: No action

1: When the CCPC bit is set, it is possible to update the CCxE, CCxNE and OCxM bits

*Note: This bit acts only on channels that have a complementary output.*

Bits 4:2 Reserved, must be kept at reset value.

**Bit 1 CC1G:** Capture/Compare 1 generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action.

1: A capture/compare event is generated on channel 1:

**Condition: channel CC1 is configured as output:**

CC1IF flag is set, Corresponding interrupt or DMA request is sent if enabled.

**Condition: channel CC1 is configured as input:**

The current value of the counter is captured in TIMx\_CCR1 register. The CC1IF flag is set, the corresponding interrupt or DMA request is sent if enabled. The CC1OF flag is set if the CC1IF flag was already high.

**Bit 0 UG:** Update generation

This bit can be set by software, it is automatically cleared by hardware.

0: No action.

1: Reinitialize the counter and generates an update of the registers. Note that the prescaler counter is cleared too (anyway the prescaler ratio is not affected).

## 20.6.6 TIMx capture/compare mode register 1 [alternate] (TIMx\_CCMR1)(x = 16 to 17)

Address offset: 0x18

Reset value: 0x0000

The same register can be used for input capture mode (this section) or for output compare mode (next section). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OCxx describes its function when the channel is configured in output, ICxx describes its function when the channel is configured in input. So one must take care that the same bit can have a different meaning for the input stage and for the output stage.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res		IC1F[3:0]			IC1PSC[1:0]		CC1S[1:0]								

Bits 15:8 Reserved, must be kept at reset value.

#### Bits 7:4 **IC1F[3:0]**: Input capture 1 filter

This bit-field defines the frequency used to sample TI1 input and the length of the digital filter applied to TI1. The digital filter is made of an event counter in which N events are needed to validate a transition on the output:

- 0000: No filter, sampling is done at  $f_{DTS}$
- 0001:  $f_{SAMPLING} = f_{CK\_INT}$ ,  $N = 2$
- 0010:  $f_{SAMPLING} = f_{CK\_INT}$ ,  $N = 4$
- 0011:  $f_{SAMPLING} = f_{CK\_INT}$ ,  $N = 8$
- 0100:  $f_{SAMPLING} = f_{DTS} / 2$ ,  $N = 6$
- 0101:  $f_{SAMPLING} = f_{DTS} / 2$ ,  $N = 8$
- 0110:  $f_{SAMPLING} = f_{DTS} / 4$ ,  $N = 6$
- 0111:  $f_{SAMPLING} = f_{DTS} / 4$ ,  $N = 8$
- 1000:  $f_{SAMPLING} = f_{DTS} / 8$ ,  $N = 6$
- 1001:  $f_{SAMPLING} = f_{DTS} / 8$ ,  $N = 8$
- 1010:  $f_{SAMPLING} = f_{DTS} / 16$ ,  $N = 5$
- 1011:  $f_{SAMPLING} = f_{DTS} / 16$ ,  $N = 6$
- 1100:  $f_{SAMPLING} = f_{DTS} / 16$ ,  $N = 8$
- 1101:  $f_{SAMPLING} = f_{DTS} / 32$ ,  $N = 5$
- 1110:  $f_{SAMPLING} = f_{DTS} / 32$ ,  $N = 6$
- 1111:  $f_{SAMPLING} = f_{DTS} / 32$ ,  $N = 8$

*Note: Care must be taken that  $f_{DTS}$  is replaced in the formula by  $CK\_INT$  when  $ICxF[3:0] = 1, 2$  or  $3$ .*

#### Bits 3:2 **IC1PSC[1:0]**: Input capture 1 prescaler

This bit-field defines the ratio of the prescaler acting on CC1 input (IC1).

The prescaler is reset as soon as  $CC1E='0'$  (TIMx\_CCER register).

- 00: no prescaler, capture is done each time an edge is detected on the capture input.
- 01: capture is done once every 2 events
- 10: capture is done once every 4 events
- 11: capture is done once every 8 events

#### Bits 1:0 **CC1S[1:0]**: Capture/Compare 1 Selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

- 00: CC1 channel is configured as output
- 01: CC1 channel is configured as input, IC1 is mapped on TI1
- 10: CC1 channel is configured as input, IC1 is mapped on TI2
- 11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC1S bits are writable only when the channel is OFF ( $CC1E = '0'$  in TIMx\_CCER).*

### 20.6.7 **TIMx capture/compare mode register 1 [alternate] (TIMx\_CCMR1)(x = 16 to 17)**

Address offset: 0x18

Reset value: 0x0000

The same register can be used for output compare mode (this section) or for input capture mode (previous section). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OCxx describes its function when the channel is configured in output, ICxx describes its function when the channel is configured in input. So

one must take care that the same bit can have a different meaning for the input stage and for the output stage.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	OC1M[2:0]	OC1PE	OC1FE	CC1S[1:0]											
									rw	rw	rw	rw	rw	rw	rw

Bits 15:7 Reserved, must be kept at reset value.

#### Bits 6:4 OC1M[2:0]: Output Compare 1 mode

These bits define the behavior of the output reference signal OC1REF from which OC1 and OC1N are derived. OC1REF is active high whereas OC1 and OC1N active level depends on CC1P and CC1NP bits.

000: Frozen - The comparison between the output compare register TIMx\_CCR1 and the counter TIMx\_CNT has no effect on the outputs.

001: Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

011: Toggle - OC1REF toggles when TIMx\_CNT=TIMx\_CCR1.

100: Force inactive level - OC1REF is forced low.

101: Force active level - OC1REF is forced high.

110: PWM mode 1 - In upcounting, channel 1 is active as long as TIMx\_CNT<TIMx\_CCR1 else inactive. In downcounting, channel 1 is inactive (OC1REF='0') as long as TIMx\_CNT>TIMx\_CCR1 else active (OC1REF='1').

111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIMx\_CNT<TIMx\_CCR1 else active. In downcounting, channel 1 is active as long as TIMx\_CNT>TIMx\_CCR1 else inactive.

*Note:* These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx\_BDTR register) and CC1S='00' (the channel is configured in output).

*Note:* In PWM mode 1 or 2, the OCREF level changes only when the result of the comparison changes or when the output compare mode switches from "frozen" mode to "PWM" mode.

#### Bit 3 OC1PE: Output Compare 1 preload enable

0: Preload register on TIMx\_CCR1 disabled. TIMx\_CCR1 can be written at anytime, the new value is taken in account immediately.

1: Preload register on TIMx\_CCR1 enabled. Read/Write operations access the preload register. TIMx\_CCR1 preload value is loaded in the active register at each update event.

*Note:* These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx\_BDTR register) and CC1S='00' (the channel is configured in output).

#### Bit 2 OC1FE: Output Compare 1 fast enable

This bit is used to accelerate the effect of an event on the trigger in input on the CC output.

0: CC1 behaves normally depending on counter and CCR1 values even when the trigger is ON. The minimum delay to activate CC1 output when an edge occurs on the trigger input is 5 clock cycles.

1: An active edge on the trigger input acts like a compare match on CC1 output. Then, OC is set to the compare level independently of the result of the comparison. Delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. OC1FE acts only if the channel is configured in PWM1 or PWM2 mode.

Bits 1:0 **CC1S[1:0]**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output

01: CC1 channel is configured as input, IC1 is mapped on TI1

10: CC1 channel is configured as input, IC1 is mapped on TI2

11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC1S bits are writable only when the channel is OFF (CC1E = '0' in TIMx\_CCER).*

**20.6.8 TIMx capture/compare enable register (TIMx\_CCER)(x = 16 to 17)**

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CC1NP	CC1NE	CC1P	CC1E											

Bits 15:4 Reserved, must be kept at reset value.

Bit 3 **CC1NP**: Capture/Compare 1 complementary output polarity

0: OC1N active high

1: OC1N active low

*Note: This bit is not writable as soon as LOCK level 2 or 3 has been programmed (LOCK bits in TIMx\_BDTR register) and CC1S="00" (the channel is configured in output).*

Bit 2 **CC1NE**: Capture/Compare 1 complementary output enable

0: Off - OC1N is not active. OC1N level is then function of MOE, OSS1, OSSR, OIS1, OIS1N and CC1E bits.

1: On - OC1N signal is output on the corresponding output pin depending on MOE, OSS1, OSSR, OIS1, OIS1N and CC1E bits.

Bit 1 **CC1P**: Capture/Compare 1 output polarity**Condition: CC1 channel configured as output**

0: OC1 active high

1: OC1 active low

**Condition: CC1 channel configured as input**

The CC1NP/CC1P bits select the polarity of TI1FP1 and TI2FP1 for capture operation.

00: Non-inverted/rising edge: circuit is sensitive to TIxFP1's rising edge TIxFP1 is not inverted.

01: Inverted/falling edge: circuit is sensitive to TIxFP1's falling edge, TIxFP1 is inverted.

10: Reserved, do not use this configuration.

11: Non-inverted/both edges: circuit is sensitive to both the rising and falling edges of TIxFP1, TIxFP1 is not inverted.

*Note: This bit is not writable as soon as LOCK level 2 or 3 has been programmed (LOCK bits in TIMx\_BDTR register)*

Bit 0 **CC1E**: Capture/Compare 1 output enable

**Condition: CC1 channel configured as output**

0: Off - OC1 is not active. OC1 level is then function of MOE, OSS1, OSSR, OIS1, OIS1N and CC1NE bits.

1: On - OC1 signal is output on the corresponding output pin depending on MOE, OSS1, OSSR, OIS1, OIS1N and CC1NE bits.

**Condition: CC1 channel configured as input**

This bit determines if a capture of the counter value can actually be done into the input capture/compare register 1 (TIMx\_CCR1) or not.

0: Capture disabled

1: Capture enabled

**Table 74. Output control bits for complementary OCx and OCxN channels with break feature**

Control bits					Output states <sup>(1)</sup>	
MOE bit	OSSI bit	OSSR bit	CCxE bit	CCxNE bit	OCx output state	OCxN output state
1	X	0	0	0	Output Disabled (not driven by the timer) OCx=0, OCx_EN=0	Output Disabled (not driven by the timer) OCxN=0, OCxN_EN=0
		0	0	1	Output Disabled (not driven by the timer) OCx=0, OCx_EN=0	OCxREF + Polarity OCxN=OCxREF xor CCxNP, OCxN_EN=1
		0	1	0	OCxREF + Polarity OCx=OCxREF xor CCxP, OCx_EN=1	Output Disabled (not driven by the timer) OCxN=0, OCxN_EN=0
		0	1	1	OCREF + Polarity + dead-time OCx_EN=1	Complementary to OCREF (not OCREF) + Polarity + dead-time OCxN_EN=1
		1	0	0	Output Disabled (not driven by the timer) OCx=CCxP, OCx_EN=0	Output Disabled (not driven by the timer) OCxN=CCxNP, OCxN_EN=0
		1	0	1	Off-State (output enabled with inactive state) OCx=CCxP, OCx_EN=1	OCxREF + Polarity OCxN=OCxREF xor CCxNP, OCxN_EN=1
		1	1	0	OCxREF + Polarity OCx=OCxREF xor CCxP, OCx_EN=1	Off-State (output enabled with inactive state) OCxN=CCxNP, OCxN_EN=1
		1	1	1	OCREF + Polarity + dead-time OCx_EN=1	Complementary to OCREF (not OCREF) + Polarity + dead-time OCxN_EN=1
0	X	0	0	0	Output Disabled (not driven by the timer)	
		0	0	1	Asynchronously: OCx=CCxP, OCx_EN=0, OCxN=CCxNP, OCxN_EN=0	
		0	1	0	Then if the clock is present: OCx=OISx and OCxN=OISxN after a dead-time, assuming that OISx and OISxN do not correspond to OCX and OCxN both in active state.	
		0	1	1		
		1	0	0		
		1	0	1	Off-State (output enabled with inactive state)	
		1	1	0	Asynchronously: OCx=CCxP, OCx_EN=1, OCxN=CCxNP, OCxN_EN=1	
		1	1	1	Then if the clock is present: OCx=OISx and OCxN=OISxN after a dead-time, assuming that OISx and OISxN do not correspond to OCX and OCxN both in active state	

- When both outputs of a channel are not used (CCxE = CCxNE = 0), the OISx, OISxN, CCxP and CCxNP bits must be kept cleared.

**Note:** The state of the external I/O pins connected to the complementary OCx and OCxN channels depends on the OCx and OCxN channel state and the GPIO and AFIO registers.

## 20.6.9 TIMx counter (TIMx\_CNT)(x = 16 to 17)

Address offset: 0x24

Reset value: 0x0000

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

Bits 15:0 **CNT[15:0]**: Counter value

## 20.6.10 TIMx prescaler (TIMx\_PSC)(x = 16 to 17)

Address offset: 0x28

Reset value: 0x0000

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

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency (CK\_CNT) is equal to  $f_{CK\_PSC} / (PSC[15:0] + 1)$ .

PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx\_EGR register or through trigger controller when configured in “reset mode”).

## 20.6.11 TIMx auto-reload register (TIMx\_ARR)(x = 16 to 17)

Address offset: 0x2C

Reset value: 0xFFFF

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

Bits 15:0 **ARR[15:0]**: Auto-reload value

ARR is the value to be loaded in the actual auto-reload register.

Refer to the [Section 19.3.1: Time-base unit on page 472](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

### 20.6.12 TIMx repetition counter register (TIMx\_RCR)(x = 16 to 17)

Address offset: 0x30

Reset value: 0x0000

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

Bits 15:8 Reserved, must be kept at reset value.

Bits 7:0 **REP[7:0]**: Repetition counter value

These bits allow the user to set-up the update rate of the compare registers (i.e. periodic transfers from preload to active registers) when preload registers are enable, as well as the update interrupt generation rate, if this interrupt is enable.

Each time the REP\_CNT related downcounter reaches zero, an update event is generated and it restarts counting from REP value. As REP\_CNT is reloaded with REP value only at the repetition update event U\_RC, any write to the TIMx\_RCR register is not taken in account until the next repetition update event.

It means in PWM mode (REP+1) corresponds to the number of PWM periods in edge-aligned mode.

### 20.6.13 TIMx capture/compare register 1 (TIMx\_CCR1)(x = 16 to 17)

Address offset: 0x34

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR1[15:0]**: Capture/Compare 1 value

**Condition: channel CC1 is configured as output**

CCR1 is the value to be loaded in the actual capture/compare 1 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR1 register (bit OC1PE). Else the preload value is copied in the active capture/compare 1 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signaled on OC1 output.

**Condition: channel CC1 is configured as input:**

CCR1 is the counter value transferred by the last input capture 1 event (IC1).

### 20.6.14 TIMx break and dead-time register (TIMx\_BDTR)(x = 16 to 17)

Address offset: 0x44

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK[1:0]		DTG[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

**Note:** As the bits AOE, BKP, BKE, OSSl, OSSR and DTG[7:0] can be write-locked depending on the LOCK configuration, it can be necessary to configure all of them during the first write access to the TIMx\_BDTR register.

**Bit 15 MOE:** Main output enable

This bit is cleared asynchronously by hardware as soon as the break input is active. It is set by software or automatically depending on the AOE bit. It is acting only on the channels which are configured in output.

0: OC and OCN outputs are disabled or forced to idle state

1: OC and OCN outputs are enabled if their respective enable bits are set (CCxE, CCxNE in TIMx\_CCER register)

See OC/OCN enable description for more details ([Section 20.5.9: TIM15 capture/compare enable register \(TIM15\\_CCER\) on page 533](#)).

**Bit 14 AOE:** Automatic output enable

0: MOE can be set only by software

1: MOE can be set by software or automatically at the next update event (if the break input is not be active)

**Note:** This bit can not be modified as long as LOCK level 1 has been programmed (LOCK bits in TIMx\_BDTR register).

**Bit 13 BKP:** Break polarity

0: Break input BRK is active low

1: Break input BRK is active high

**Note:** This bit can not be modified as long as LOCK level 1 has been programmed (LOCK bits in TIMx\_BDTR register).

**Note:** Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.

**Bit 12 BKE:** Break enable

0: Break inputs (BRK and CCS clock failure event) disabled

1: Break inputs (BRK and CCS clock failure event) enabled

**Note:** This bit cannot be modified when LOCK level 1 has been programmed (LOCK bits in TIMx\_BDTR register).

**Note:** Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.

**Bit 11 OSSR:** Off-state selection for Run mode

This bit is used when MOE=1 on channels having a complementary output which are configured as outputs. OSSR is not implemented if no complementary output is implemented in the timer.

See OC/OCN enable description for more details ([Section 20.5.9: TIM15 capture/compare enable register \(TIM15\\_CCER\) on page 533](#)).

0: When inactive, OC/OCN outputs are disabled (OC/OCN enable output signal=0)

1: When inactive, OC/OCN outputs are enabled with their inactive level as soon as CCxE=1 or CCxNE=1. Then, OC/OCN enable output signal=1

**Note:** This bit can not be modified as soon as the LOCK level 2 has been programmed (LOCK bits in TIMx\_BDTR register).

Bit 10 **OSSI**: Off-state selection for Idle mode

This bit is used when MOE=0 on channels configured as outputs.

See OC/OCN enable description for more details ([Section 20.5.9: TIM15 capture/compare enable register \(TIM15\\_CCER\) on page 533](#)).

0: When inactive, OC/OCN outputs are disabled (OC/OCN enable output signal=0)

1: When inactive, OC/OCN outputs are forced first with their idle level as soon as CCxE=1 or CCxNE=1. OC/OCN enable output signal=1)

*Note:* This bit can not be modified as soon as the LOCK level 2 has been programmed (LOCK bits in TIMx\_BDTR register).

Bits 9:8 **LOCK[1:0]**: Lock configuration

These bits offer a write protection against software errors.

00: LOCK OFF - No bit is write protected

01: LOCK Level 1 = DTG bits in TIMx\_BDTR register, OISx and OISxN bits in TIMx\_CR2 register and BKE/BKP/AOE bits in TIMx\_BDTR register can no longer be written.

10: LOCK Level 2 = LOCK Level 1 + CC Polarity bits (CCxP/CCxNP bits in TIMx\_CCER register, as long as the related channel is configured in output through the CCxS bits) as well as OSSR and OSSI bits can no longer be written.

11: LOCK Level 3 = LOCK Level 2 + CC Control bits (OCxM and OCxPE bits in TIMx\_CCMRx registers, as long as the related channel is configured in output through the CCxS bits) can no longer be written.

*Note:* The LOCK bits can be written only once after the reset. Once the TIMx\_BDTR register has been written, their content is frozen until the next reset.

Bits 7:0 **DTG[7:0]**: Dead-time generator setup

This bit-field defines the duration of the dead-time inserted between the complementary outputs. DT correspond to this duration.

DTG[7:5]=0xx => DT=DTG[7:0]x t<sub>dtg</sub> with t<sub>dtg</sub>=t<sub>DTS</sub>

DTG[7:5]=10x => DT=(64+DTG[5:0])x t<sub>dtg</sub> with T<sub>dtg</sub>=2x t<sub>DTS</sub>

DTG[7:5]=110 => DT=(32+DTG[4:0])x t<sub>dtg</sub> with T<sub>dtg</sub>=8x t<sub>DTS</sub>

DTG[7:5]=111 => DT=(32+DTG[4:0])x t<sub>dtg</sub> with T<sub>dtg</sub>=16x t<sub>DTS</sub>

Example if T<sub>DTS</sub>=125ns (8MHz), dead-time possible values are:

0 to 15875 ns by 125 ns steps,

16 µs to 31750 ns by 250 ns steps,

32 µs to 63 µs by 1 µs steps,

64 µs to 126 µs by 2 µs steps

*Note:* This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx\_BDTR register).

## 20.6.15 TIMx DMA control register (TIMx\_DCR)(x = 16 to 17)

Address offset: 0x48

Reset value: 0x0000

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

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

Bits 12:8 **DBL[4:0]**: DMA burst length

This 5-bit vector defines the length of DMA transfers (the timer recognizes a burst transfer when a read or a write access is done to the TIMx\_DMAR address), i.e. the number of transfers. Transfers can be in half-words or in bytes (see example below).

- 00000: 1 transfer
- 00001: 2 transfers
- 00010: 3 transfers
- ...
- 10001: 18 transfers.

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

Bits 4:0 **DBA[4:0]**: DMA base address

This 5-bits vector defines the base-address for DMA transfers (when read/write access are done through the TIMx\_DMAR address). DBA is defined as an offset starting from the address of the TIMx\_CR1 register.

Example:

- 00000: TIMx\_CR1
- 00001: TIMx\_CR2
- 00010: TIMx\_SMCR
- ...

**Example:** Let us consider the following transfer: DBL = 7 transfers and DBA = TIMx\_CR1. In this case the transfer is done to/from 7 registers starting from the TIMx\_CR1 address.

## 20.6.16 TIMx DMA address for full transfer (TIMx\_DMAR)(x = 16 to 17)

Address offset: 0x4C

Reset value: 0x0000

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

Bits 15:0 **DMAB[15:0]**: DMA register for burst accesses

A read or write access to the DMAR register accesses the register located at the address: "(TIMx\_CR1 address) + DBA + (DMA index)" in which:

TIMx\_CR1 address is the address of the control register 1, DBA is the DMA base address configured in TIMx\_DCR register, DMA index is the offset automatically controlled by the DMA transfer, depending on the length of the transfer DBL in the TIMx\_DCR register.

### Example of how to use the DMA burst feature

In this example the timer DMA burst feature is used to update the contents of the CCRx registers (x = 2, 3, 4) with the DMA transferring half words into the CCRx registers.

This is done in the following steps:

1. Configure the corresponding DMA channel as follows:
  - DMA channel peripheral address is the DMAR register address
  - DMA channel memory address is the address of the buffer in the RAM containing the data to be transferred by DMA into CCRx registers.
  - Number of data to transfer = 3 (See note below).
  - Circular mode disabled.
2. Configure the DCR register by configuring the DBA and DBL bit fields as follows:  
DBL = 3 transfers, DBA = 0xE.
3. Enable the TIMx update DMA request (set the UDE bit in the DIER register).
4. Enable TIMx
5. Enable the DMA channel

**Note:**

*This example is for the case where every CCRx register to be updated once. If every CCRx register is to be updated twice for example, the number of data to transfer should be 6. Let us take the example of a buffer in the RAM containing data1, data2, data3, data4, data5 and data6. The data is transferred to the CCRx registers as follows: on the first update DMA request, data1 is transferred to CCR2, data2 is transferred to CCR3, data3 is transferred to CCR4 and on the second update DMA request, data4 is transferred to CCR2, data5 is transferred to CCR3 and data6 is transferred to CCR4.*

### 20.6.17 TIM16/TIM17 register map

TIM16 and TIM17 registers are mapped as 16-bit addressable registers as described in the table below:

**Table 75. TIM16/TIM17 register map and reset values**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5
0x00	TIM16_CR1 and TIM17_CR1	Res.																										
	Reset value																											
0x04	TIM16_CR2 and TIM17_CR2	Res.																										
	Reset value																											
0x0C	TIM16_DIER and TIM17_DIER	Res.																										
	Reset value																											
0x10	TIM16_SR and TIM17_SR	Res.																										
	Reset value																											
0x14	TIM16_EGR and TIM17_EGR	Res.																										
	Reset value																											

Table 75. TIM16/TIM17 register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x18	TIM16_CCMR1 and TIM17_CCMR1 Output compare mode	Res.																																
	Reset value																																	
	TIM16_CCMR1 and TIM17_CCMR1 Input capture mode																																	
	Reset value																																	
0x20	TIM16_CCER and TIM17_CCER	Res.																																
	Reset value																																	
0x24	TIM16_CNT and TIM17_CNT	Res.																																
	Reset value																																	
0x28	TIM16_PSC and TIM17_PSC	Res.																																
	Reset value																																	
0x2C	TIM16_ARR and TIM17_ARR	Res.																																
	Reset value																																	
0x30	TIM16_RCR and TIM17_RCR	Res.																																
	Reset value																																	
0x34	TIM16_CCR1 and TIM17_CCR1	Res.																																
	Reset value																																	
0x44	TIM16_BDTR and TIM17_BDTR	Res.																																
	Reset value																																	
0x48	TIM16_DCR and TIM17_DCR	Res.																																
	Reset value																																	
0x4C	TIM16_DMAR and TIM17_DMAR	Res.																																
	Reset value																																	

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 21 Basic timer (TIM6/TIM7)

This section applies to STM32F05x, STM32F07x and STM32F09x devices only. TIM7 is available only on STM32F07x and STM32F09x devices.

### 21.1 TIM6/TIM7 introduction

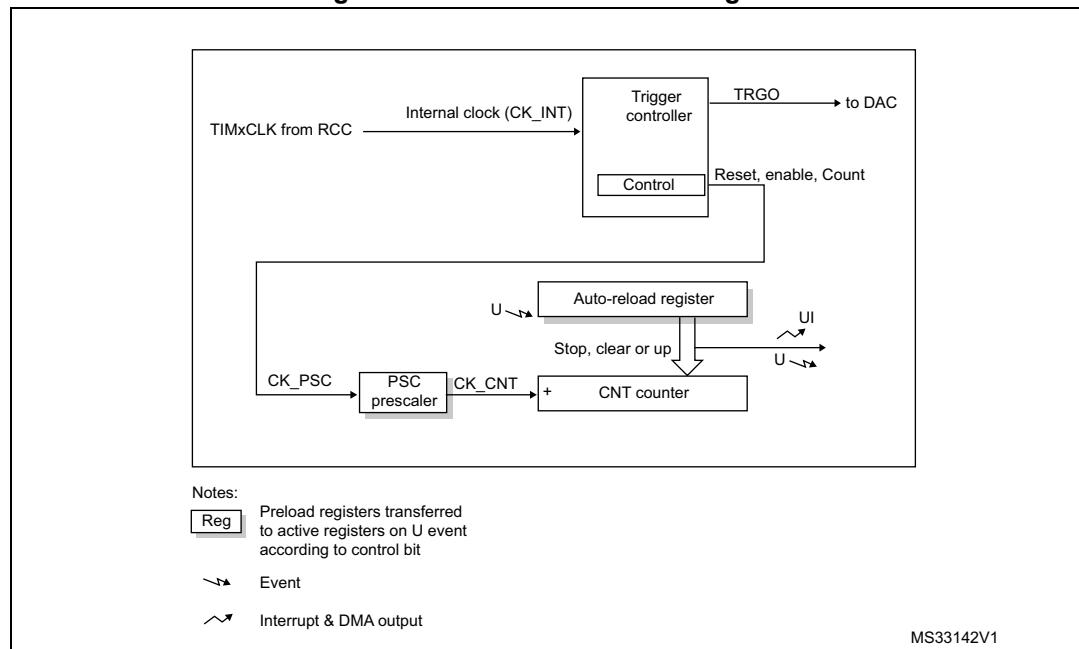
The basic timer TIM6 consists of a 16-bit auto-reload counter driven by a programmable prescaler.

It may be used as a generic timer for time-base generation but it is also specifically used to drive the digital-to-analog converter (DAC). In fact, TIM6 is internally connected to the DAC and is able to drive it through its trigger outputs.

### 21.2 TIM6/TIM7 main features

- 16-bit auto-reload upcounter
- 16-bit programmable prescaler used to divide (also “on the fly”) the counter clock frequency by any factor between 1 and 65535
- Synchronization circuit to trigger the DAC
- Interrupt/DMA generation on the update event: counter overflow

**Figure 201. Basic timer block diagram**



## 21.3 TIM6/TIM7 functional description

### 21.3.1 Time-base unit

The main block of the programmable timer is a 16-bit upcounter with its related auto-reload register. The counter clock can be divided by a prescaler.

The counter, the auto-reload register and the prescaler register can be written or read by software. This is true even when the counter is running.

The time-base unit includes:

- Counter Register (TIMx\_CNT)
- Prescaler Register (TIMx\_PSC)
- Auto-Reload Register (TIMx\_ARR)

The auto-reload register is preloaded. The preload register is accessed each time an attempt is made to write or read the auto-reload register. The contents of the preload register are transferred into the shadow register permanently or at each update event UEV, depending on the auto-reload preload enable bit (ARPE) in the TIMx\_CR1 register. The update event is sent when the counter reaches the overflow value and if the UDIS bit equals 0 in the TIMx\_CR1 register. It can also be generated by software. The generation of the update event is described in detail for each configuration.

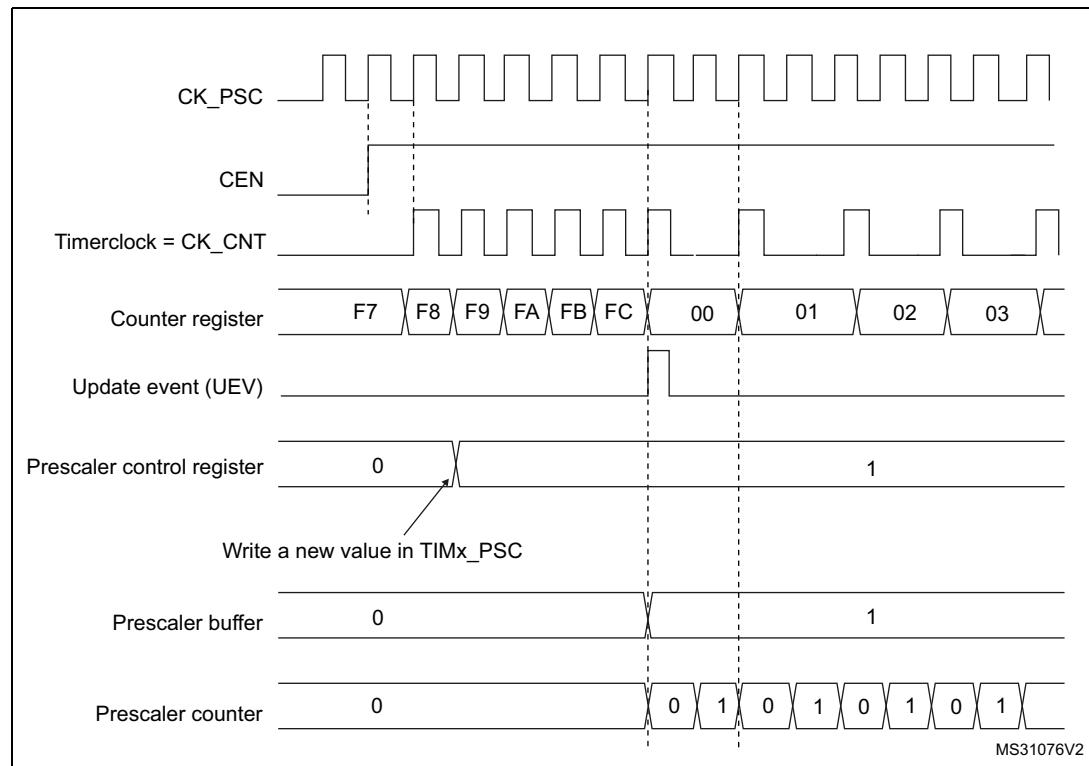
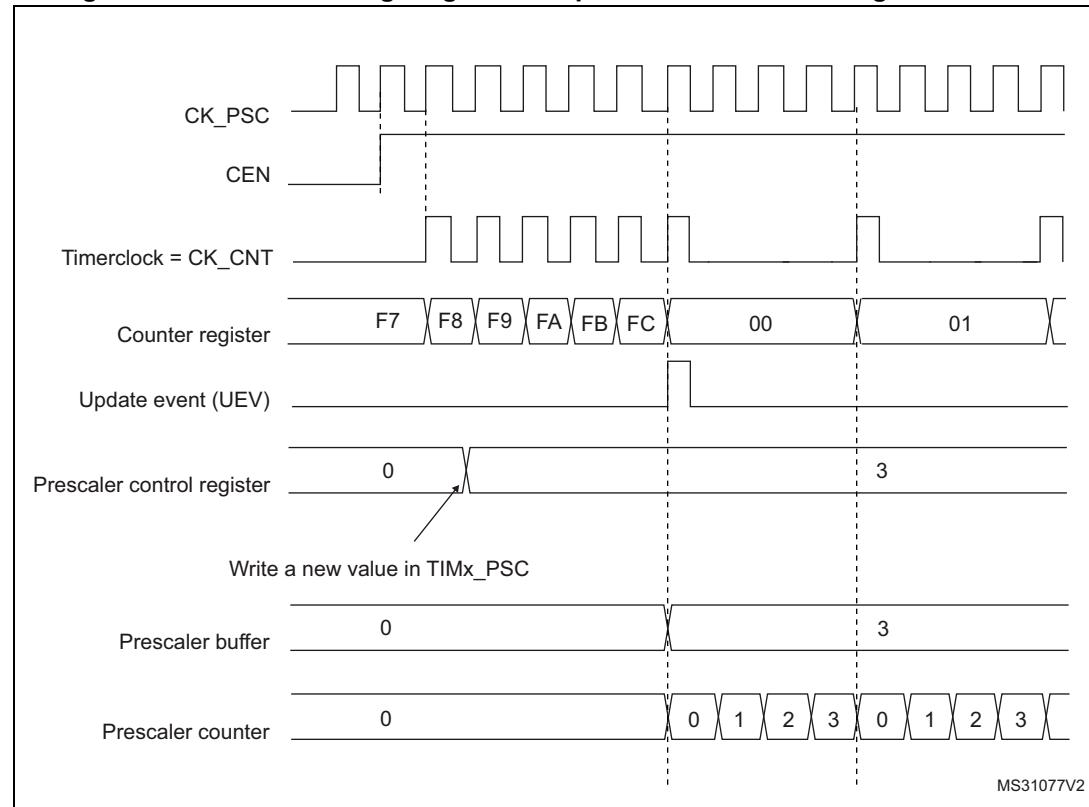
The counter is clocked by the prescaler output CK\_CNT, which is enabled only when the counter enable bit (CEN) in the TIMx\_CR1 register is set.

Note that the actual counter enable signal CNT\_EN is set 1 clock cycle after CEN.

#### Prescaler description

The prescaler can divide the counter clock frequency by any factor between 1 and 65536. It is based on a 16-bit counter controlled through a 16-bit register (in the TIMx\_PSC register). It can be changed on the fly as the TIMx\_PSC control register is buffered. The new prescaler ratio is taken into account at the next update event.

*Figure 202* and *Figure 203* give some examples of the counter behavior when the prescaler ratio is changed on the fly.

**Figure 202. Counter timing diagram with prescaler division change from 1 to 2****Figure 203. Counter timing diagram with prescaler division change from 1 to 4**

### 21.3.2 Counter modes

The counter counts from 0 to the auto-reload value (contents of the TIMx\_ARR register), then restarts from 0 and generates a counter overflow event.

An update event can be generated at each counter overflow or by setting the UG bit in the TIMx\_EGR register (by software or by using the slave mode controller).

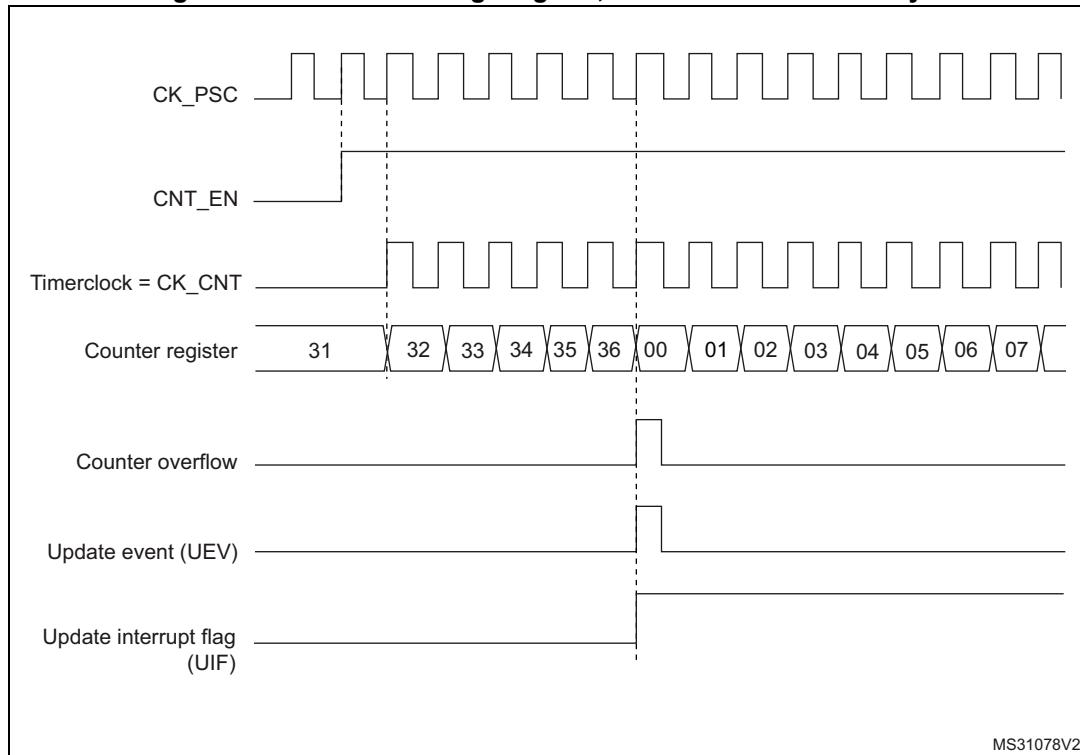
The UEV event can be disabled by software by setting the UDIS bit in the TIMx\_CR1 register. This avoids updating the shadow registers while writing new values into the preload registers. In this way, no update event occurs until the UDIS bit has been written to 0, however, the counter and the prescaler counter both restart from 0 (but the prescale rate does not change). In addition, if the URS (update request selection) bit in the TIMx\_CR1 register is set, setting the UG bit generates an update event UEV, but the UIF flag is not set (so no interrupt or DMA request is sent).

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

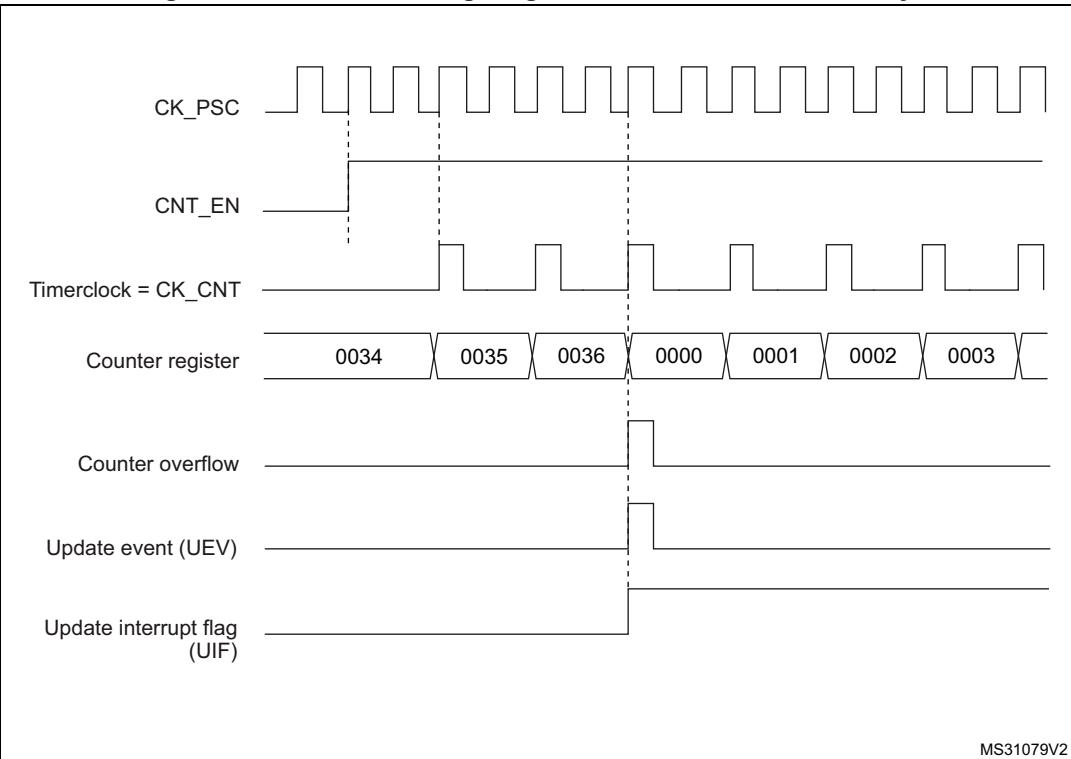
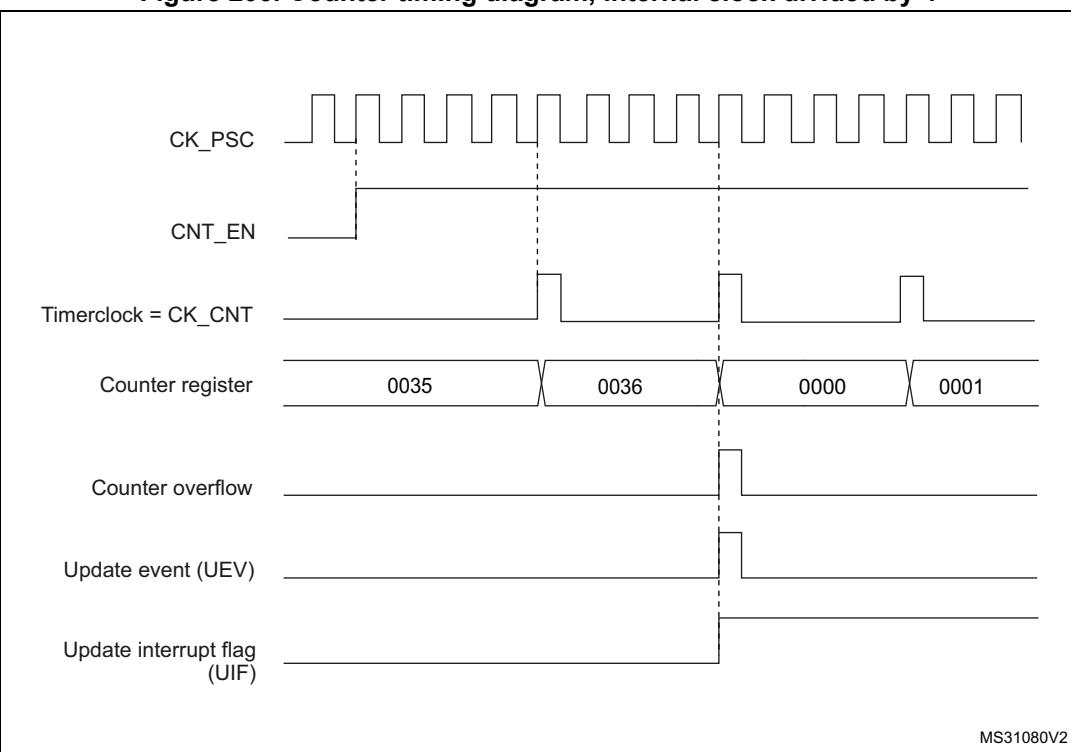
- The buffer of the prescaler is reloaded with the preload value (contents of the TIMx\_PSC register)
- The auto-reload shadow register is updated with the preload value (TIMx\_ARR)

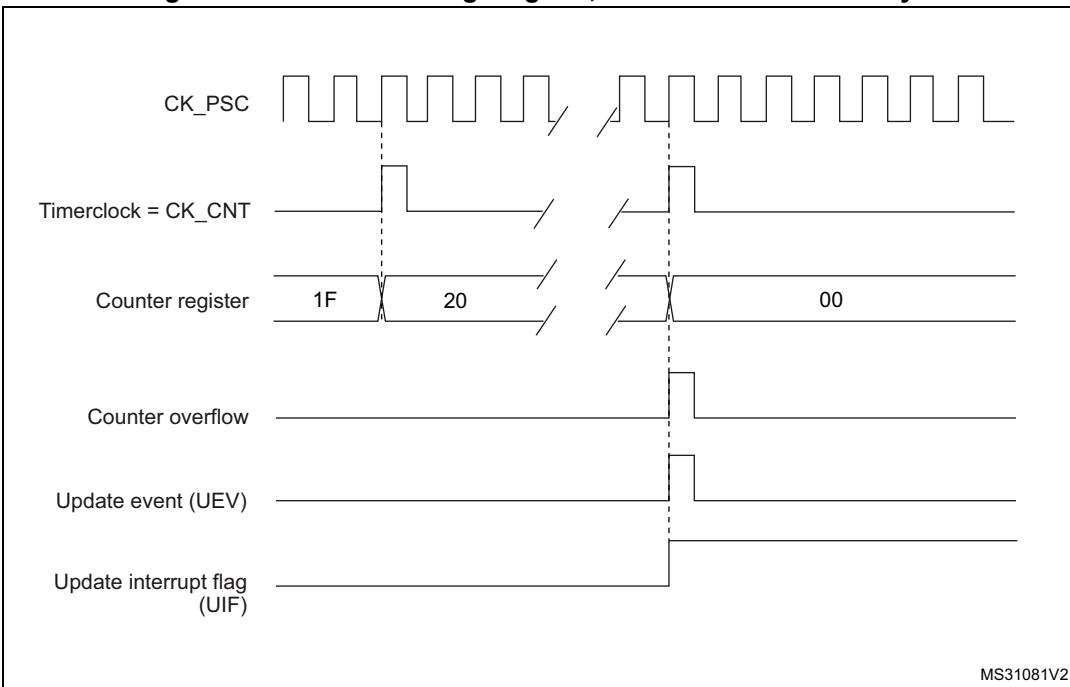
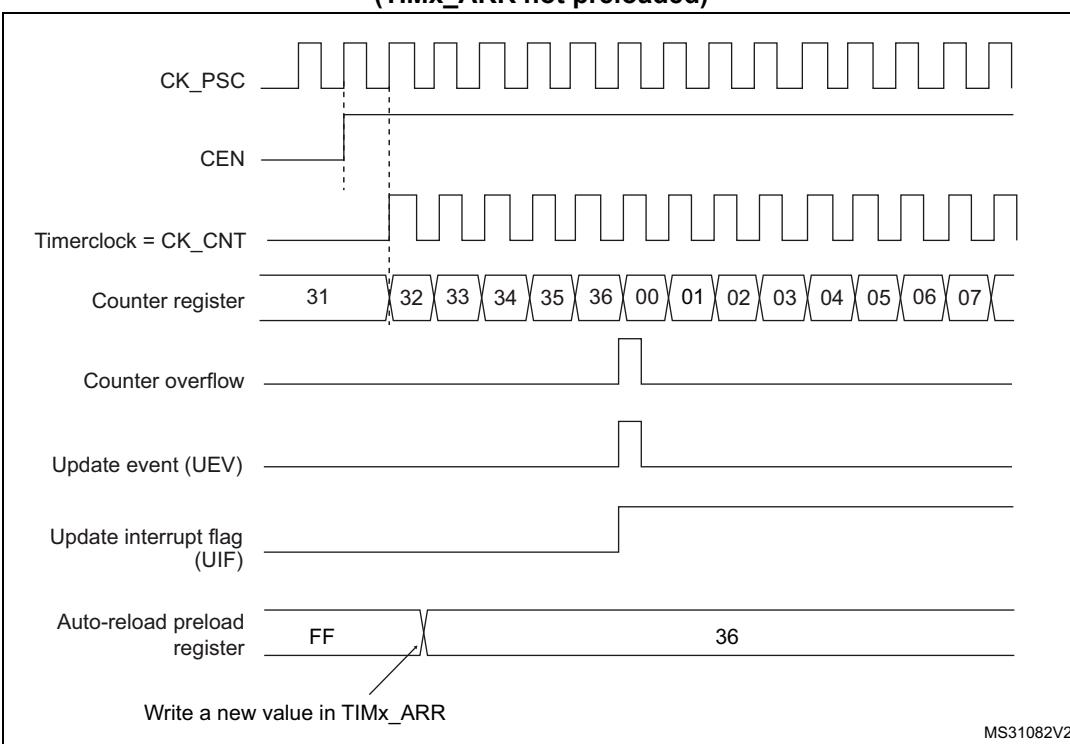
The following figures show some examples of the counter behavior for different clock frequencies when TIMx\_ARR = 0x36.

**Figure 204. Counter timing diagram, internal clock divided by 1**

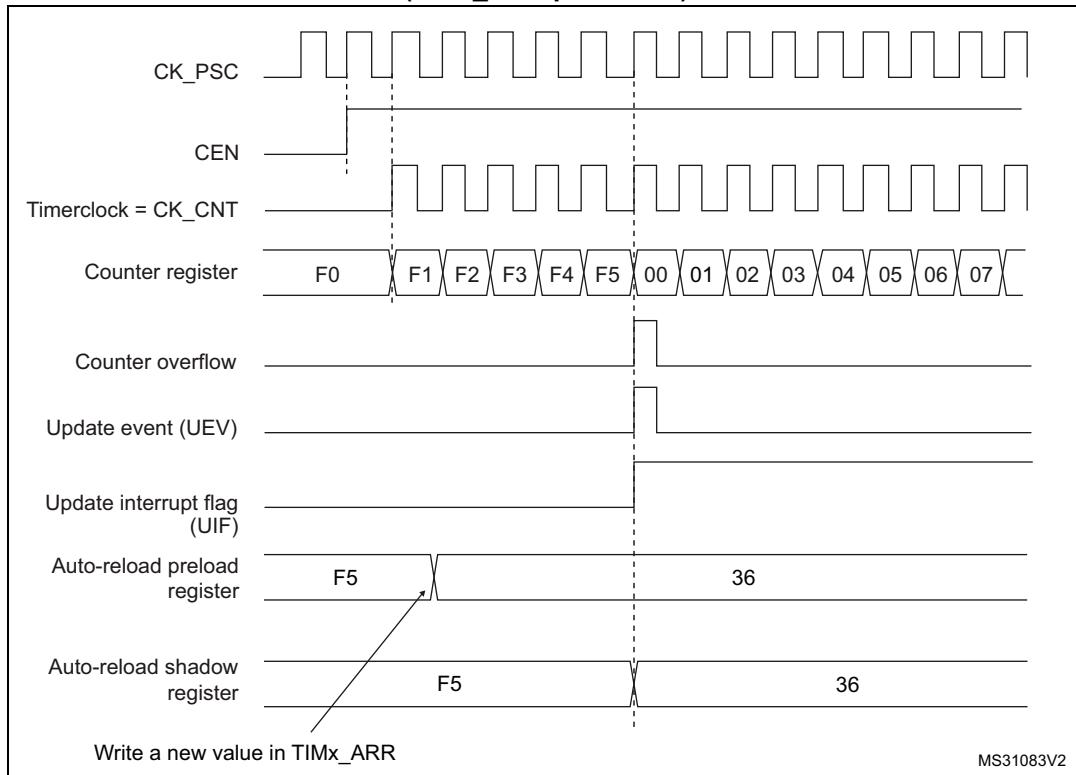


MS31078V2

**Figure 205. Counter timing diagram, internal clock divided by 2****Figure 206. Counter timing diagram, internal clock divided by 4**

**Figure 207. Counter timing diagram, internal clock divided by N****Figure 208. Counter timing diagram, update event when ARPE = 0  
(TIMx\_ARR not preloaded)**

**Figure 209. Counter timing diagram, update event when ARPE=1  
(TIMx\_ARR preloaded)**



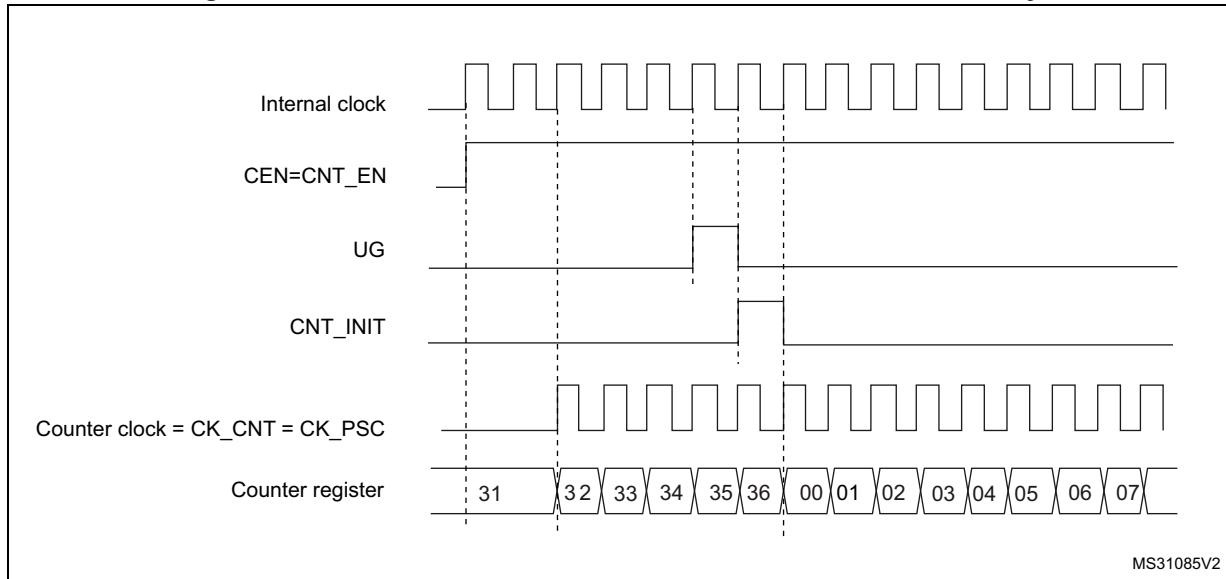
### 21.3.3 Clock source

The counter clock is provided by the Internal clock (CK\_INT) source.

The CEN (in the TIMx\_CR1 register) and UG bits (in the TIMx\_EGR register) are actual control bits and can be changed only by software (except for UG that remains cleared automatically). As soon as the CEN bit is written to 1, the prescaler is clocked by the internal clock CK\_INT.

*Figure 210* shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

**Figure 210. Control circuit in normal mode, internal clock divided by 1**



### 21.3.4 Debug mode

When the microcontroller enters the debug mode (Cortex™-M0 core - halted), the TIMx counter either continues to work normally or stops, depending on the DBG\_TIMx\_STOP configuration bit in the DBG module.

## 21.4 TIM6/TIM7 registers

Refer to [Section 1.2 on page 42](#) for a list of abbreviations used in register descriptions.

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

### 21.4.1 TIM6/TIM7 control register 1 (TIMx\_CR1)

Address offset: 0x000

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	ARPE	Res.	Res.	Res.	OPM	URS	UDIS	CEN							

Bits 15:8 Reserved, always read as 0.

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

- 0: TIMx\_ARR register is not buffered.
- 1: TIMx\_ARR register is buffered.

Bits 6:4 Reserved, always read as 0.

Bit 3 **OPM**: One-pulse mode

- 0: Counter is not stopped at update event
- 1: Counter stops counting at the next update event (clearing the CEN bit).

Bit 2 **URS**: Update request source

This bit is set and cleared by software to select the UEV event sources.

- 0: Any of the following events generates an update interrupt or DMA request if enabled.  
These events can be:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

- 1: Only counter overflow/underflow generates an update interrupt or DMA request if enabled.

Bit 1 **UDIS**: Update disable

This bit is set and cleared by software to enable/disable UEV event generation.

- 0: UEV enabled. The Update (UEV) event is generated by one of the following events:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

Buffered registers are then loaded with their preload values.

- 1: UEV disabled. The Update event is not generated, shadow registers keep their value (ARR, PSC). However the counter and the prescaler are reinitialized if the UG bit is set or if a hardware reset is received from the slave mode controller.

Bit 0 **CEN**: Counter enable

- 0: Counter disabled
- 1: Counter enabled

*Note: Gated mode can work only if the CEN bit has been previously set by software.  
However trigger mode can set the CEN bit automatically by hardware.*

CEN is cleared automatically in one-pulse mode, when an update event occurs.

### 21.4.2 TIM6/TIM7 control register 2 (TIMx\_CR2)

Address offset: 0x04

Reset value: 0x0000

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

Bits 15:7 Reserved, must be kept at reset value.

Bits 6:4 **MMS**: Master mode selection

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

000: **Reset** - the UG bit from the TIMx\_EGR register is used as a trigger output (TRGO). If reset is generated by the trigger input (slave mode controller configured in reset mode) then the signal on TRGO is delayed compared to the actual reset.

001: **Enable** - the Counter enable signal, CNT\_EN, is used as a trigger output (TRGO). It is useful to start several timers at the same time or to control a window in which a slave timer is enabled. The Counter Enable signal is generated by a logic OR between CEN control bit and the trigger input when configured in gated mode.

When the Counter Enable signal is controlled by the trigger input, there is a delay on TRGO, except if the master/slave mode is selected (see the MSM bit description in the TIMx\_SMCR register).

010: **Update** - The update event is selected as a trigger output (TRGO). For instance a master timer can then be used as a prescaler for a slave timer.

Bits 3:0 Reserved, always read as 0.

### 21.4.3 TIM6/TIM7 DMA/Interrupt enable register (TIMx\_DIER)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	UDE	Res.	UIE												
							rw								rw

Bits 15:9 Reserved, must be kept at reset value.

Bit 8 **UDE**: Update DMA request enable

- 0: Update DMA request disabled.
- 1: Update DMA request enabled.

Bits 7:1 Reserved, must be kept at reset value.

Bit 0 **UIE**: Update interrupt enable

- 0: Update interrupt disabled.
- 1: Update interrupt enabled.

#### 21.4.4 TIM6/TIM7 status register (TIMx\_SR)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	UIF														
															rc_w0

Bits 15:1 Reserved, must be kept at reset value.

Bit 0 **UIF**: Update interrupt flag

This bit is set by hardware on an update event. It is cleared by software.

0: No update occurred.

1: Update interrupt pending. This bit is set by hardware when the registers are updated:

- At overflow or underflow regarding the repetition counter value and if UDIS = 0 in the TIMx\_CR1 register.

- When CNT is reinitialized by software using the UG bit in the TIMx\_EGR register, if URS = 0 and UDIS = 0 in the TIMx\_CR1 register.

#### 21.4.5 TIM6/TIM7 event generation register (TIMx\_EGR)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	UG														
															w

Bits 15:1 Reserved, must be kept at reset value.

Bit 0 **UG**: Update generation

This bit can be set by software, it is automatically cleared by hardware.

0: No action.

1: Re-initializes the timer counter and generates an update of the registers. Note that the prescaler counter is cleared too (but the prescaler ratio is not affected).

#### 21.4.6 TIM6/TIM7 counter (TIMx\_CNT)

Address offset: 0x24

Reset value: 0x0000

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

Bits 15:0      **CNT[15:0]**: Counter value

### 21.4.7 TIM6/TIM7 prescaler (TIMx\_PSC)

Address offset: 0x28

Reset value: 0x0000

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

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK\_CNT is equal to  $f_{CK\_PSC} / (PSC[15:0] + 1)$ .

PSC contains the value to be loaded into the active prescaler register at each update event.

### 21.4.8 TIM6/TIM7 auto-reload register (TIMx\_ARR)

Address offset: 0x2C

Reset value: 0xFFFF

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

Bits 15:0 **ARR[15:0]**: Auto-reload value

ARR is the value to be loaded into the actual auto-reload register.

Refer to [Section 21.3.1: Time-base unit on page 561](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

### 21.4.9 TIM6/TIM7 register map

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

**Table 76. TIM6/TIM7 register map and reset values**

Offset	Register	Reset value
0x00	<b>TIMx_CR1</b>	31
	Reset value	Res.
0x04	<b>TIMx_CR2</b>	30
	Reset value	Res.
0x0C	<b>TIMx_DIER</b>	29
	Reset value	Res.
0x10	<b>TIMx_SR</b>	28
	Reset value	Res.
0x14	<b>TIMx_EGR</b>	27
	Reset value	Res.
0x24	<b>TIMx_CNT</b>	26
	Reset value	Res.
0x28	<b>TIMx_PSC</b>	25
	Reset value	Res.
0x2C	<b>TIMx_ARR</b>	24
	Reset value	Res.
CNT[15:0]		
PSC[15:0]		
ARR[15:0]		
		MMS[2:0]
		0
		16
		15
		12
		11
		10
		9
		8
		7
		6
		5
		4
		3
		2
		1
		0

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 22 Infrared interface (IRTIM)

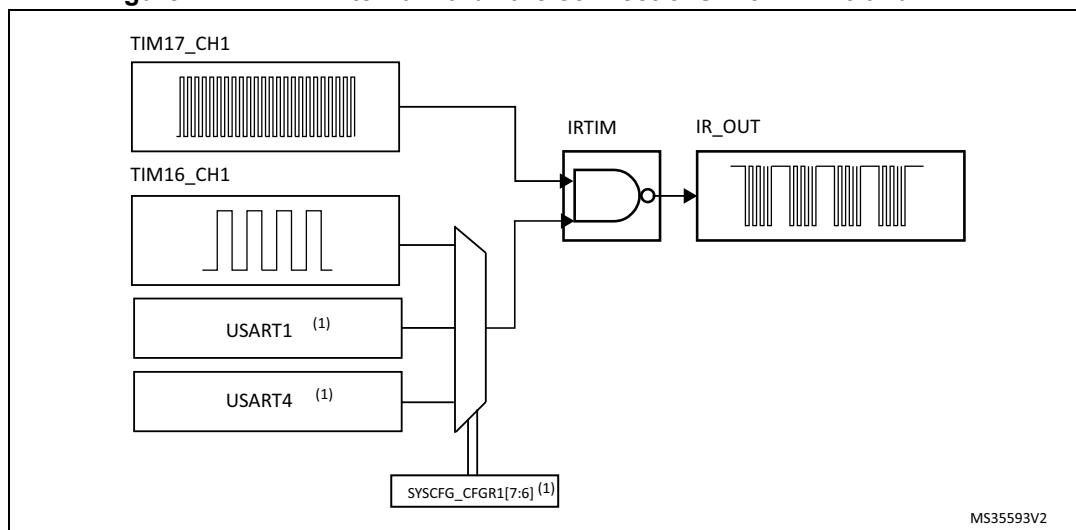
An infrared interface (IRTIM) for remote control is available on the device. It can be used with an infrared LED to perform remote control functions.

It uses internal connections with USART1, USART4 (on STM32F03x and STM32F05x) or USART2 (STM32F07x), TIM16 and TIM17 as shown in [Figure 211](#).

To generate the infrared remote control signals, the IR interface must be enabled and TIM16 channel 1 (TIM16\_OC1) and TIM17 channel 1 (TIM17\_OC1) must be properly configured to generate correct waveforms.

The infrared receiver can be implemented easily through a basic input capture mode.

**Figure 211. IRTIM internal hardware connections with TIM16 and TIM17**



1. USART1 and USART4 can be linked to IRTIM on STM32F09x devices only.

All standard IR pulse modulation modes can be obtained by programming the two timer output compare channels.

TIM17 is used to generate the high frequency carrier signal, while TIM16 generates the modulation envelope.

On STM32F09x devices, the modulation envelope can also be created from USART1 or USART4 transmitter line, upon setting appropriately the IR\_MOD[1:0] bits in SYSCFG\_CFGR1 register.

The infrared function is output on the IR\_OUT pin. The activation of this function is done through the GPIOx\_AFRx register by enabling the related alternate function bit.

The high sink LED driver capability (only available on the PB9 pin) can be activated through the I2C\_PB9\_FMP bit in the SYSCFG\_CFGR1 register and used to sink the high current needed to directly control an infrared LED.

For code example refer to the Appendix section [A.10.1: TIM16 and TIM17 configuration code example](#).

## 23 Independent watchdog (IWDG)

### 23.1 Introduction

The devices feature an embedded watchdog peripheral that offers a combination of high safety level, timing accuracy and flexibility of use. The Independent watchdog peripheral detects and solves malfunctions due to software failure, and triggers system reset when the counter reaches a given timeout value.

The independent watchdog (IWDG) is clocked by its own dedicated low-speed clock (LSI) and thus stays active even if the main clock fails.

The IWDG is best suited for applications that require the watchdog to run as a totally independent process outside the main application, but have lower timing accuracy constraints. For further information on the window watchdog, refer to [Section 24 on page 583](#).

### 23.2 IWDG main features

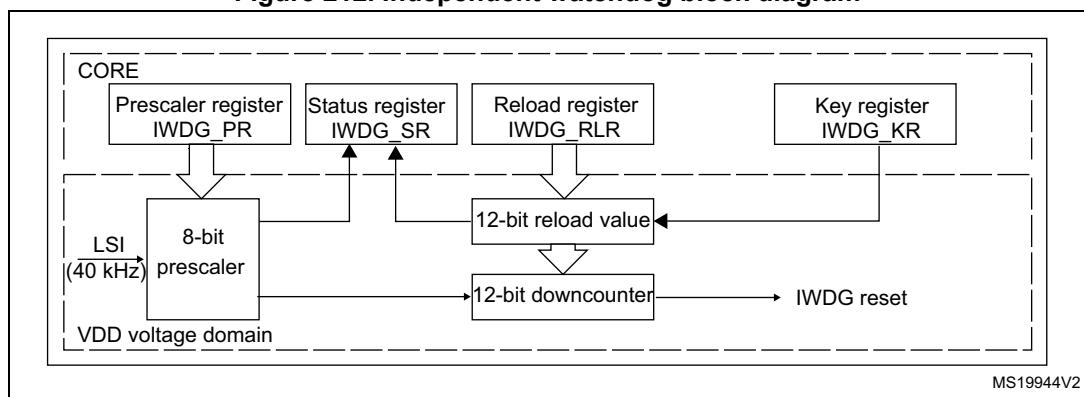
- Free-running downcounter
- Clocked from an independent RC oscillator (can operate in Standby and Stop modes)
- Conditional reset
  - Reset (if watchdog activated) when the downcounter value becomes lower than 0x000
  - Reset (if watchdog activated) if the downcounter is reloaded outside the window

### 23.3 IWDG functional description

#### 23.3.1 IWDG block diagram

*Figure 212* shows the functional blocks of the independent watchdog module.

**Figure 212. Independent watchdog block diagram**



MS19944V2

1. The register interface is located in the CORE voltage domain. The watchdog function is located in the V<sub>DD</sub> voltage domain, still functional in Stop and Standby modes.

When the independent watchdog is started by writing the value 0x0000 CCCC in the [IWDG key register \(IWDG\\_KR\)](#), the counter starts counting down from the reset value of 0xFFFF. When it reaches the end of count value (0x000) a reset signal is generated (IWDG reset).

Whenever the key value 0x0000 AAAA is written in the [IWDG key register \(IWDG\\_KR\)](#), the IWDG\_RLR value is reloaded in the counter and the watchdog reset is prevented.

Once running, the IWDG cannot be stopped.

### 23.3.2 Window option

The IWDG can also work as a window watchdog by setting the appropriate window in the [IWDG window register \(IWDG\\_WINR\)](#).

If the reload operation is performed while the counter is greater than the value stored in the [IWDG window register \(IWDG\\_WINR\)](#), then a reset is provided.

The default value of the [IWDG window register \(IWDG\\_WINR\)](#) is 0x0000 0FFF, so if it is not updated, the window option is disabled.

As soon as the window value is changed, a reload operation is performed in order to reset the downcounter to the [IWDG reload register \(IWDG\\_RLR\)](#) value and ease the cycle number calculation to generate the next reload.

#### Configuring the IWDG when the window option is enabled

1. Enable the IWDG by writing 0x0000 CCCC in the [IWDG key register \(IWDG\\_KR\)](#).
2. Enable register access by writing 0x0000 5555 in the [IWDG key register \(IWDG\\_KR\)](#).
3. Write the IWDG prescaler by programming [IWDG prescaler register \(IWDG\\_PR\)](#) from 0 to 7.
4. Write the [IWDG reload register \(IWDG\\_RLR\)](#).
5. Wait for the registers to be updated (IWDG\_SR = 0x0000 0000).
6. Write to the [IWDG window register \(IWDG\\_WINR\)](#). This automatically refreshes the counter value in the [IWDG reload register \(IWDG\\_RLR\)](#).

*Note:* Writing the window value allows the counter value to be refreshed by the RLR when [IWDG status register \(IWDG\\_SR\)](#) is set to 0x0000 0000.

For code example refer to the Appendix section [A.15.2: IWDG configuration with window code example](#).

#### Configuring the IWDG when the window option is disabled

When the window option is not used, the IWDG can be configured as follows:

1. Enable the IWDG by writing 0x0000 CCCC in the [IWDG key register \(IWDG\\_KR\)](#).
2. Enable register access by writing 0x0000 5555 in the [IWDG key register \(IWDG\\_KR\)](#).
3. Write the prescaler by programming the [IWDG prescaler register \(IWDG\\_PR\)](#) from 0 to 7.
4. Write the [IWDG reload register \(IWDG\\_RLR\)](#).
5. Wait for the registers to be updated (IWDG\_SR = 0x0000 0000).
6. Refresh the counter value with IWDG\_RLR (IWDG\_KR = 0x0000 AAAA).

For code example refer to the Appendix section [A.15.1: IWDG configuration code example](#).

### 23.3.3 Hardware watchdog

If the “Hardware watchdog” feature is enabled through the device option bits, the watchdog is automatically enabled at power-on, and generates a reset unless the *IWDG key register (IWDG\_KR)* is written by the software before the counter reaches end of count or if the downcounter is reloaded inside the window.

### 23.3.4 Register access protection

Write access to *IWDG prescaler register (IWDG\_PR)*, *IWDG reload register (IWDG\_RLR)* and *IWDG window register (IWDG\_WINR)* is protected. To modify them, the user must first write the code 0x0000 5555 in the *IWDG key register (IWDG\_KR)*. A write access to this register with a different value breaks the sequence and register access is protected again. This is the case of the reload operation (writing 0x0000 AAAA).

A status register is available to indicate that an update of the prescaler or of the downcounter reload value or of the window value is ongoing.

For code example refer to the Appendix section [A.15.1: IWDG configuration code example](#).

### 23.3.5 Debug mode

When the device enters Debug mode (core halted), the IWDG counter either continues to work normally or stops, depending on the configuration of the corresponding bit in DBGMCU freeze register.

## 23.4 IWDG registers

Refer to [Section 1.2 on page 42](#) for a list of abbreviations used in register descriptions.

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

### 23.4.1 IWDG key register (IWDG\_KR)

Address offset: 0x00

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
KEY[15:0]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

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

Bits 15:0 **KEY[15:0]**: Key value (write only, read 0x0000)

These bits must be written by software at regular intervals with the key value 0xAAAA, otherwise the watchdog generates a reset when the counter reaches 0.

Writing the key value 0x5555 to enable access to the IWDG\_PR, IWDG\_RLR and IWDG\_WINR registers (see [Section 23.3.4: Register access protection](#))

Writing the key value 0xCCCC starts the watchdog (except if the hardware watchdog option is selected)

### 23.4.2 IWDG prescaler register (IWDG\_PR)

Address offset: 0x04

Reset value: 0x0000 0000

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

Bits 31:3 Reserved, must be kept at reset value.

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

These bits are write access protected see [Section 23.3.4: Register access protection](#). They are written by software to select the prescaler divider feeding the counter clock. PVU bit of the [IWDG status register \(IWDG\\_SR\)](#) must be reset in order to be able to change the prescaler divider.

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

*Note: Reading this register returns the prescaler value from the V<sub>DD</sub> voltage domain. This value may not be up to date/valid if a write operation to this register is ongoing. For this reason the value read from this register is valid only when the PVU bit in the [IWDG status register \(IWDG\\_SR\)](#) is reset.*

### 23.4.3 IWDG reload register (IWDG\_RLR)

Address offset: 0x08

Reset value: 0x0000 0FFF (reset by Standby mode)

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

Bits 31:12 Reserved, must be kept at reset value.

Bits 11:0 **RL[11:0]**: Watchdog counter reload value

These bits are write access protected see [Register access protection](#). They are written by software to define the value to be loaded in the watchdog counter each time the value 0xAAAA is written in the [IWDG key register \(IWDG\\_KR\)](#). The watchdog counter counts down from this value. The timeout period is a function of this value and the clock prescaler. Refer to the datasheet for the timeout information.

The RVU bit in the [IWDG status register \(IWDG\\_SR\)](#) must be reset to be able to change the reload value.

*Note: Reading this register returns the reload value from the  $V_{DD}$  voltage domain. This value may not be up to date/valid if a write operation to this register is ongoing on it. For this reason the value read from this register is valid only when the RVU bit in the [IWDG status register \(IWDG\\_SR\)](#) is reset.*

### 23.4.4 IWDG status register (IWDG\_SR)

Address offset: 0x0C

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

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

Bits 31:3 Reserved, must be kept at reset value.

Bit 2 **WVU**: Watchdog counter window value update

This bit is set by hardware to indicate that an update of the window value is ongoing. It is reset by hardware when the reload value update operation is completed in the V<sub>DD</sub> voltage domain (takes up to five RC 40 kHz cycles).

Window value can be updated only when WVU bit is reset.

Bit 1 **RVU**: Watchdog counter reload value update

This bit is set by hardware to indicate that an update of the reload value is ongoing. It is reset by hardware when the reload value update operation is completed in the V<sub>DD</sub> voltage domain (takes up to five RC 40 kHz cycles).

Reload value can be updated only when RVU bit is reset.

Bit 0 **PVU**: Watchdog prescaler value update

This bit is set by hardware to indicate that an update of the prescaler value is ongoing. It is reset by hardware when the prescaler update operation is completed in the V<sub>DD</sub> voltage domain (takes up to five RC 40 kHz cycles).

Prescaler value can be updated only when PVU bit is reset.

Note:

*If several reload, prescaler, or window values are used by the application, it is mandatory to wait until RVU bit is reset before changing the reload value, to wait until PVU bit is reset before changing the prescaler value, and to wait until WVU bit is reset before changing the window value. However, after updating the prescaler and/or the reload/window value it is not necessary to wait until RVU or PVU or WVU is reset before continuing code execution except in case of low-power mode entry.*

### 23.4.5 IWDG window register (IWDG\_WINR)

Address offset: 0x10

Reset value: 0x0000 0FFF (reset by Standby mode)

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

Bits 31:12 Reserved, must be kept at reset value.

Bits 11:0 **WIN[11:0]**: Watchdog counter window value

These bits are write access protected, see [Section 23.3.4](#), they contain the high limit of the window value to be compared with the downcounter.

To prevent a reset, the downcounter must be reloaded when its value is lower than the window register value and greater than 0x0

The WVU bit in the [IWDG status register \(IWDG\\_SR\)](#) must be reset in order to be able to change the reload value.

*Note: Reading this register returns the reload value from the  $V_{DD}$  voltage domain. This value may not be valid if a write operation to this register is ongoing. For this reason the value read from this register is valid only when the WVU bit in the [IWDG status register \(IWDG\\_SR\)](#) is reset.*

## **23.4.6 IWDG register map**

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

**Table 77. IWDG register map and reset values**

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 24 System window watchdog (WWDG)

### 24.1 Introduction

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

The WWDG clock is prescaled from the APB clock and has a configurable time-window that can be programmed to detect abnormally late or early application behavior.

The WWDG is best suited for applications which require the watchdog to react within an accurate timing window.

### 24.2 WWDG main features

- Programmable free-running down-counter
- Conditional reset
  - Reset (if watchdog activated) when the down-counter value becomes lower than 0x40
  - Reset (if watchdog activated) if the down-counter is reloaded outside the window (see [Figure 214](#))
- Early wake-up interrupt (EWI): triggered (if enabled and the watchdog activated) when the down-counter is equal to 0x40.

### 24.3 WWDG functional description

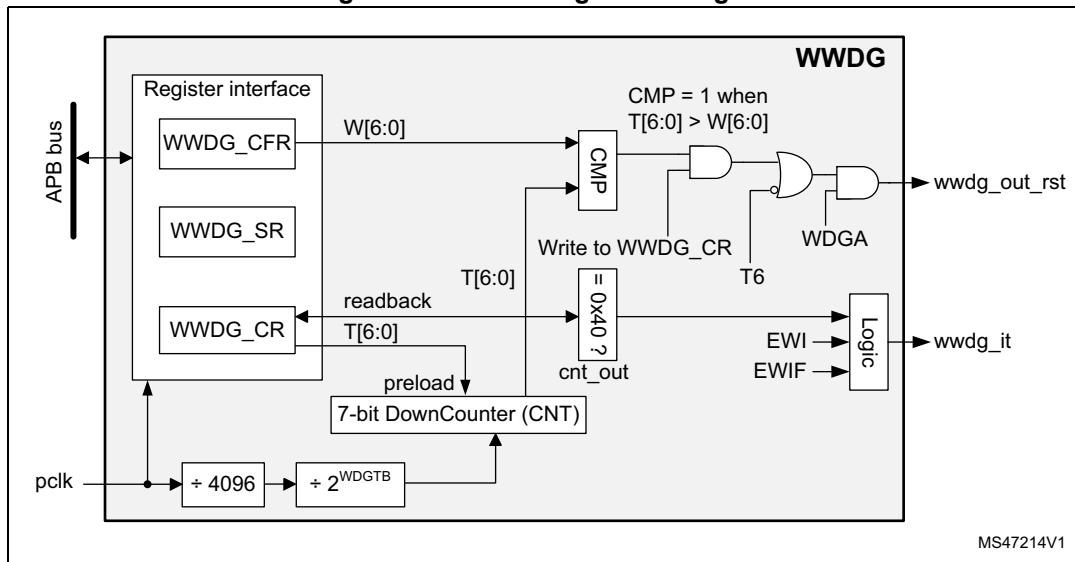
If the watchdog is activated (the WDGA bit is set in the WWDG\_CR register) and when the 7-bit down-counter (T[6:0] bits) is decremented from 0x40 to 0x3F (T6 becomes cleared), it initiates a reset. If the software reloads the counter while the counter is greater than the value stored in the window register, then a reset is generated.

The application program must write in the WWDG\_CR register at regular intervals during normal operation to prevent an MCU reset. This operation must occur only when the counter value is lower than the window register value and higher than 0x3F. The value to be stored in the WWDG\_CR register must be between 0xFF and 0xC0.

Refer to [Figure 213](#) for the WWDG block diagram.

### 24.3.1 WWDG block diagram

Figure 213. Watchdog block diagram



### 24.3.2 Enabling the watchdog

The watchdog is always disabled after a reset. It is enabled by setting the WDGA bit in the WWDG\_CR register, then it cannot be disabled again except by a reset.

### 24.3.3 Controlling the down-counter

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

The T[5:0] bits contain the number of increments that represent the time delay before the watchdog produces a reset. The timing varies between a minimum and a maximum value due to the unknown status of the prescaler when writing to the WWDG\_CR register (see [Figure 214](#)). The [WWDG configuration register \(WWDG\\_CFR\)](#) contains the high limit of the window: to prevent a reset, the down-counter must be reloaded when its value is lower than the window register value and greater than 0x3F. [Figure 214](#) 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).

### 24.3.4 How to program the watchdog timeout

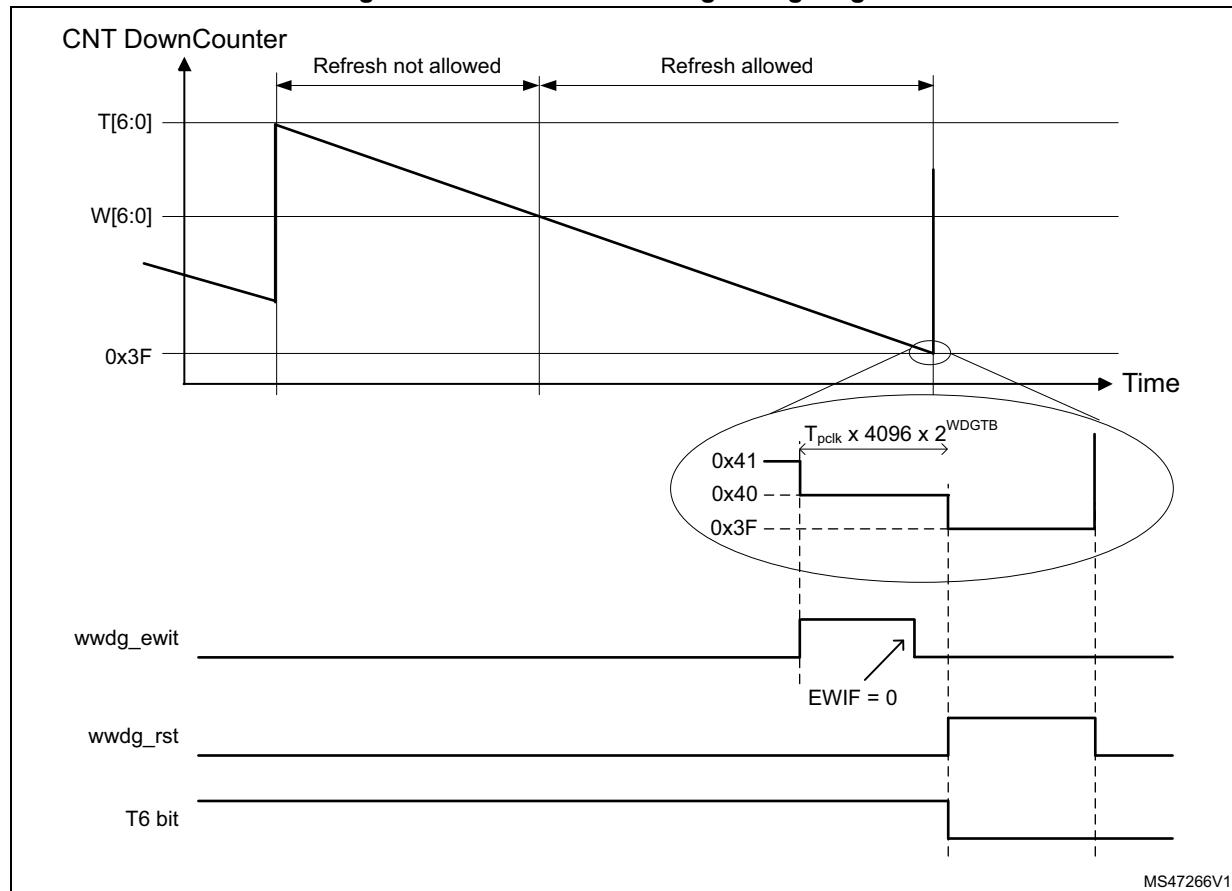
Use the formula in [Figure 214](#) to calculate the WWDG timeout.

---

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

---

Figure 214. Window watchdog timing diagram



The formula to calculate the timeout value is given by:

$$t_{\text{WWDG}} = t_{\text{PCLK}} \times 4096 \times 2^{\text{WDGTB}[1:0]} \times (T[5:0] + 1) \quad (\text{ms})$$

where:

$t_{\text{WWDG}}$ : WWDG timeout

$t_{\text{PCLK}}$ : APB clock period measured in ms

4096: value corresponding to internal divider

As an example, if APB frequency is 48 MHz, WDGTB1:0] is set to 3 and T[5:0] is set to 63:

$$t_{\text{WWDG}} = (1 / 48000) \times 4096 \times 2^3 \times (63 + 1) = 43.69\text{ms}$$

Refer to the datasheet for the minimum and maximum values of  $t_{\text{WWDG}}$ .

### 24.3.5 Debug mode

When the device enters debug mode (processor halted), the WWDG counter either continues to work normally or stops, depending on the configuration bit in DBG module. For more details refer to [Section 32.9.2: Debug support for timers, watchdog and I<sup>2</sup>C](#).

## 24.4 WWDG interrupts

The early wake-up interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. The EWI interrupt is enabled by setting the EWI bit in the WWDG\_CFR register. When the down-counter reaches the value 0x40, an EWI interrupt is generated and the corresponding interrupt service routine (ISR) can be used to trigger specific actions (such as communications or data logging) before resetting the device.

In some applications the EWI interrupt can be used to manage a software system check and/or system recovery/graceful degradation, without generating a WWDG reset. In this case the corresponding ISR has to reload the WWDG counter to avoid the WWDG reset, then trigger the required actions.

The EWI interrupt is cleared by writing '0' to the EWIF bit in the WWDG\_SR register.

**Note:** *When the EWI interrupt cannot be served (for example due to a system lock in a higher priority task) the WWDG reset is eventually generated.*

## 24.5 WWDG registers

Refer to [Section 1.2 on page 42](#) for a list of abbreviations used in register descriptions.

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

### 24.5.1 WWDG control register (WWDG\_CR)

Address offset: 0x000

Reset value: 0x0000 007F

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

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

Bit 7 **WDGA:** Activation bit

This bit is set by software and only cleared by hardware after a reset. When WDGA = 1, the watchdog can generate a reset.

0: Watchdog disabled

1: Watchdog enabled

Bits 6:0 **T[6:0]:** 7-bit counter (MSB to LSB)

These bits contain the value of the watchdog counter, decremented every  $(4096 \times 2^{\text{WDGTB1:0}})$  PCLK cycles. A reset is produced when it is decremented from 0x40 to 0x3F (T6 becomes cleared).

## 24.5.2 WWDG configuration register (WWDG\_CFR)

Address offset: 0x004

Reset value: 0x0000 007F

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	EWI	WDGTB[1:0]	W[6:0]							
						rs	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:10 Reserved, must be kept at reset value.

Bit 9 **EWI**: Early wake-up interrupt

When set, an interrupt occurs whenever the counter reaches the value 0x40. This interrupt is only cleared by hardware after a reset.

Bits 8:7 **WDGTB[1:0]**: Timer base

The time base of the prescaler can be modified as follows:

- 00: CK counter clock (PCLK div 4096) div 1
- 01: CK counter clock (PCLK div 4096) div 2
- 10: CK counter clock (PCLK div 4096) div 4
- 11: CK counter clock (PCLK div 4096) div 8

Bits 6:0 **W[6:0]**: 7-bit window value

These bits contain the window value to be compared with the down-counter.

## 24.5.3 WWDG status register (WWDG\_SR)

Address offset: 0x008

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	EWIF														
															rc_w0

Bits 31:1 Reserved, must be kept at reset value.

Bit 0 **EWIF**: Early wake-up interrupt flag

This bit is set by hardware when the counter has reached the value 0x40. It must be cleared by software by writing 0. Writing 1 has no effect. This bit is also set if the interrupt is not enabled.

#### **24.5.4 WWDG register map**

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

**Table 78. WWDG register map and reset values**

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 25 Real-time clock (RTC)

### 25.1 Introduction

The RTC provides an automatic wake-up to manage all low-power modes.

The real-time clock (RTC) is an independent BCD timer/counter. The RTC provides a time-of-day clock/calendar with programmable alarm interrupt.

The RTC includes also a periodic programmable wake-up flag with interrupt capability.

Two 32-bit registers contain the seconds, minutes, hours (12- or 24-hour format), day (day of week), date (day of month), month, and year, expressed in binary coded decimal format (BCD). The sub-seconds value is also available in binary format.

Compensations for 28-, 29- (leap year), 30-, and 31-day months are performed automatically. Daylight saving time compensation can also be performed.

Additional 32-bit registers contain the programmable alarm subseconds, seconds, minutes, hours, day, and date.

A digital calibration feature is available to compensate for any deviation in crystal oscillator accuracy.

After RTC domain reset, all RTC registers are protected against possible parasitic write accesses.

As long as the supply voltage remains in the operating range, the RTC never stops, regardless of the device status (Run mode, low-power mode or under reset).

## 25.2 RTC main features

The RTC unit main features are the following (see [Figure 215: RTC block diagram in STM32F03x, STM32F04x and STM32F05x devices](#) and [Figure 216: RTC block diagram for STM32F07x and STM32F09x devices](#)):

- Calendar with subseconds, seconds, minutes, hours (12 or 24 format), day (day of week), date (day of month), month, and year.
- Daylight saving compensation programmable by software.
- Programmable alarm with interrupt function. The alarm can be triggered by any combination of the calendar fields.
- Automatic wake-up unit generating a periodic flag that triggers an automatic wake-up interrupt.
- Reference clock detection: a more precise second source clock (50 or 60 Hz) can be used to enhance the calendar precision.
- Accurate synchronization with an external clock using the subsecond shift feature.
- Digital calibration circuit (periodic counter correction): 0.95 ppm accuracy, obtained in a calibration window of several seconds
- Time-stamp function for event saving
- Tamper detection event with configurable filter and internal pull-up
- Maskable interrupts/events:
  - Alarm A
  - Wake-up interrupt
  - Time-stamp
  - Tamper detection
- 5 backup registers.

## 25.3 RTC implementation

**Table 79. STM32F0xx RTC implementation<sup>(1)</sup>**

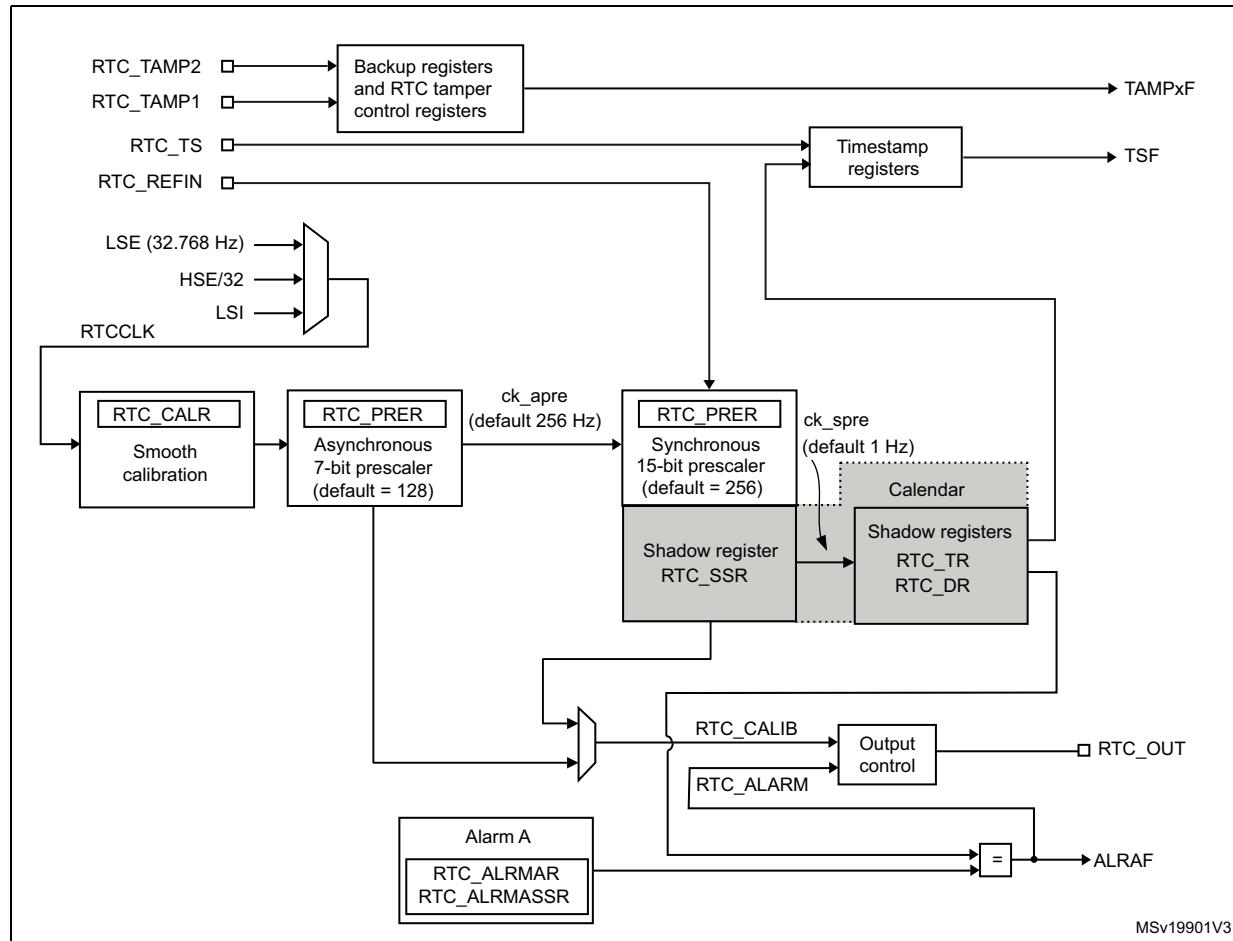
RTC Features	STM32F03x STM32F04x STM32F05x	STM32F07x STM32F09x
Periodic wake-up timer	-	X
RTC_TAMP1	X	X
RTC_TAMP2	X	X
RTC_TAMP3	-	X
Alarm A	X	X

1. X = supported, - = not supported.

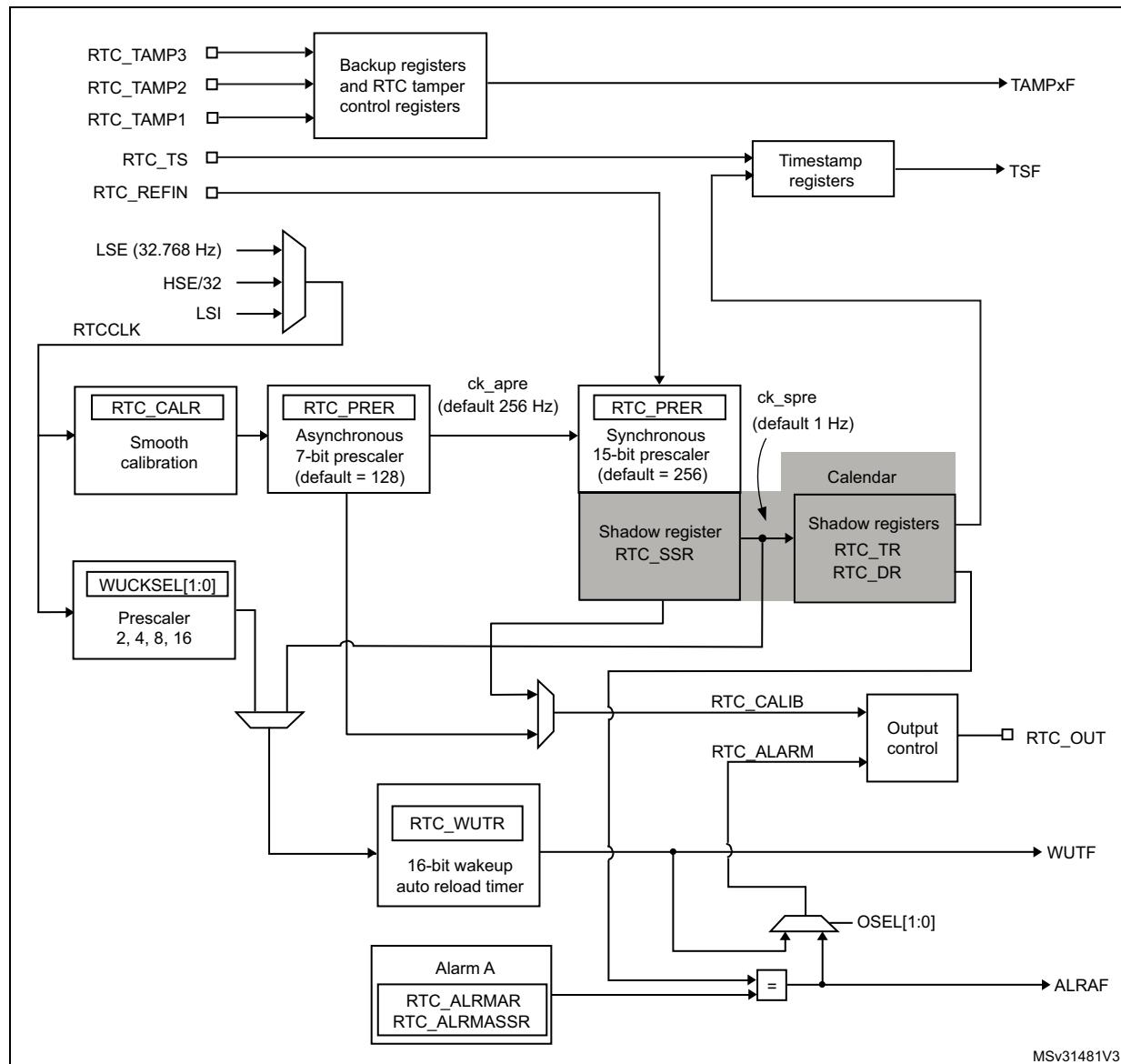
## 25.4 RTC functional description

### 25.4.1 RTC block diagram

**Figure 215. RTC block diagram in STM32F03x, STM32F04x and STM32F05x devices**



**Figure 216. RTC block diagram for STM32F07x and STM32F09x devices**



The RTC includes:

- One alarm
- Up to three tamper events from I/Os
  - Tamper detection erases the backup registers.
- One timestamp event from I/O
- Tamper event detection can generate a timestamp event
- 5 32-bit backup registers
  - The backup registers (RTC\_BKPxR) are implemented in the RTC domain that remains powered-on by VBAT when the V<sub>DD</sub> power is switched off.
- Output functions: RTC\_OUT which selects one of the following two outputs:
  - RTC\_CALIB: 512 Hz or 1Hz clock output (with an LSE frequency of 32.768 kHz). This output is enabled by setting the COE bit in the RTC\_CR register.
  - RTC\_ALARM: Alarm A. This output is selected by configuring the OSEL[1:0] bits in the RTC\_CR register.
- Input functions:
  - RTC\_TS: timestamp event
  - RTC\_TAMP1: tamper1 event detection
  - RTC\_TAMP2: tamper2 event detection
  - RTC\_TAMP3: tamper3 event detection
  - RTC\_REFIN: 50 or 60 Hz reference clock input

#### 25.4.2 GPIOs controlled by the RTC

RTC\_OUT, RTC\_TS and RTC\_TAMP1 are mapped on the same pin (PC13). PC13 pin configuration is controlled by the RTC, whatever the PC13 GPIO configuration, except for the RTC\_ALARM output open-drain mode. In this particular case, the GPIO must be configured as input. The RTC functions mapped on PC13 are available in all low-power modes and in VBAT mode.

The selection of the RTC\_ALARM output is performed through the RTC\_TAFCR register as follows: the PC13VALUE bit is used to select whether the RTC\_ALARM output is configured in push-pull or open drain mode.

When PC13 is not used as RTC function, it can be forced in output push-pull mode by setting the PC13MODE bit in the RTC\_TAFCR. The output data value is then given by the PC13VALUE bit. In this case, PC13 output push-pull state and data are preserved in Standby mode.

The output mechanism follows the priority order shown in [Table 80](#).

When PC14 and PC15 are not used as LSE oscillator, they can be forced in output push-pull mode by setting the PC14MODE and PC15MODE bits in the RTC\_TAFCR register respectively. The output data values are then given by PC14VALUE and PC15VALUE. In this case, the PC14 and PC15 output push-pull states and data values are preserved in Standby mode.

The output mechanism follows the priority order shown in [Table 81](#) and [Table 82](#).

**Table 80. RTC pin PC13 configuration<sup>(1)</sup>**

<b>Pin configuration and function</b>	<b>RTC_ALARM output enabled</b>	<b>RTC_CALIB output enabled</b>	<b>RTC_TAMP1 input enabled</b>	<b>RTC_TS input enabled</b>	<b>PC13MODE bit</b>	<b>PC13VALUE bit</b>
RTC_ALARM output OD	1	Don't care	Don't care	Don't care	Don't care	0
RTC_ALARM output PP	1	Don't care	Don't care	Don't care	Don't care	1
RTC_CALIB output PP	0	1	Don't care	Don't care	Don't care	Don't care
RTC_TAMP1 input floating	0	0	1	0	Don't care	Don't care
RTC_TS and RTC_TAMP1 input floating	0	0	1	1	Don't care	Don't care
RTC_TS input floating	0	0	0	1	Don't care	Don't care
Output PP forced	0	0	0	0	1	PC13 output data value
Wake-up pin or Standard GPIO	0	0	0	0	0	Don't care

1. OD: open drain; PP: push-pull.

**Table 81. LSE pin PC14 configuration<sup>(1)</sup>**

<b>Pin configuration and function</b>	<b>LSEON bit in RCC_BDCR register</b>	<b>LSEBYP bit in RCC_BDCR register</b>	<b>PC14MODE bit</b>	<b>PC14VALUE bit</b>
LSE oscillator	1	0	Don't care	Don't care
LSE bypass	1	1	Don't care	Don't care
Output PP forced	0	Don't care	1	PC14 output data value
Standard GPIO	0	Don't care	0	Don't care

1. OD: open drain; PP: push-pull.

**Table 82. LSE pin PC15 configuration<sup>(1)</sup>**

<b>Pin configuration and function</b>	<b>LSEON bit in RCC_BDCR register</b>	<b>LSEBYP bit in RCC_BDCR register</b>	<b>PC15MODE bit</b>	<b>PC15VALUE bit</b>
LSE oscillator	1	0	Don't care	Don't care
Output PP forced	1	1	1	PC15 output data value
	0	Don't care		
Standard GPIO	0	Don't care	0	Don't care

1. OD: open drain; PP: push-pull.

### 25.4.3 Clock and prescalers

The RTC clock source (RTCCLK) is selected through the clock controller among the LSE clock, the LSI oscillator clock, and the HSE clock. For more information on the RTC clock source configuration, refer to [Section 6: Reset and clock control \(RCC\)](#).

A programmable prescaler stage generates a 1 Hz clock which is used to update the calendar. To minimize power consumption, the prescaler is split into 2 programmable prescalers (see [Figure 215: RTC block diagram in STM32F03x, STM32F04x and STM32F05x devices](#) and [Figure 216: RTC block diagram for STM32F07x and STM32F09x devices](#)):

- A 7-bit asynchronous prescaler configured through the PREDIV\_A bits of the RTC\_PRER register.
- A 15-bit synchronous prescaler configured through the PREDIV\_S bits of the RTC\_PRER register.

**Note:** When both prescalers are used, it is recommended to configure the asynchronous prescaler to a high value to minimize consumption.

The asynchronous prescaler division factor is set to 128, and the synchronous division factor to 256, to obtain an internal clock frequency of 1 Hz (ck\_spre) with an LSE frequency of 32.768 kHz.

The minimum division factor is 1 and the maximum division factor is  $2^{22}$ .

This corresponds to a maximum input frequency of around 4 MHz.

$f_{ck\_apre}$  is given by the following formula:

$$f_{CK\_APRE} = \frac{f_{RTCCLK}}{\text{PREDIV\_A} + 1}$$

The ck\_apre clock is used to clock the binary RTC\_SSR subseconds downcounter. When it reaches 0, RTC\_SSR is reloaded with the content of PREDIV\_S.

$f_{ck\_spre}$  is given by the following formula:

$$f_{CK\_SPRE} = \frac{f_{RTCCLK}}{(\text{PREDIV\_S} + 1) \times (\text{PREDIV\_A} + 1)}$$

The ck\_spre clock can be used either to update the calendar or as timebase for the 16-bit wake-up auto-reload timer. To obtain short timeout periods, the 16-bit wake-up auto-reload timer can also run with the RTCCLK divided by the programmable 4-bit asynchronous prescaler (see [Section 25.4.6: Periodic auto-wake-up](#) for details).

### 25.4.4 Real-time clock and calendar

The RTC calendar time and date registers are accessed through shadow registers which are synchronized with PCLK (APB clock). They can also be accessed directly in order to avoid waiting for the synchronization duration.

- RTC\_SSR for the subseconds
- RTC\_TR for the time
- RTC\_DR for the date

Every RTCCLK period, the current calendar value is copied into the shadow registers, and the RSF bit of RTC\_ISR register is set (see [Section 25.7.4: RTC initialization and status register \(RTC\\_ISR\)](#)). The copy is not performed in Stop and Standby mode. When exiting these modes, the shadow registers are updated after up to 1 RTCCLK period.

When the application reads the calendar registers, it accesses the content of the shadow registers. It is possible to make a direct access to the calendar registers by setting the BYPSHAD control bit in the RTC\_CR register. By default, this bit is cleared, and the user accesses the shadow registers.

When reading the RTC\_SSR, RTC\_TR or RTC\_DR registers in BYPSHAD=0 mode, the frequency of the APB clock ( $f_{APB}$ ) must be at least 7 times the frequency of the RTC clock ( $f_{RTCCLK}$ ).

The shadow registers are reset by system reset.

#### 25.4.5 Programmable alarm

The RTC unit provides programmable alarm: Alarm A.

The programmable alarm function is enabled through the ALRAE bit in the RTC\_CR register. The ALRAF is set to 1 if the calendar subseconds, seconds, minutes, hours, date or day match the values programmed in the alarm registers RTC\_ALRMASSR and RTC\_ALRMAR. Each calendar field can be independently selected through the MSKx bits of the RTC\_ALRMAR register, and through the MASKSSx bits of the RTC\_ALRMASSR register. The alarm interrupt is enabled through the ALRAIE bit in the RTC\_CR register.

**Caution:** If the seconds field is selected (MSK1 bit reset in RTC\_ALRMAR), the synchronous prescaler division factor set in the RTC\_PRER register must be at least 3 to ensure correct behavior.

Alarm A (if enabled by bits OSEL[1:0] in RTC\_CR register) can be routed to the RTC\_ALARM output. RTC\_ALARM output polarity can be configured through bit POL the RTC\_CR register.

#### 25.4.6 Periodic auto-wake-up

The periodic wake-up flag is generated by a 16-bit programmable auto-reload down-counter. The wake-up timer range can be extended to 17 bits.

The wake-up function is enabled through the WUTE bit in the RTC\_CR register.

The wake-up timer clock input can be:

- RTC clock (RTCCLK) divided by 2, 4, 8, or 16.  
When RTCCLK is LSE(32.768 kHz), this allows to configure the wake-up interrupt period from 122  $\mu$ s to 32 s, with a resolution down to 61  $\mu$ s.
- ck\_spre (usually 1 Hz internal clock)  
When ck\_spre frequency is 1Hz, this allows to achieve a wake-up time from 1 s to around 36 hours with one-second resolution. This large programmable time range is divided in 2 parts:
  - from 1s to 18 hours when WUCKSEL [2:1] = 10
  - and from around 18h to 36h when WUCKSEL[2:1] = 11. In this last case 2<sup>16</sup> is added to the 16-bit counter current value. When the initialization sequence is complete (see [Programming the wake-up timer on page 598](#)), the timer starts counting down. When the wake-up function is enabled, the down-counting remains

active in low-power modes. In addition, when it reaches 0, the WUTF flag is set in the RTC\_ISR register, and the wake-up counter is automatically reloaded with its reload value (RTC\_WUTR register value).

The WUTF flag must then be cleared by software.

When the periodic wake-up interrupt is enabled by setting the WUTIE bit in the RTC\_CR register, it can exit the device from low-power modes.

The periodic wake-up flag can be routed to the RTC\_ALARM output provided it has been enabled through bits OSEL[1:0] of RTC\_CR register. RTC\_ALARM output polarity can be configured through the POL bit in the RTC\_CR register.

System reset, as well as low-power modes (Sleep, Stop and Standby) have no influence on the wake-up timer.

## 25.4.7 RTC initialization and configuration

### RTC register access

The RTC registers are 32-bit registers. The APB interface introduces 2 wait-states in RTC register accesses except on read accesses to calendar shadow registers when BYPSHAD=0.

### RTC register write protection

After system reset, the RTC registers are protected against parasitic write access by clearing the DBP bit in the PWR\_CR register (refer to the power control section). DBP bit must be set in order to enable RTC registers write access.

After RTC domain reset, all the RTC registers are write-protected. Writing to the RTC registers is enabled by writing a key into the Write Protection register, RTC\_WPR.

The following steps are required to unlock the write protection on all the RTC registers except for RTC\_TAFCR, RTC\_BKPxR and RTC\_ISR[13:8].

1. Write '0xCA' into the RTC\_WPR register.
2. Write '0x53' into the RTC\_WPR register.

Writing a wrong key reactivates the write protection.

The protection mechanism is not affected by system reset.

### Calendar initialization and configuration

To program the initial time and date calendar values, including the time format and the prescaler configuration, the following sequence is required:

1. Set INIT bit to 1 in the RTC\_ISR register to enter initialization mode. In this mode, the calendar counter is stopped and its value can be updated.
2. Poll INITF bit of in the RTC\_ISR register. The initialization phase mode is entered when INITF is set to 1. It takes around 2 RTCCLK clock cycles (due to clock synchronization).
3. To generate a 1 Hz clock for the calendar counter, program both the prescaler factors in RTC\_PRER register.
4. Load the initial time and date values in the shadow registers (RTC\_TR and RTC\_DR), and configure the time format (12 or 24 hours) through the FMT bit in the RTC\_CR register.

5. Exit the initialization mode by clearing the INIT bit. The actual calendar counter value is then automatically loaded and the counting restarts after 4 RTCCLK clock cycles.

When the initialization sequence is complete, the calendar starts counting.

**Note:** *After a system reset, the application can read the INITS flag in the RTC\_ISR register to check if the calendar has been initialized or not. If this flag equals 0, the calendar has not been initialized since the year field is set at its RTC domain reset default value (0x00).*

*To read the calendar after initialization, the software must first check that the RSF flag is set in the RTC\_ISR register.*

For code example refer to the Appendix section [A.16.1: RTC calendar configuration code example](#).

### Daylight saving time

The daylight saving time management is performed through bits SUB1H, ADD1H, and BKP of the RTC\_CR register.

Using SUB1H or ADD1H, the software can subtract or add one hour to the calendar in one single operation without going through the initialization procedure.

In addition, the software can use the BKP bit to memorize this operation.

### Programming the alarm

A similar procedure must be followed to program or update the programmable alarms.

1. Clear ALRAE in RTC\_CR to disable Alarm A.
2. Program the Alarm A registers (RTC\_ALRMASSR/RTC\_ALRMAR).
3. Set ALRAE in the RTC\_CR register to enable Alarm A again.

**Note:** *Each change of the RTC\_CR register is taken into account after around 2 RTCCLK clock cycles due to clock synchronization.*

For code example refer to the Appendix section [A.16.2: RTC alarm configuration code example](#).

### Programming the wake-up timer

The following sequence is required to configure or change the wake-up timer auto-reload value (WUT[15:0] in RTC\_WUTR):

1. Clear WUTE in RTC\_CR to disable the wake-up timer.
2. Poll WUTWF until it is set in RTC\_ISR to make sure the access to wake-up auto-reload counter and to WUCKSEL[2:0] bits is allowed. It takes around 2 RTCCLK clock cycles (due to clock synchronization).
3. Program the wake-up auto-reload value WUT[15:0], and the wake-up clock selection (WUCKSEL[2:0] bits in RTC\_CR). Set WUTE in RTC\_CR to enable the timer again. The wake-up timer restarts down-counting. The WUTWF bit is cleared up to 2 RTCCLK clock cycles after WUTE is cleared, due to clock synchronization.

For code example refer to the Appendix section [A.16.3: RTC WUT configuration code example](#).

## 25.4.8 Reading the calendar

### When BYPSHAD control bit is cleared in the RTC\_CR register

To read the RTC calendar registers (RTC\_SSR, RTC\_TR and RTC\_DR) properly, the APB clock frequency ( $f_{PCLK}$ ) must be equal to or greater than seven times the RTC clock frequency ( $f_{RTCCLK}$ ). This ensures a secure behavior of the synchronization mechanism.

If the APB clock frequency is less than seven times the RTC clock frequency, the software must read the calendar time and date registers twice. If the second read of the RTC\_TR gives the same result as the first read, this ensures that the data is correct. Otherwise a third read access must be done. In any case the APB clock frequency must never be lower than the RTC clock frequency.

The RSF bit is set in RTC\_ISR register each time the calendar registers are copied into the RTC\_SSR, RTC\_TR and RTC\_DR shadow registers. The copy is performed every RTCCLK cycle. To ensure consistency between the 3 values, reading either RTC\_SSR or RTC\_TR locks the values in the higher-order calendar shadow registers until RTC\_DR is read. In case the software makes read accesses to the calendar in a time interval smaller than 1 RTCCLK period: RSF must be cleared by software after the first calendar read, and then the software must wait until RSF is set before reading again the RTC\_SSR, RTC\_TR and RTC\_DR registers.

After waking up from low-power mode (Stop or Standby), RSF must be cleared by software. The software must then wait until it is set again before reading the RTC\_SSR, RTC\_TR and RTC\_DR registers.

The RSF bit must be cleared after wake-up and not before entering low-power mode.

After a system reset, the software must wait until RSF is set before reading the RTC\_SSR, RTC\_TR and RTC\_DR registers. Indeed, a system reset resets the shadow registers to their default values.

After an initialization (refer to [Calendar initialization and configuration on page 597](#)): the software must wait until RSF is set before reading the RTC\_SSR, RTC\_TR and RTC\_DR registers.

After synchronization (refer to [Section 25.4.10: RTC synchronization](#)): the software must wait until RSF is set before reading the RTC\_SSR, RTC\_TR and RTC\_DR registers.

For code example refer to the Appendix section [A.16.4: RTC read calendar code example](#).

### When the BYPSHAD control bit is set in the RTC\_CR register (bypass shadow registers)

Reading the calendar registers gives the values from the calendar counters directly, thus eliminating the need to wait for the RSF bit to be set. This is especially useful after exiting from low-power modes (STOP or Standby), since the shadow registers are not updated during these modes.

When the BYPSHAD bit is set to 1, the results of the different registers might not be coherent with each other if an RTCCLK edge occurs between two read accesses to the registers. Additionally, the value of one of the registers may be incorrect if an RTCCLK edge occurs during the read operation. The software must read all the registers twice, and then compare the results to confirm that the data is coherent and correct. Alternatively, the software can just compare the two results of the least-significant calendar register.

**Note:** While BYPSHAD=1, instructions which read the calendar registers require one extra APB cycle to complete.

### 25.4.9 Resetting the RTC

The calendar shadow registers (RTC\_SSR, RTC\_TR and RTC\_DR) and some bits of the RTC status register (RTC\_ISR) are reset to their default values by all available system reset sources.

On the contrary, the following registers are reset to their default values by a RTC domain reset and are not affected by a system reset: the RTC current calendar registers, the RTC control register (RTC\_CR), the prescaler register (RTC\_PRER), the RTC calibration register (RTC\_CALR), the RTC shift register (RTC\_SHIFTR), the RTC timestamp registers (RTC\_TSSSR, RTC\_TSTR and RTC\_TSDR), the RTC tamper and alternate function configuration register (RTC\_TAFCR), the RTC backup registers (RTC\_BKPxR), the wake-up timer register (RTC\_WUTR), the Alarm A registers (RTC\_ALRMASSR/RTC\_ALRMAR).

In addition, when it is clocked by the LSE, the RTC keeps on running under system reset if the reset source is different from the RTC domain reset one (refer to the RTC clock section of the Reset and clock controller for details on the list of RTC clock sources not affected by system reset). When a RTC domain reset occurs, the RTC is stopped and all the RTC registers are set to their reset values.

### 25.4.10 RTC synchronization

The RTC can be synchronized to a remote clock with a high degree of precision. After reading the sub-second field (RTC\_SSR or RTC\_TSSSR), a calculation can be made of the precise offset between the times being maintained by the remote clock and the RTC. The RTC can then be adjusted to eliminate this offset by “shifting” its clock by a fraction of a second using RTC\_SHIFTR.

RTC\_SSR contains the value of the synchronous prescaler counter. This allows one to calculate the exact time being maintained by the RTC down to a resolution of  $1 / (\text{PREDIV\_S} + 1)$  seconds. As a consequence, the resolution can be improved by increasing the synchronous prescaler value (PREDIV\_S[14:0]. The maximum resolution allowed (30.52 µs with a 32768 Hz clock) is obtained with PREDIV\_S set to 0x7FFF.

However, increasing PREDIV\_S means that PREDIV\_A must be decreased in order to maintain the synchronous prescaler output at 1 Hz. In this way, the frequency of the asynchronous prescaler output increases, which may increase the RTC dynamic consumption.

The RTC can be finely adjusted using the RTC shift control register (RTC\_SHIFTR). Writing to RTC\_SHIFTR can shift (either delay or advance) the clock by up to a second with a resolution of  $1 / (\text{PREDIV\_S} + 1)$  seconds. The shift operation consists of adding the SUBFS[14:0] value to the synchronous prescaler counter SS[15:0]: this will delay the clock. If at the same time the ADD1S bit is set, this results in adding one second and at the same time subtracting a fraction of second, so this will advance the clock.

**Caution:** Before initiating a shift operation, the user must check that SS[15] = 0 in order to ensure that no overflow will occur.

As soon as a shift operation is initiated by a write to the RTC\_SHIFTR register, the SHPF flag is set by hardware to indicate that a shift operation is pending. This bit is cleared by hardware as soon as the shift operation has completed.

**Caution:** This synchronization feature is not compatible with the reference clock detection feature: firmware must not write to RTC\_SHIFTR when REFCKON=1.

#### 25.4.11 RTC reference clock detection

The update of the RTC calendar can be synchronized to a reference clock, RTC\_REFIN, which is usually the mains frequency (50 or 60 Hz). The precision of the RTC\_REFIN reference clock should be higher than the 32.768 kHz LSE clock. When the RTC\_REFIN detection is enabled (REFCKON bit of RTC\_CR set to 1), the calendar is still clocked by the LSE, and RTC\_REFIN is used to compensate for the imprecision of the calendar update frequency (1 Hz).

Each 1 Hz clock edge is compared to the nearest RTC\_REFIN clock edge (if one is found within a given time window). In most cases, the two clock edges are properly aligned. When the 1 Hz clock becomes misaligned due to the imprecision of the LSE clock, the RTC shifts the 1 Hz clock a bit so that future 1 Hz clock edges are aligned. Thanks to this mechanism, the calendar becomes as precise as the reference clock.

The RTC detects if the reference clock source is present by using the 256 Hz clock (ck\_apre) generated from the 32.768 kHz quartz. The detection is performed during a time window around each of the calendar updates (every 1 s). The window equals 7 ck\_apre periods when detecting the first reference clock edge. A smaller window of 3 ck\_apre periods is used for subsequent calendar updates.

Each time the reference clock is detected in the window, the synchronous prescaler which outputs the ck\_spre clock is forced to reload. This has no effect when the reference clock and the 1 Hz clock are aligned because the prescaler is being reloaded at the same moment. When the clocks are not aligned, the reload shifts future 1 Hz clock edges a little for them to be aligned with the reference clock.

If the reference clock halts (no reference clock edge occurred during the 3 ck\_apre window), the calendar is updated continuously based solely on the LSE clock. The RTC then waits for the reference clock using a large 7 ck\_apre period detection window centered on the ck\_spre edge.

When the RTC\_REFIN detection is enabled, PREDIV\_A and PREDIV\_S must be set to their default values:

- PREDIV\_A = 0x007F
- PREDIV\_S = 0x00FF

*Note:* *RTC\_REFIN clock detection is not available in Standby mode.*

#### 25.4.12 RTC smooth digital calibration

The RTC frequency can be digitally calibrated with a resolution of about 0.954 ppm with a range from -487.1 ppm to +488.5 ppm. The correction of the frequency is performed using series of small adjustments (adding and/or subtracting individual RTCCLK pulses). These adjustments are fairly well distributed so that the RTC is well calibrated even when observed over short durations of time.

The smooth digital calibration is performed during a cycle of about  $2^{20}$  RTCCLK pulses, or 32 seconds when the input frequency is 32768 Hz. This cycle is maintained by a 20-bit counter, cal\_cnt[19:0], clocked by RTCCLK.

The smooth calibration register (RTC\_CALR) specifies the number of RTCCLK clock cycles to be masked during the 32-second cycle:

- Setting the bit CALM[0] to 1 causes exactly one pulse to be masked during the 32-second cycle.
- Setting CALM[1] to 1 causes two additional cycles to be masked
- Setting CALM[2] to 1 causes four additional cycles to be masked
- and so on up to CALM[8] set to 1 which causes 256 clocks to be masked.

*Note: CALM[8:0] (RTC\_CALR) specifies the number of RTCCLK pulses to be masked during the 32-second cycle. Setting the bit CALM[0] to '1' causes exactly one pulse to be masked during the 32-second cycle at the moment when cal\_cnt[19:0] is 0x80000; CALM[1]=1 causes two other cycles to be masked (when cal\_cnt is 0x40000 and 0xC0000); CALM[2]=1 causes four other cycles to be masked (cal\_cnt = 0x20000/0x60000/0xA0000/0xE0000); and so on up to CALM[8]=1 which causes 256 clocks to be masked (cal\_cnt = 0xXX800).*

While CALM allows the RTC frequency to be reduced by up to 487.1 ppm with fine resolution, the bit CALP can be used to increase the frequency by 488.5 ppm. Setting CALP to '1' effectively inserts an extra RTCCLK pulse every  $2^{11}$  RTCCLK cycles, which means that 512 clocks are added during every 32-second cycle.

Using CALM together with CALP, an offset ranging from -511 to +512 RTCCLK cycles can be added during the 32-second cycle, which translates to a calibration range of -487.1 ppm to +488.5 ppm with a resolution of about 0.954 ppm.

The formula to calculate the effective calibrated frequency ( $F_{CAL}$ ) given the input frequency ( $F_{RTCCLK}$ ) is as follows:

$$F_{CAL} = F_{RTCCLK} \times [1 + (CALP \times 512 - CALM) / (2^{20} + CALM - CALP \times 512)]$$

### Calibration when PREDIV\_A<3

The CALP bit can not be set to 1 when the asynchronous prescaler value (PREDIV\_A bits in RTC\_PRER register) is less than 3. If CALP was already set to 1 and PREDIV\_A bits are set to a value less than 3, CALP is ignored and the calibration operates as if CALP was equal to 0.

To perform a calibration with PREDIV\_A less than 3, the synchronous prescaler value (PREDIV\_S) should be reduced so that each second is accelerated by 8 RTCCLK clock cycles, which is equivalent to adding 256 clock cycles every 32 seconds. As a result, between 255 and 256 clock pulses (corresponding to a calibration range from 243.3 to 244.1 ppm) can effectively be added during each 32-second cycle using only the CALM bits.

With a nominal RTCCLK frequency of 32768 Hz, when PREDIV\_A equals 1 (division factor of 2), PREDIV\_S should be set to 16379 rather than 16383 (4 less). The only other interesting case is when PREDIV\_A equals 0, PREDIV\_S should be set to 32759 rather than 32767 (8 less).

If PREDIV\_S is reduced in this way, the formula given the effective frequency of the calibrated input clock is as follows:

$$F_{CAL} = F_{RTCCLK} \times [1 + (256 - CALM) / (2^{20} + CALM - 256)]$$

In this case, CALM[7:0] equals 0x100 (the midpoint of the CALM range) is the correct setting if RTCCLK is exactly 32768.00 Hz.

### Verifying the RTC calibration

RTC precision is ensured by measuring the precise frequency of RTCCLK and calculating the correct CALM value and CALP values. An optional 1 Hz output is provided to allow applications to measure and verify the RTC precision.

Measuring the precise frequency of the RTC over a limited interval can result in a measurement error of up to 2 RTCCLK clock cycles over the measurement period, depending on how the digital calibration cycle is aligned with the measurement period.

However, this measurement error can be eliminated if the measurement period is the same length as the calibration cycle period. In this case, the only error observed is the error due to the resolution of the digital calibration.

- By default, the calibration cycle period is 32 seconds.

Using this mode and measuring the accuracy of the 1 Hz output over exactly 32 seconds guarantees that the measure is within 0.477 ppm (0.5 RTCCLK cycles over 32 seconds, due to the limitation of the calibration resolution).

- CALW16 bit of the RTC\_CALR register can be set to 1 to force a 16- second calibration cycle period.

In this case, the RTC precision can be measured during 16 seconds with a maximum error of 0.954 ppm (0.5 RTCCLK cycles over 16 seconds). However, since the calibration resolution is reduced, the long term RTC precision is also reduced to 0.954 ppm: CALM[0] bit is stuck at 0 when CALW16 is set to 1.

- CALW8 bit of the RTC\_CALR register can be set to 1 to force a 8- second calibration cycle period.

In this case, the RTC precision can be measured during 8 seconds with a maximum error of 1.907 ppm (0.5 RTCCLK cycles over 8s). The long term RTC precision is also reduced to 1.907 ppm: CALM[1:0] bits are stuck at 00 when CALW8 is set to 1.

### Re-calibration on-the-fly

The calibration register (RTC\_CALR) can be updated on-the-fly while RTC\_ISR/INITF=0, by using the follow process:

1. Poll the RTC\_ISR/RECALPF (re-calibration pending flag).
2. If it is set to 0, write a new value to RTC\_CALR, if necessary. RECALPF is then automatically set to 1
3. Within three ck\_apre cycles after the write operation to RTC\_CALR, the new calibration settings take effect.

For code example refer to the Appendix section [A.16.5: RTC calibration code example](#).

### 25.4.13 Time-stamp function

Time-stamp is enabled by setting the TSE bit of RTC\_CR register to 1.

The calendar is saved in the time-stamp registers (RTC\_TSSSR, RTC\_TSTR, RTC\_TSDR) when a time-stamp event is detected on the RTC\_TS pin.

When a time-stamp event occurs, the time-stamp flag bit (TSF) in RTC\_ISR register is set.

By setting the TSIE bit in the RTC\_CR register, an interrupt is generated when a time-stamp event occurs.

If a new time-stamp event is detected while the time-stamp flag (TSF) is already set, the time-stamp overflow flag (TSOVF) flag is set and the time-stamp registers (RTC\_TSTR and RTC\_TSDR) maintain the results of the previous event.

**Note:** *TSF is set 2 ck\_apre cycles after the time-stamp event occurs due to synchronization process.*

*There is no delay in the setting of TSOVF. This means that if two time-stamp events are close together, TSOVF can be seen as '1' while TSF is still '0'. As a consequence, it is recommended to poll TSOVF only after TSF has been set.*

**Caution:** If a time-stamp event occurs immediately after the TSF bit is supposed to be cleared, then both TSF and TSOVF bits are set. To avoid masking a time-stamp event occurring at the same moment, the application must not write '0' into TSF bit unless it has already read it to '1'.

Optionally, a tamper event can cause a time-stamp to be recorded. See the description of the TAMPTS control bit in [Section 25.7.15: RTC tamper and alternate function configuration register \(RTC\\_TAFCR\)](#).

#### 25.4.14 Tamper detection

The RTC\_TAMPx input events can be configured either for edge detection, or for level detection with filtering.

The tamper detection can be configured for the following purposes:

- erase the RTC backup registers
- generate an interrupt, capable to wake-up from Stop and Standby modes

#### RTC backup registers

The backup registers (RTC\_BKPxR) are not reset by system reset or when the device wakes up from Standby mode.

The backup registers are reset when a tamper detection event occurs (see [Section 25.7.17: RTC backup registers \(RTC\\_BKPxR\)](#) and [Tamper detection initialization on page 604](#)).

#### Tamper detection initialization

Each input can be enabled by setting the corresponding TAMPxE bits to 1 in the RTC\_TAFCR register.

Each RTC\_TAMPx tamper detection input is associated with a flag TAMPxF in the RTC\_ISR register.

The TAMPxF flag is asserted after the tamper event on the pin, with the latency provided below:

- 3 ck\_apre cycles when TAMPFLT differs from 0x0 (Level detection with filtering)
- 3 ck\_apre cycles when TAMPTS=1 (Timestamp on tamper event)
- No latency when TAMPFLT=0x0 (Edge detection) and TAMPTS=0

A new tamper occurring on the same pin during this period and as long as TAMPxF is set cannot be detected.

By setting the TAMPIE bit in the RTC\_TAFCR register, an interrupt is generated when a tamper detection event occurs. .

### Timestamp on tamper event

With TAMPTS set to ‘1’, any tamper event causes a timestamp to occur. In this case, either the TSF bit or the TSOVF bit are set in RTC\_ISR, in the same manner as if a normal timestamp event occurs. The affected tamper flag register TAMPxF is set at the same time that TSF or TSOVF is set.

### Edge detection on tamper inputs

If the TAMPFLT bits are “00”, the RTC\_TAMPx pins generate tamper detection events when either a rising edge or a falling edge is observed depending on the corresponding TAMPxTRG bit. The internal pull-up resistors on the RTC\_TAMPx inputs are deactivated when edge detection is selected.

**Caution:** To avoid losing tamper detection events, the signal used for edge detection is logically ANDed with the corresponding TAMPxE bit in order to detect a tamper detection event in case it occurs before the RTC\_TAMPx pin is enabled.

- When TAMPxTRG = 0: if the RTC\_TAMPx is already high before tamper detection is enabled (TAMPxE bit set to 1), a tamper event is detected as soon as the RTC\_TAMPx input is enabled, even if there was no rising edge on the RTC\_TAMPx input after TAMPxE was set.
- When TAMPxTRG = 1: if the RTC\_TAMPx is already low before tamper detection is enabled, a tamper event is detected as soon as the RTC\_TAMPx input is enabled (even if there was no falling edge on the RTC\_TAMPx input after TAMPxE was set).

After a tamper event has been detected and cleared, the RTC\_TAMPx should be disabled and then re-enabled (TAMPxE set to 1) before re-programming the backup registers (RTC\_BKPxR). This prevents the application from writing to the backup registers while the RTC\_TAMPx input value still indicates a tamper detection. This is equivalent to a level detection on the RTC\_TAMPx input.

**Note:** *Tamper detection is still active when V<sub>DD</sub> power is switched off. To avoid unwanted resetting of the backup registers, the pin to which the RTC\_TAMPx is mapped should be externally tied to the correct level.*

### Level detection with filtering on RTC\_TAMPx inputs

Level detection with filtering is performed by setting TAMPFLT to a non-zero value. A tamper detection event is generated when either 2, 4, or 8 (depending on TAMPFLT) consecutive samples are observed at the level designated by the TAMPxTRG bits.

The RTC\_TAMPx inputs are precharged through the I/O internal pull-up resistance before its state is sampled, unless disabled by setting TAMPPUDIS to 1. The duration of the precharge is determined by the TAMPPRCH bits, allowing for larger capacitances on the RTC\_TAMPx inputs.

The trade-off between tamper detection latency and power consumption through the pull-up can be optimized by using TAMPFREQ to determine the frequency of the sampling for level detection.

**Note:** *Refer to the datasheets for the electrical characteristics of the pull-up resistors.*

For code example refer to the Appendix sections: [A.16.6: RTC tamper and time stamp configuration code example](#) and [A.16.7: RTC tamper and time stamp code example](#).

### 25.4.15 Calibration clock output

When the COE bit is set to 1 in the RTC\_CR register, a reference clock is provided on the RTC\_CALIB device output.

If the COSEL bit in the RTC\_CR register is reset and PREDIV\_A = 0x7F, the RTC\_CALIB frequency is  $f_{RTCCLK}/64$ . This corresponds to a calibration output at 512 Hz for an RTCCLK frequency at 32.768 kHz. The RTC\_CALIB duty cycle is irregular: there is a light jitter on falling edges. It is therefore recommended to use rising edges.

When COSEL is set and “PREDIV\_S+1” is a non-zero multiple of 256 (i.e: PREDIV\_S[7:0] = 0xFF), the RTC\_CALIB frequency is  $f_{RTCCLK}/(256 * (PREDIV_A+1))$ . This corresponds to a calibration output at 1 Hz for prescaler default values (PREDIV\_A = 0x7F, PREDIV\_S = 0xFF), with an RTCCLK frequency at 32.768 kHz. The 1 Hz output is affected when a shift operation is on going and may toggle during the shift operation (SHPF=1).

*Note:* When COSEL bit is cleared, the RTC\_CALIB output is the output of the 6th stage of the asynchronous prescaler.

When COSEL bit is set, the RTC\_CALIB output is the output of the 8th stage of the synchronous prescaler.

For code example refer to the Appendix section [A.16.8: RTC clock output code example](#).

### 25.4.16 Alarm output

The OSEL[1:0] control bits in the RTC\_CR register are used to activate the alarm output RTC\_ALARM, and to select the function which is output. These functions reflect the contents of the corresponding flags in the RTC\_ISR register.

The polarity of the output is determined by the POL control bit in RTC\_CR so that the opposite of the selected flag bit is output when POL is set to 1.

#### Alarm output

The RTC\_ALARM pin can be configured in output open drain or output push-pull using RTC\_TAFCR register.

*Note:* Once the RTC\_ALARM output is enabled, it has priority over RTC\_CALIB (COE bit is don't care and must be kept cleared).

## 25.5 RTC low-power modes

Table 83. Effect of low-power modes on RTC

Mode	Description
Sleep	No effect RTC interrupts cause the device to exit the Sleep mode.
Stop	The RTC remains active when the RTC clock source is LSE or LSI. RTC alarm, RTC tamper event, RTC timestamp event, and RTC wake-up cause the device to exit the Stop mode.
Standby	The RTC remains active when the RTC clock source is LSE or LSI. RTC alarm, RTC tamper event, RTC timestamp event, and RTC wake-up cause the device to exit the Standby mode.

## 25.6 RTC interrupts

All RTC interrupts are connected to the EXTI controller. Refer to [Section 11.2: Extended interrupts and events controller \(EXTI\)](#).

To enable RTC interrupt(s), the following sequence is required:

1. Configure and enable the NVIC line(s) corresponding to the RTC event(s) in interrupt mode and select the rising edge sensitivity.
2. Configure and enable the RTC IRQ channel in the NVIC.
3. Configure the RTC to generate RTC interrupt(s).

**Table 84. Interrupt control bits**

Interrupt event	Event flag	Enable control bit	Exit the Sleep mode	Exit the Stop mode	Exit the Standby mode
Alarm A	ALRAF	ALRAIE	Yes	Yes <sup>(1)</sup>	Yes <sup>(1)</sup>
RTC_TS input (timestamp)	TSF	TSIE	Yes	Yes <sup>(1)</sup>	Yes <sup>(1)</sup>
RTC_TAMP1 input detection	TAMP1F	TAMPIE	Yes	Yes <sup>(1)</sup>	Yes <sup>(1)</sup>
RTC_TAMP2 input detection	TAMP2F	TAMPIE	Yes	Yes <sup>(1)</sup>	Yes <sup>(1)</sup>
RTC_TAMP3 input detection <sup>(2)</sup>	TAMP3F	TAMPIE	Yes	Yes <sup>(1)</sup>	Yes <sup>(1)</sup>

1. Wake-up from STOP and Standby modes is possible only when the RTC clock source is LSE or LSI.
2. On STM32F07x and STM32F09x devices.

## 25.7 RTC registers

Refer to [Section 1.2 on page 42](#) of the reference manual for a list of abbreviations used in register descriptions.

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

### 25.7.1 RTC time register (RTC\_TR)

The RTC\_TR is the calendar time shadow register. This register must be written in initialization mode only. Refer to [Calendar initialization and configuration on page 597](#) and [Reading the calendar on page 599](#).

This register is write protected. The write access procedure is described in [RTC register write protection on page 597](#).

Address offset: 0x00

RTC domain reset value: 0x0000 0000

System reset: 0x0000 0000 when BYPSHAD = 0. Not affected when BYPSHAD = 1.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	PM	HT[1:0]		HU[3:0]											

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	MNT[2:0]			MNU[3:0]				Res.	ST[2:0]				SU[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw		rw	rw	rw	rw

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

Bit 22 **PM**: AM/PM notation

- 0: AM or 24-hour format
- 1: PM

Bits 21:20 **HT[1:0]**: Hour tens in BCD format

Bits 19:16 **HU[3:0]**: Hour units in BCD format

Bit 15 Reserved, must be kept at reset value.

Bits 14:12 **MNT[2:0]**: Minute tens in BCD format

Bits 11:8 **MNU[3:0]**: Minute units in BCD format

Bit 7 Reserved, must be kept at reset value.

Bits 6:4 **ST[2:0]**: Second tens in BCD format

Bits 3:0 **SU[3:0]**: Second units in BCD format

## 25.7.2 RTC date register (RTC\_DR)

The RTC\_DR is the calendar date shadow register. This register must be written in initialization mode only. Refer to [Calendar initialization and configuration on page 597](#) and [Reading the calendar on page 599](#).

This register is write protected. The write access procedure is described in [RTC register write protection on page 597](#).

Address offset: 0x04

RTC domain reset value: 0x0000 2101

System reset: 0x0000 2101 when BYPSHAD = 0. Not affected when BYPSHAD = 1.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	YT[3:0]				YU[3:0]			
								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDU[2:0]			MT	MU[3:0]				Res.	Res.	DT[1:0]		DU[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw

Bits 31:24 Reserved, must be kept at reset value.

Bits 23:20 **YT[3:0]**: Year tens in BCD format

Bits 19:16 **YU[3:0]**: Year units in BCD format

Bits 15:13 **WDU[2:0]**: Week day units

- 000: forbidden
- 001: Monday
- ...
- 111: Sunday

- Bit 12 **MT**: Month tens in BCD format
- Bits 11:8 **MU[3:0]**: Month units in BCD format
- Bits 7:6 Reserved, must be kept at reset value.
- Bits 5:4 **DT[1:0]**: Date tens in BCD format
- Bits 3:0 **DU[3:0]**: Date units in BCD format

### 25.7.3 RTC control register (RTC\_CR)

Address offset: 0x08

RTC domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	COE	OSEL[1:0]	POL	COSEL	BKP	SUB1H	ADD1H	
								rw	rw	rw	rw	rw	rw	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TSIE	WUTIE	Res.	ALRAIE	TSE	WUTE	Res.	ALRAE	Res.	FMT	BYPS HAD	REFCKON	TSEDGE	WUCKSEL[2:0]		
rw	rw		rw	rw	rw		rw		rw	rw	rw	rw	rw	rw	rw

Bits 31:24 Reserved, must be kept at reset value.

Bit 23 **COE**: Calibration output enable

This bit enables the RTC\_CALIB output

0: Calibration output disabled

1: Calibration output enabled

Bits 22:21 **OSEL[1:0]**: Output selection

These bits are used to select the flag to be routed to RTC\_ALARM output

00: Output disabled

01: Alarm A output enabled

10: Reserved

11: Wake-up output enabled

Bit 20 **POL**: Output polarity

This bit is used to configure the polarity of RTC\_ALARM output

0: The pin is high when ALRAF/WUTF is asserted (depending on OSEL[1:0])

1: The pin is low when ALRAF/WUTF is asserted (depending on OSEL[1:0]).

Bit 19 **COSEL**: Calibration output selection

When COE=1, this bit selects which signal is output on RTC\_CALIB.

0: Calibration output is 512 Hz (with default prescaler setting)

1: Calibration output is 1 Hz (with default prescaler setting)

These frequencies are valid for RTCCLK at 32.768 kHz and prescalers at their default values (PREDIV\_A=127 and PREDIV\_S=255). Refer to [Section 25.4.15: Calibration clock output](#)

Bit 18 **BKP**: Backup

This bit can be written by the user to memorize whether the daylight saving time change has been performed or not.

Bit 17 **SUB1H**: Subtract 1 hour (winter time change)

When this bit is set, 1 hour is subtracted to the calendar time if the current hour is not 0. This bit is always read as 0.

Setting this bit has no effect when current hour is 0.

0: No effect

1: Subtracts 1 hour to the current time. This can be used for winter time change outside initialization mode.

Bit 16 **ADD1H**: Add 1 hour (summer time change)

When this bit is set, 1 hour is added to the calendar time. This bit is always read as 0.

0: No effect

1: Adds 1 hour to the current time. This can be used for summer time change outside initialization mode.

Bit 15 **TSIE**: Time-stamp interrupt enable

0: Time-stamp Interrupt disable

1: Time-stamp Interrupt enable

Bit 14 **WUTIE**: Wake-up timer interrupt enable

0: Wake-up timer interrupt disabled

1: Wake-up timer interrupt enabled

Bit 13 Reserved, must be kept at reset value.

Bit 12 **ALRAIE**: Alarm A interrupt enable

0: Alarm A interrupt disabled

1: Alarm A interrupt enabled

Bit 11 **TSE**: timestamp enable

0: timestamp disable

1: timestamp enable

Bit 10 **WUTE**: Wake-up timer enable

0: Wake-up timer disabled

1: Wake-up timer enabled

*Note: When the wake-up timer is disabled, wait for WUTWF=1 before enabling it again.*

Bit 9 Reserved, must be kept at reset value.

Bit 8 **ALRAE**: Alarm A enable

0: Alarm A disabled

1: Alarm A enabled

Bit 7 Reserved, must be kept at reset value.

Bit 6 **FMT**: Hour format

0: 24 hour/day format

1: AM/PM hour format

Bit 5 **BYPSHAD**: Bypass the shadow registers

0: Calendar values (when reading from RTC\_SSR, RTC\_TR, and RTC\_DR) are taken from the shadow registers, which are updated once every two RTCCLK cycles.

1: Calendar values (when reading from RTC\_SSR, RTC\_TR, and RTC\_DR) are taken directly from the calendar counters.

*Note: If the frequency of the APB clock is less than seven times the frequency of RTCCLK, BYPSHAD must be set to '1'.*

Bit 4 **REFCKON**: RTC\_REFIN reference clock detection enable (50 or 60 Hz)

- 0: RTC\_REFIN detection disabled
- 1: RTC\_REFIN detection enabled

*Note:* *PREDIV\_S* must be 0x00FF.

Bit 3 **TSEDGE**: Time-stamp event active edge

- 0: RTC\_TS input rising edge generates a time-stamp event
  - 1: RTC\_TS input falling edge generates a time-stamp event
- TSE must be reset when TSEDGE is changed to avoid unwanted TSF setting.

Bits 2:0 **WUCKSEL[2:0]**: Wake-up clock selection

- 000: RTC/16 clock is selected
- 001: RTC/8 clock is selected
- 010: RTC/4 clock is selected
- 011: RTC/2 clock is selected
- 10x: ck\_spre (usually 1 Hz) clock is selected
- 11x: ck\_spre (usually 1 Hz) clock is selected and  $2^{16}$  is added to the WUT counter value  
(see note below)

*Note:* Bits 7, 6 and 4 of this register can be written in initialization mode only (RTC\_ISR/INITF = 1).

WUT = Wake-up unit counter value.  $WUT = (0x0000 \text{ to } 0xFFFF) + 0x10000$  added when  $WUCKSEL[2:1] = 11$ .

Bits 2 to 0 of this register can be written only when RTC\_CR WUTE bit = 0 and RTC\_ISR WUTWF bit = 1.

*It is recommended not to change the hour during the calendar hour increment as it could mask the incrementation of the calendar hour.*

*ADD1H and SUB1H changes are effective in the next second.*

*This register is write protected. The write access procedure is described in [RTC register write protection on page 597](#).*

**Caution:** TSE must be reset when TSEDGE is changed to avoid spuriously setting of TSF.

## 25.7.4 RTC initialization and status register (RTC\_ISR)

This register is write protected (except for RTC\_ISR[13:8] bits). The write access procedure is described in [RTC register write protection on page 597](#).

Address offset: 0x0C

RTC domain reset value: 0x0000 0007

System reset: not affected except INIT, INITF, and RSF bits which are cleared to '0'

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RECALPF
																r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
TAMP3F	TAMP2F	TAMP1F	TSOVF	TSF	WUTF	Res.	ALRAF	INIT	INITF	RSF	INITS	SHPF	WUTWF	Res.	ALRAWF	
rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0		rc_w0	rw	r	rc_w0	r	r	r		r	

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

Bit 16 **RECALPF**: Recalibration pending Flag

The RECALPF status flag is automatically set to '1' when software writes to the RTC\_CALR register, indicating that the RTC\_CALR register is blocked. When the new calibration settings are taken into account, this bit returns to '0'. Refer to [Re-calibration on-the-fly](#).

Bit 15 **TAMP3F**: RTC\_TAMP3 detection flag

This flag is set by hardware when a tamper detection event is detected on the RTC\_TAMP3 input.

It is cleared by software writing 0

Bit 14 **TAMP2F**: RTC\_TAMP2 detection flag

This flag is set by hardware when a tamper detection event is detected on the RTC\_TAMP2 input.

It is cleared by software writing 0

Bit 13 **TAMP1F**: RTC\_TAMP1 detection flag

This flag is set by hardware when a tamper detection event is detected on the RTC\_TAMP1 input.

It is cleared by software writing 0

Bit 12 **TSOVF**: Time-stamp overflow flag

This flag is set by hardware when a time-stamp event occurs while TSF is already set.

This flag is cleared by software by writing 0. It is recommended to check and then clear TSOVF only after clearing the TSF bit. Otherwise, an overflow might not be noticed if a time-stamp event occurs immediately before the TSF bit is cleared.

Bit 11 **TSF**: Time-stamp flag

This flag is set by hardware when a time-stamp event occurs.

This flag is cleared by software by writing 0.

Bit 10 **WUTF**: Wake-up timer flag

This flag is set by hardware when the wake-up auto-reload counter reaches 0.

This flag is cleared by software by writing 0.

This flag must be cleared by software at least 1.5 RTCCLK periods before WUTF is set to 1 again.

Bit 9 Reserved, must be kept at reset value.

**Bit 8 ALRAF:** Alarm A flag

This flag is set by hardware when the time/date registers (RTC\_TR and RTC\_DR) match the Alarm A register (RTC\_ALRMAR).

This flag is cleared by software by writing 0.

**Bit 7 INIT:** Initialization mode

0: Free running mode

1: Initialization mode used to program time and date register (RTC\_TR and RTC\_DR), and prescaler register (RTC\_PRER). Counters are stopped and start counting from the new value when INIT is reset.

**Bit 6 INITF:** Initialization flag

When this bit is set to 1, the RTC is in initialization state, and the time, date and prescaler registers can be updated.

0: Calendar registers update is not allowed

1: Calendar registers update is allowed

**Bit 5 RSF:** Registers synchronization flag

This bit is set by hardware each time the calendar registers are copied into the shadow registers (RTC\_SSR, RTC\_TR and RTC\_DR). This bit is cleared by hardware in initialization mode, while a shift operation is pending (SHPF=1), or when in bypass shadow register mode (BYPSSHAD=1). This bit can also be cleared by software.

It is cleared either by software or by hardware in initialization mode.

0: Calendar shadow registers not yet synchronized

1: Calendar shadow registers synchronized

**Bit 4 INITS:** Initialization status flag

This bit is set by hardware when the calendar year field is different from 0 (RTC domain reset state).

0: Calendar has not been initialized

1: Calendar has been initialized

**Bit 3 SHPF:** Shift operation pending

0: No shift operation is pending

1: A shift operation is pending

This flag is set by hardware as soon as a shift operation is initiated by a write to the RTC\_SHIFTR register. It is cleared by hardware when the corresponding shift operation has been executed. Writing to the SHPF bit has no effect.

**Bit 2 WUTWF:** Wake-up timer write flag

This bit is set by hardware up to 2 RTCCLK cycles after the WUTE bit has been set to 0 in RTC\_CR, and is cleared up to 2 RTCCLK cycles after the WUTE bit has been set to 1. The wake-up timer values can be changed when WUTE bit is cleared and WUTWF is set.

0: Wake-up timer configuration update not allowed

1: Wake-up timer configuration update allowed

**Bit 1 Reserved,** must be kept at reset value.**Bit 0 ALRAWF:** Alarm A write flag

This bit is set by hardware when Alarm A values can be changed, after the ALRAE bit has been set to 0 in RTC\_CR.

It is cleared by hardware in initialization mode.

0: Alarm A update not allowed

1: Alarm A update allowed

**Note:** The bits ALRAF, WUTF and TSF are cleared 2 APB clock cycles after programming them to 0.

### 25.7.5 RTC prescaler register (RTC\_PRER)

This register must be written in initialization mode only. The initialization must be performed in two separate write accesses. Refer to [Calendar initialization and configuration on page 597](#).

This register is write protected. The write access procedure is described in [RTC register write protection on page 597](#).

Address offset: 0x10

RTC domain reset value: 0x007F 00FF

System reset: not affected

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

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

Bits 22:16 **PREDIV\_A[6:0]**: Asynchronous prescaler factor

This is the asynchronous division factor:

$$\text{ck_apre frequency} = \text{RTCCLK frequency}/(\text{PREDIV\_A}+1)$$

Bit 15 Reserved, must be kept at reset value.

Bits 14:0 **PREDIV\_S[14:0]**: Synchronous prescaler factor

This is the synchronous division factor:

$$\text{ck_spre frequency} = \text{ck_apre frequency}/(\text{PREDIV\_S}+1)$$

## 25.7.6 RTC wake-up timer register (RTC\_WUTR)

This register can be written only when WUTWF is set to 1 in RTC\_ISR.

This register is write protected. The write access procedure is described in [RTC register write protection on page 597](#).

Address offset: 0x14

RTC domain reset value: 0x0000 FFFF

System reset: not affected

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

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

Bits 15:0 **WUT[15:0]**: Wake-up auto-reload value bits

When the wake-up timer is enabled (WUTE set to 1), the WUTF flag is set every (WUT[15:0] + 1) ck\_wut cycles. The ck\_wut period is selected through WUCKSEL[2:0] bits of the RTC\_CR register

When WUCKSEL[2] = 1, the wake-up timer becomes 17-bits and WUCKSEL[1] effectively becomes WUT[16] the most-significant bit to be reloaded into the timer.

The first assertion of WUTF occurs (WUT+1) ck\_wut cycles after WUTE is set. Setting WUT[15:0] to 0x0000 with WUCKSEL[2:0] =011 (RTCCLK/2) is forbidden.

### 25.7.7 RTC alarm A register (RTC\_ALRMAR)

This register can be written only when ALRAWF is set to 1 in RTC\_ISR, or in initialization mode.

This register is write protected. The write access procedure is described in [RTC register write protection on page 597](#).

Address offset: 0x1C

RTC domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MSK4	WDSEL	DT[1:0]		DU[3:0]				MSK3	PM	HT[1:0]		HU[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSK2	MNT[2:0]			MNU[3:0]				MSK1	ST[2:0]			SU[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **MSK4**: Alarm A date mask

- 0: Alarm A set if the date/day match
- 1: Date/day don't care in Alarm A comparison

Bit 30 **WDSEL**: Week day selection

- 0: DU[3:0] represents the date units
- 1: DU[3:0] represents the week day. DT[1:0] is don't care.

Bits 29:28 **DT[1:0]**: Date tens in BCD format.

Bits 27:24 **DU[3:0]**: Date units or day in BCD format.

Bit 23 **MSK3**: Alarm A hours mask

- 0: Alarm A set if the hours match
- 1: Hours don't care in Alarm A comparison

Bit 22 **PM**: AM/PM notation

- 0: AM or 24-hour format
- 1: PM

Bits 21:20 **HT[1:0]**: Hour tens in BCD format.

Bits 19:16 **HU[3:0]**: Hour units in BCD format.

Bit 15 **MSK2**: Alarm A minutes mask

- 0: Alarm A set if the minutes match
- 1: Minutes don't care in Alarm A comparison

Bits 14:12 **MNT[2:0]**: Minute tens in BCD format.

Bits 11:8 **MNU[3:0]**: Minute units in BCD format.

Bit 7 **MSK1**: Alarm A seconds mask

- 0: Alarm A set if the seconds match
- 1: Seconds don't care in Alarm A comparison

Bits 6:4 **ST[2:0]**: Second tens in BCD format.

Bits 3:0 **SU[3:0]**: Second units in BCD format.

### 25.7.8 RTC write protection register (RTC\_WPR)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	KEY[7:0]														
								w	w	w	w	w	w	w	w

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

Bits 7:0 **KEY[7:0]**: Write protection key

This byte is written by software.

Reading this byte always returns 0x00.

Refer to [RTC register write protection](#) for a description of how to unlock RTC register write protection.

### 25.7.9 RTC sub second register (RTC\_SSR)

Address offset: 0x28

RTC domain reset value: 0x0000 0000

System reset: 0x0000 0000 when BYPSHAD = 0. Not affected when BYPSHAD = 1.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SS[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

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

Bits 15:0 **SS[15:0]**: Sub second value

SS[15:0] is the value in the synchronous prescaler counter. The fraction of a second is given by the formula below:

Second fraction = (PREDIV\_S - SS) / (PREDIV\_S + 1)

Note: SS can be larger than PREDIV\_S only after a shift operation. In that case, the correct time/date is one second less than as indicated by RTC\_TR/RTC\_DR.

### 25.7.10 RTC shift control register (RTC\_SHIFTR)

This register is write protected. The write access procedure is described in [RTC register write protection on page 597](#).

Address offset: 0x2C

RTC domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADD1S	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
w															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SUBFS[14:0]														
	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bit 31 **ADD1S**: Add one second

0: No effect

1: Add one second to the clock/calendar

This bit is write only and is always read as zero. Writing to this bit has no effect when a shift operation is pending (when SHPF=1, in RTC\_ISR).

This function is intended to be used with SUBFS (see description below) in order to effectively add a fraction of a second to the clock in an atomic operation.

Bits 30:15 Reserved, must be kept at reset value.

Bits 14:0 **SUBFS[14:0]**: Subtract a fraction of a second

These bits are write only and is always read as zero. Writing to this bit has no effect when a shift operation is pending (when SHPF=1, in RTC\_ISR).

The value which is written to SUBFS is added to the synchronous prescaler counter. Since this counter counts down, this operation effectively subtracts from (delays) the clock by:

Delay (seconds) = SUBFS / (PREDIV\_S + 1)

A fraction of a second can effectively be added to the clock (advancing the clock) when the ADD1S function is used in conjunction with SUBFS, effectively advancing the clock by:

Advance (seconds) = (1 - (SUBFS / (PREDIV\_S + 1))).

*Note: Writing to SUBFS causes RSF to be cleared. Software can then wait until RSF=1 to be sure that the shadow registers have been updated with the shifted time.*

### 25.7.11 RTC timestamp time register (RTC\_TSTR)

The content of this register is valid only when TSF is set to 1 in RTC\_ISR. It is cleared when TSF bit is reset.

Address offset: 0x30

RTC domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PM	HT[1:0]		HU[3:0]			
									r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	MNT[2:0]			MNU[3:0]				Res.	ST[2:0]			SU[3:0]			
	r	r	r	r	r	r	r		r	r	r	r	r	r	r

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

Bit 22 **PM**: AM/PM notation

0: AM or 24-hour format

1: PM

Bits 21:20 **HT[1:0]**: Hour tens in BCD format.

Bits 19:16 **HU[3:0]**: Hour units in BCD format.

Bit 15 Reserved, must be kept at reset value.

Bits 14:12 **MNT[2:0]**: Minute tens in BCD format.

Bits 11:8 **MNU[3:0]**: Minute units in BCD format.

Bit 7 Reserved, must be kept at reset value.

Bits 6:4 **ST[2:0]**: Second tens in BCD format.

Bits 3:0 **SU[3:0]**: Second units in BCD format.

### 25.7.12 RTC timestamp date register (RTC\_TSDR)

The content of this register is valid only when TSF is set to 1 in RTC\_ISR. It is cleared when TSF bit is reset.

Address offset: 0x34

RTC domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDU[2:0]			MT	MU[3:0]				Res.	Res.	DT[1:0]		DU[3:0]			
r	r	r	r	r	r	r	r			r	r	r	r	r	r

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

Bits 15:13 **WDU[2:0]**: Week day units

Bit 12 **MT**: Month tens in BCD format

Bits 11:8 **MU[3:0]**: Month units in BCD format

Bits 7:6 Reserved, must be kept at reset value.

Bits 5:4 **DT[1:0]**: Date tens in BCD format

Bits 3:0 **DU[3:0]**: Date units in BCD format

### 25.7.13 RTC time-stamp sub second register (RTC\_TSSSR)

The content of this register is valid only when RTC\_ISR/TSF is set. It is cleared when the RTC\_ISR/TSF bit is reset.

Address offset: 0x38

RTC domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SS[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

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

Bits 15:0 **SS[15:0]**: Sub second value

SS[15:0] is the value of the synchronous prescaler counter when the timestamp event occurred.

### 25.7.14 RTC calibration register (RTC\_CALR)

This register is write protected. The write access procedure is described in [RTC register write protection on page 597](#).

Address offset: 0x3C

RTC domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CALP	CALW8	CALW16	Res.	Res.	Res.	Res.	CALM[8:0]								
rw	rw	rw					rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bit 15 **CALP**: Increase frequency of RTC by 488.5 ppm

0: No RTCCLK pulses are added.

1: One RTCCLK pulse is effectively inserted every  $2^{11}$  pulses (frequency increased by 488.5 ppm).

This feature is intended to be used in conjunction with CALM, which lowers the frequency of the calendar with a fine resolution. If the input frequency is 32768 Hz, the number of RTCCLK pulses added during a 32-second window is calculated as follows:  $(512 * \text{CALP}) - \text{CALM}$ .

Refer to [Section 25.4.12: RTC smooth digital calibration](#).

Bit 14 **CALW8**: Use an 8-second calibration cycle period

When CALW8 is set to '1', the 8-second calibration cycle period is selected.

Note: CALM[1:0] are stuck at "00" when CALW8='1'. Refer to [Section 25.4.12: RTC smooth digital calibration](#).

Bit 13 **CALW16**: Use a 16-second calibration cycle period

When CALW16 is set to '1', the 16-second calibration cycle period is selected. This bit must not be set to '1' if CALW8=1.

Note: CALM[0] is stuck at '0' when CALW16='1'. Refer to [Section 25.4.12: RTC smooth digital calibration](#).

Bits 12:9 Reserved, must be kept at reset value.

Bits 8:0 **CALM[8:0]**: Calibration minus

The frequency of the calendar is reduced by masking CALM out of  $2^{20}$  RTCCLK pulses (32 seconds if the input frequency is 32768 Hz). This decreases the frequency of the calendar with a resolution of 0.9537 ppm.

To increase the frequency of the calendar, this feature should be used in conjunction with CALP. See [Section 25.4.12: RTC smooth digital calibration on page 601](#).

### 25.7.15 RTC tamper and alternate function configuration register (RTC\_TAFCR)

Address offset: 0x40

RTC domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PC15 MODE	PC15 VALUE	PC14 MODE	PC14 VALUE	PC13 MODE	PC13 VALUE	Res.	Res.
								rw	rw	rw	rw	rw	rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TAMPP UDIS	TAMPPRCH [1:0]	TAMPFLT[1:0]	TAMPFREQ[2:0]	TAMPT S	TAMP3 TRG	TAMP3 E	TAMP2 TRG	TAMP2 E	TAMPIE	TAMP1 TRG	TAMP1 E				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 Reserved, must be kept at reset value.

Bit 23 **PC15MODE**: PC15 mode

0: PC15 is controlled by the GPIO configuration registers. Consequently PC15 is floating in Standby mode.

1: PC15 is forced to push-pull output if LSE is disabled.

Bit 22 **PC15VALUE**: PC15 value

If the LSE is disabled and PC15MODE = 1, PC15VALUE configures the PC15 output data.

Bit 21 **PC14MODE**: PC14 mode

0: PC14 is controlled by the GPIO configuration registers. Consequently PC14 is floating in Standby mode.

1: PC14 is forced to push-pull output if LSE is disabled.

Bit 20 **PC14VALUE**: PC14 value

If the LSE is disabled and PC14MODE = 1, PC14VALUE configures the PC14 output data.

Bit 19 **PC13MODE**: PC13 mode

0: PC13 is controlled by the GPIO configuration registers. Consequently PC13 is floating in Standby mode.

1: PC13 is forced to push-pull output if all RTC functions are disabled.

Bit 18 **PC13VALUE**: RTC\_ALARM output type/PC13 value

If PC13 is used to output RTC\_ALARM, PC13VALUE configures the output configuration:

0: RTC\_ALARM is an open-drain output

1: RTC\_ALARM is a push-pull output

If all RTC functions are disabled and PC13MODE = 1, PC13VALUE configures the PC13 output data.

Bits 17:16 Reserved, must be kept at reset value.

Bit 15 **TAMPPUDIS**: RTC\_TAMPx pull-up disable

This bit determines if each of the RTC\_TAMPx pins are pre-charged before each sample.

0: Precharge RTC\_TAMPx pins before sampling (enable internal pull-up)

1: Disable precharge of RTC\_TAMPx pins.

Bits 14:13 **TAMPPRCH[1:0]**: RTC\_TAMPx precharge duration

These bit determines the duration of time during which the pull-up is activated before each sample. TAMPPRCH is valid for each of the RTC\_TAMPx inputs.

- 0x0: 1 RTCCLK cycle
- 0x1: 2 RTCCLK cycles
- 0x2: 4 RTCCLK cycles
- 0x3: 8 RTCCLK cycles

Bits 12:11 **TAMPFLT[1:0]**: RTC\_TAMPx filter count

These bits determines the number of consecutive samples at the specified level (TAMP\*TRG) needed to activate a Tamper event. TAMPFLT is valid for each of the RTC\_TAMPx inputs.

- 0x0: Tamper event is activated on edge of RTC\_TAMPx input transitions to the active level (no internal pull-up on RTC\_TAMPx input).
- 0x1: Tamper event is activated after 2 consecutive samples at the active level.
- 0x2: Tamper event is activated after 4 consecutive samples at the active level.
- 0x3: Tamper event is activated after 8 consecutive samples at the active level.

Bits 10:8 **TAMPFREQ[2:0]**: Tamper sampling frequency

Determines the frequency at which each of the RTC\_TAMPx inputs are sampled.

- 0x0: RTCCLK / 32768 (1 Hz when RTCCLK = 32768 Hz)
- 0x1: RTCCLK / 16384 (2 Hz when RTCCLK = 32768 Hz)
- 0x2: RTCCLK / 8192 (4 Hz when RTCCLK = 32768 Hz)
- 0x3: RTCCLK / 4096 (8 Hz when RTCCLK = 32768 Hz)
- 0x4: RTCCLK / 2048 (16 Hz when RTCCLK = 32768 Hz)
- 0x5: RTCCLK / 1024 (32 Hz when RTCCLK = 32768 Hz)
- 0x6: RTCCLK / 512 (64 Hz when RTCCLK = 32768 Hz)
- 0x7: RTCCLK / 256 (128 Hz when RTCCLK = 32768 Hz)

Bit 7 **TAMPTS**: Activate timestamp on tamper detection event

- 0: Tamper detection event does not cause a timestamp to be saved
- 1: Save timestamp on tamper detection event

TAMPTS is valid even if TSE=0 in the RTC\_CR register.

Bit 6 **TAMP3TRG**: Active level for RTC\_TAMP3 input

- if TAMPFLT != 00:
- 0: RTC\_TAMP3 input staying low triggers a tamper detection event.
- 1: RTC\_TAMP3 input staying high triggers a tamper detection event.
- if TAMPFLT = 00:
- 0: RTC\_TAMP3 input rising edge triggers a tamper detection event.
- 1: RTC\_TAMP3 input falling edge triggers a tamper detection event.

Bit 5 **TAMP3E**: RTC\_TAMP3 detection enable

- 0: RTC\_TAMP3 input detection disabled
- 1: RTC\_TAMP3 input detection enabled

Bit 4 **TAMP2TRG**: Active level for RTC\_TAMP2 input

- if TAMPFLT != 00:
- 0: RTC\_TAMP2 input staying low triggers a tamper detection event.
- 1: RTC\_TAMP2 input staying high triggers a tamper detection event.
- if TAMPFLT = 00:
- 0: RTC\_TAMP2 input rising edge triggers a tamper detection event.
- 1: RTC\_TAMP2 input falling edge triggers a tamper detection event.

- Bit 3 **TAMP2E**: RTC\_TAMP2 input detection enable  
0: RTC\_TAMP2 detection disabled  
1: RTC\_TAMP2 detection enabled
- Bit 2 **TAMPIE**: Tamper interrupt enable  
0: Tamper interrupt disabled  
1: Tamper interrupt enabled.
- Bit 1 **TAMP1TRG**: Active level for RTC\_TAMP1 input  
If TAMPFLT != 00  
0: RTC\_TAMP1 input staying low triggers a tamper detection event.  
1: RTC\_TAMP1 input staying high triggers a tamper detection event.  
if TAMPFLT = 00:  
0: RTC\_TAMP1 input rising edge triggers a tamper detection event.  
1: RTC\_TAMP1 input falling edge triggers a tamper detection event.
- Bit 0 **TAMP1E**: RTC\_TAMP1 input detection enable  
0: RTC\_TAMP1 detection disabled  
1: RTC\_TAMP1 detection enabled

**Caution:** When TAMPFLT = 0, TAMPxE must be reset when TAMPxTRG is changed to avoid spuriously setting TAMPxF.

### 25.7.16 RTC alarm A sub second register (RTC\_ALRMASSR)

This register can be written only when ALRAE is reset in RTC\_CR register, or in initialization mode.

This register is write protected. The write access procedure is described in [RTC register write protection on page 597](#)

Address offset: 0x44

RTC domain reset value: 0x0000 0000

System reset: not affected

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

Bits 31:28 Reserved, must be kept at reset value.

Bits 27:24 MASKSS[3:0]: Mask the most-significant bits starting at this bit

0: No comparison on sub seconds for Alarm A. The alarm is set when the seconds unit is incremented (assuming that the rest of the fields match).

1: SS[14:1] are don't care in Alarm A comparison. Only SS[0] is compared.

2: SS[14:2] are don't care in Alarm A comparison. Only SS[1:0] are compared.

3: SS[14:3] are don't care in Alarm A comparison. Only SS[2:0] are compared.

...

12: SS[14:12] are don't care in Alarm A comparison. SS[11:0] are compared.

13: SS[14:13] are don't care in Alarm A comparison. SS[12:0] are compared.

14: SS[14] is don't care in Alarm A comparison. SS[13:0] are compared.

15: All 15 SS bits are compared and must match to activate alarm.

The overflow bits of the synchronous counter (bits 15) is never compared. This bit can be different from 0 only after a shift operation.

Bits 23:15 Reserved, must be kept at reset value.

Bits 14:0 SS[14:0]: Sub seconds value

This value is compared with the contents of the synchronous prescaler counter to determine if Alarm A is to be activated. Only bits 0 up MASKSS-1 are compared.

### 25.7.17 RTC backup registers (RTC\_BKPxR)

Address offset: 0x50 to 0x60

RTC domain reset value: 0x0000 0000

System reset: not affected

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

Bits 31:0 BKP[31:0]

The application can write or read data to and from these registers.

They are powered-on by V<sub>BAT</sub> when V<sub>DD</sub> is switched off, so that they are not reset by System reset, and their contents remain valid when the device operates in low-power mode. This register is reset on a tamper detection event, as long as TAMPxF=1.

### 25.7.18 RTC register map

Table 85. RTC register map and reset values

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x00	RTC_TR	Res.	Res.	Res.	Res.	Res.	Res.	Res.									
	Reset value									0	0	0	0	0	0	0	0
0x04	RTC_DR	Res.	Res.	Res.	Res.	Res.	Res.	Res.									
	Reset value								0	0	0	0	0	0	0	0	0
0x08	RTC_CR	Res.	COE	OSEL[1:0]	HT[1:0]	PM	22	21	20	19							
	Reset value								0	0	0	0	0	0	0	0	0
0x0C	RTC_ISR	Res.	Res.	COSEL	17	16	15	14									
	Reset value								0	0	0	0	0	0	0	0	0
0x10	RTC_PRER	Res.	Res.	RECALDF	13	12	11	10									
	Reset value								1	1	1	1	1	0	0	0	0
0x14	RTC_WUTR	Res.	Res.	TSOVF	9	8	7	6									
	Reset value								1	1	1	1	1	1	1	1	1

Table 85. RTC register map and reset values (continued)

0x1C	<b>RTC_ALRMAR</b>																							
		Reset value	0	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
0x24	<b>RTC_WPR</b>																							
		Reset value	0	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
0x28	<b>RTC_SSR</b>																							
		Reset value	0	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
0x2C	<b>RTC_SHIFTR</b>																							
		Reset value	0	ADD1S	0	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
0x30	<b>RTC_TSTR</b>																							
		Reset value	0	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
0x34	<b>RTC_TSDR</b>																							
		Reset value	0	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
0x38	<b>RTC_TSSSR</b>																							
		Reset value	0	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
0x3C	<b>RTC_CALR</b>																							
		Reset value	0	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
0x40	<b>RTC_TAFCR</b>																							
		Reset value	0	PC15MODE	0	PC15VALUE	0	PC14MODE	0	PC14VALUE	0	PC13MODE	0	PC13VALUE	0	WDU[1:0]	0	MNT[2:0]	0	MNU[3:0]	0	ST[2:0]	0	SU[3:0]
0x44	<b>RTC_ALMASSR</b>																							
		Reset value	0	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
0x4C	<b>RTC_OR</b>																							
		Reset value	0	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

**Table 85. RTC register map and reset values (continued)**

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x50 to 0x60	RTC_BKP0R																																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	to RTC_BKP4R																																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 26 Inter-integrated circuit (I2C) interface

### 26.1 Introduction

The I<sup>2</sup>C (inter-integrated circuit) bus interface handles communications between the microcontroller and the serial I<sup>2</sup>C bus. It provides multimaster capability, and controls all I<sup>2</sup>C bus-specific sequencing, protocol, arbitration and timing. It supports Standard-mode (Sm), Fast-mode (Fm) and Fast-mode Plus (Fm+).

It is also SMBus (system management bus) and PMBus® (power management bus) compatible.

DMA can be used to reduce CPU overload.

### 26.2 I2C main features

- I<sup>2</sup>C bus specification rev03 compatibility:
  - Slave and master modes
  - Multimaster capability
  - Standard-mode (up to 100 kHz)
  - Fast-mode (up to 400 kHz)
  - Fast-mode Plus (up to 1 MHz)
  - 7-bit and 10-bit addressing mode
  - Multiple 7-bit slave addresses (2 addresses, 1 with configurable mask)
  - All 7-bit addresses acknowledge mode
  - General call
  - Programmable setup and hold times
  - Easy to use event management
  - Optional clock stretching
  - Software reset
- 1-byte buffer with DMA capability
- Programmable analog and digital noise filters

The following additional features are also available, depending on the product implementation (see [Section 26.3: I2C implementation](#)):

- SMBus specification rev 3.0 compatibility:
  - Hardware PEC (packet error checking) generation and verification with ACK control
  - Command and data acknowledge control
  - Address resolution protocol (ARP) support
  - Host and device support
  - SMBus alert
  - Timeouts and idle condition detection
- PMBus rev 1.3 standard compatibility
- Independent clock: a choice of independent clock sources allowing the I2C communication speed to be independent from the PCLK reprogramming

- Wake-up from Stop mode on address match.

## 26.3 I2C implementation

This manual describes the full set of features implemented in I2C1. I2C2 supports a smaller set of features, but is otherwise identical to I2C1. The differences are listed below.

**Table 86. STM32F0xx I2C implementation**

I2C features <sup>(1)</sup>	STM32F03x STM32F04x		STM32F05x		STM32F07x STM32F09x	
	I2C1	I2C1	I2C2	I2C1	I2C2	
7-bit addressing mode	X	X	X	X	X	
10-bit addressing mode	X	X	X	X	X	
Standard mode (up to 100 kbit/s)	X	X	X	X	X	
Fast-mode (up to 400 kbit/s)	X	X	X	X	X	
Fast-mode Plus with 20 mA output drive I/Os (up to 1 Mbit/s)	X	X	-	X	X	
Independent clock	X	X	-	X	-	
Wake-up from Stop mode	X	X	-	X	-	
SMBus/PMBus	X	X	-	X	-	

1. X = supported.

## 26.4 I2C functional description

In addition to receiving and transmitting data, this interface converts them from serial to parallel format and vice versa. The interrupts are enabled or disabled by software. The interface is connected to the I<sup>2</sup>C bus by a data pin (SDA) and by a clock pin (SCL). It can be connected with a standard (up to 100 kHz), Fast-mode (up to 400 kHz) or Fast-mode Plus (up to 1 MHz) I<sup>2</sup>C bus.

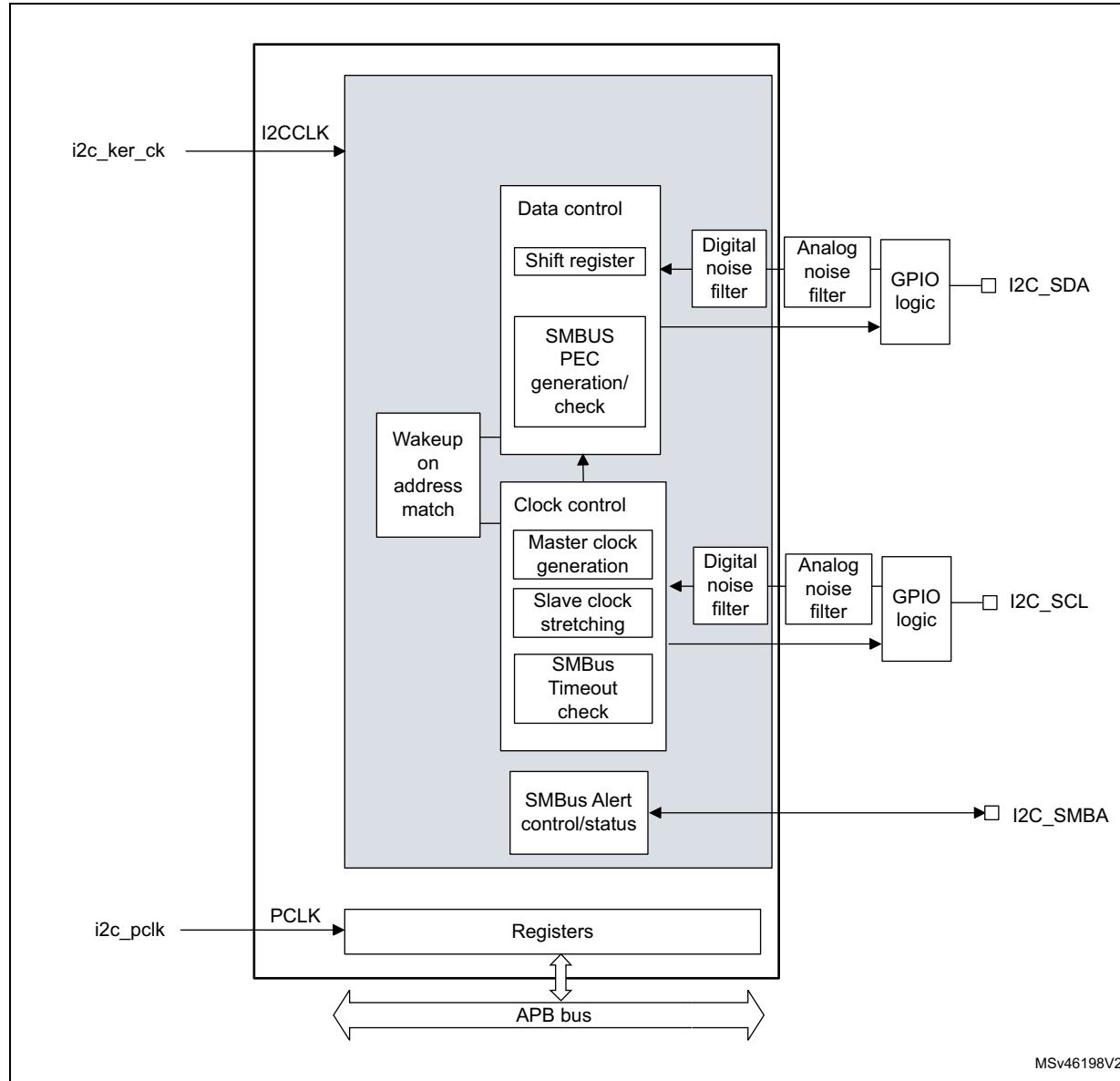
This interface can also be connected to an SMBus with the data pin (SDA) and clock pin (SCL).

If SMBus feature is supported, the additional optional SMBus Alert pin (SMBA) is also available.

### 26.4.1 I2C1 block diagram

The block diagram of the I2C1 interface is shown in [Figure 217](#).

**Figure 217. I2C1 block diagram**



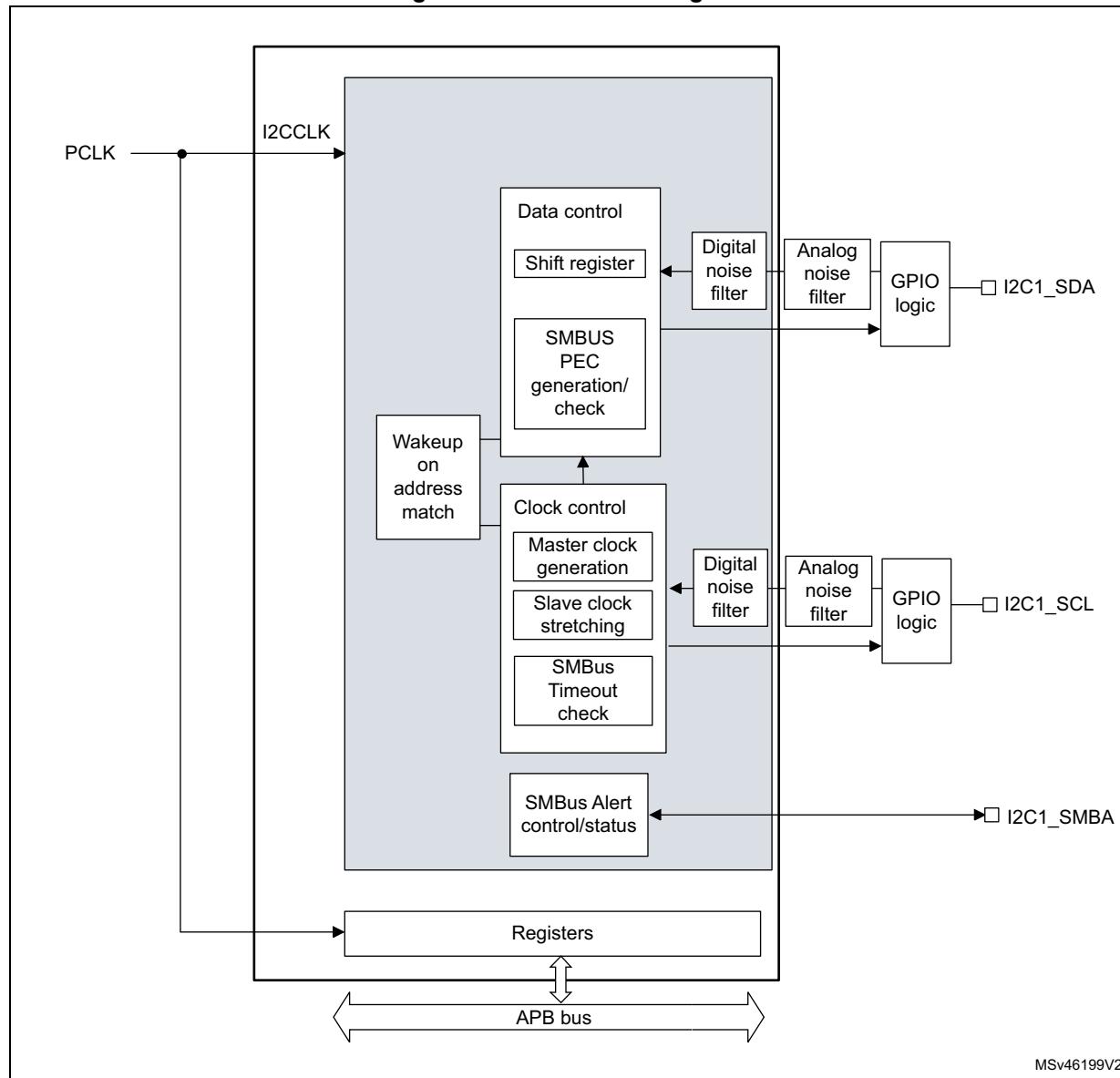
The I<sub>2</sub>C1 is clocked by an independent clock source, which allows the I<sub>2</sub>C to operate independently from the PCLK frequency.

For I<sub>2</sub>C I/Os supporting 20 mA output current drive for Fast-mode Plus operation, the driving capability is enabled through control bits in the system configuration controller (SYSCFG). Refer to [Section 26.3: I<sub>2</sub>C implementation](#).

### 26.4.2 I2C2 block diagram

The block diagram of the I<sub>2</sub>C2 interface is shown in [Figure 218](#).

Figure 218. I2C2 block diagram



### 26.4.3 I2C pins and internal signals

Table 87. I2C input/output pins

Pin name	Signal type	Description
I2C_SDA	Bidirectional	I2C data
I2C_SCL	Bidirectional	I2C clock
I2C_SMBA	Bidirectional	SMBus alert

**Table 88. I2C internal input/output signals**

Internal signal name	Signal type	Description
i2c_ker_ck	Input	I2C kernel clock, also named I2CCLK in this document
i2c_pclk	Input	I2C APB clock
i2c_it	Output	I2C interrupts, refer to <a href="#">Table 102</a> for the full list of interrupt sources
i2c_rx_dma	Output	I2C receive data DMA request (I2C_RX)
i2c_tx_dma	Output	I2C transmit data DMA request (I2C_TX)

#### 26.4.4 I2C clock requirements

The I2C kernel is clocked by I2CCLK.

The I2CCLK period  $t_{I2CCLK}$  must respect the following conditions:

$$t_{I2CCLK} < (t_{LOW} - t_{filters}) / 4 \text{ and } t_{I2CCLK} < t_{HIGH}$$

with:

$t_{LOW}$ : SCL low time and  $t_{HIGH}$ : SCL high time

$t_{filters}$ : when enabled, sum of the delays brought by the analog filter and by the digital filter.

Analog filter delay is maximum 260 ns. Digital filter delay is DNF x  $t_{I2CCLK}$ .

The PCLK clock period  $t_{PCLK}$  must respect the following condition:

$$t_{PCLK} < 4 / 3 t_{SCL}$$

with  $t_{SCL}$ : SCL period

**Caution:** When the I2C kernel is clocked by PCLK, this clock must respect the conditions for  $t_{I2CCLK}$ .

#### 26.4.5 Mode selection

The interface can operate in one of the four following modes:

- Slave transmitter
- Slave receiver
- Master transmitter
- Master receiver

By default, it operates in slave mode. The interface automatically switches from slave to master when it generates a START condition, and from master to slave if an arbitration loss or a STOP generation occurs, allowing multimaster capability.

#### Communication flow

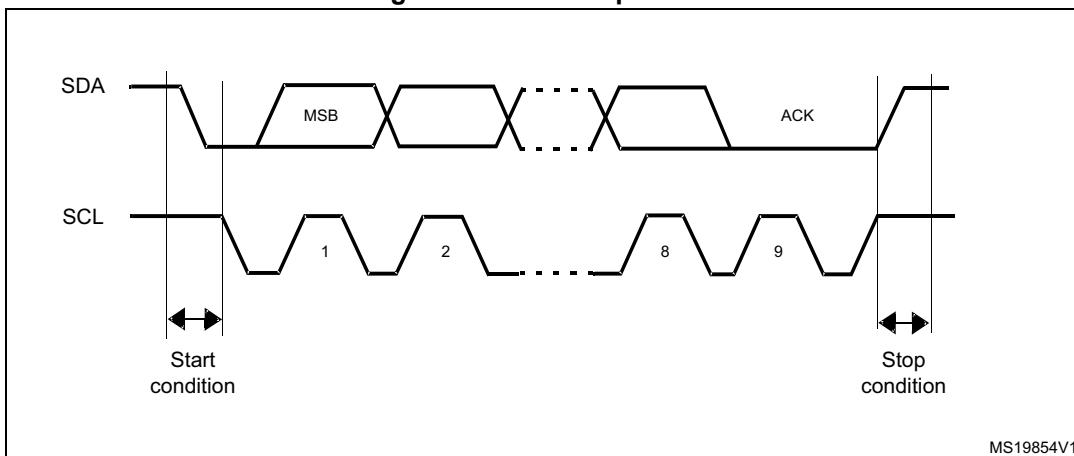
In Master mode, the I2C interface initiates a data transfer and generates the clock signal. A serial data transfer always begins with a START condition and ends with a STOP condition. Both START and STOP conditions are generated in master mode by software.

In Slave mode, the interface is capable of recognizing its own addresses (7- or 10-bit), and the general call address. The general call address detection can be enabled or disabled by software. The reserved SMBus addresses can also be enabled 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 ninth clock pulse follows the eight clock cycles of a byte transfer, during which the receiver must send an acknowledge bit to the transmitter (see [Figure 219](#)).

**Figure 219. I<sup>2</sup>C bus protocol**



Acknowledge can be enabled or disabled by software. The I<sup>2</sup>C interface addresses can be selected by software.

## 26.4.6 I<sup>2</sup>C initialization

### Enabling and disabling the peripheral

The I<sup>2</sup>C peripheral clock must be configured and enabled in the clock controller, then the I<sup>2</sup>C can be enabled by setting the PE bit in the I<sup>2</sup>C\_CR1 register.

When the I<sup>2</sup>C is disabled (PE = 0), the I<sup>2</sup>C performs a software reset. Refer to [Section 26.4.7](#) for more details.

### Noise filters

Before enabling the I<sup>2</sup>C peripheral by setting the PE bit in I<sup>2</sup>C\_CR1 register, the user must configure the noise filters, if needed. By default, an analog noise filter is present on the SDA and SCL inputs. This analog filter is compliant with the I<sup>2</sup>C specification, which requires the suppression of spikes with a pulse width up to 50 ns in Fast-mode and Fast-mode Plus. The user can disable this analog filter by setting the ANFOFF bit, and/or select a digital filter by configuring the DNF[3:0] bit in the I<sup>2</sup>C\_CR1 register.

When the digital filter is enabled, the level of the SCL or the SDA line is internally changed only if it remains stable for more than DNF x I<sup>2</sup>CCLK periods. This allows spikes with a programmable length of 1 to 15 I<sup>2</sup>CCLK periods to be suppressed.

**Table 89. Comparison of analog vs. digital filters**

-	Analog filter	Digital filter
Pulse width of suppressed spikes	$\geq 50$ ns	Programmable length from 1 to 15 I2C peripheral clocks
Benefits	Available in Stop mode	<ul style="list-style-type: none"><li>– Programmable length: extra filtering capability versus standard requirements</li><li>– Stable length</li></ul>
Drawbacks	Variation vs. temperature, voltage, process	Wake-up from Stop mode on address match is not available when digital filter is enabled

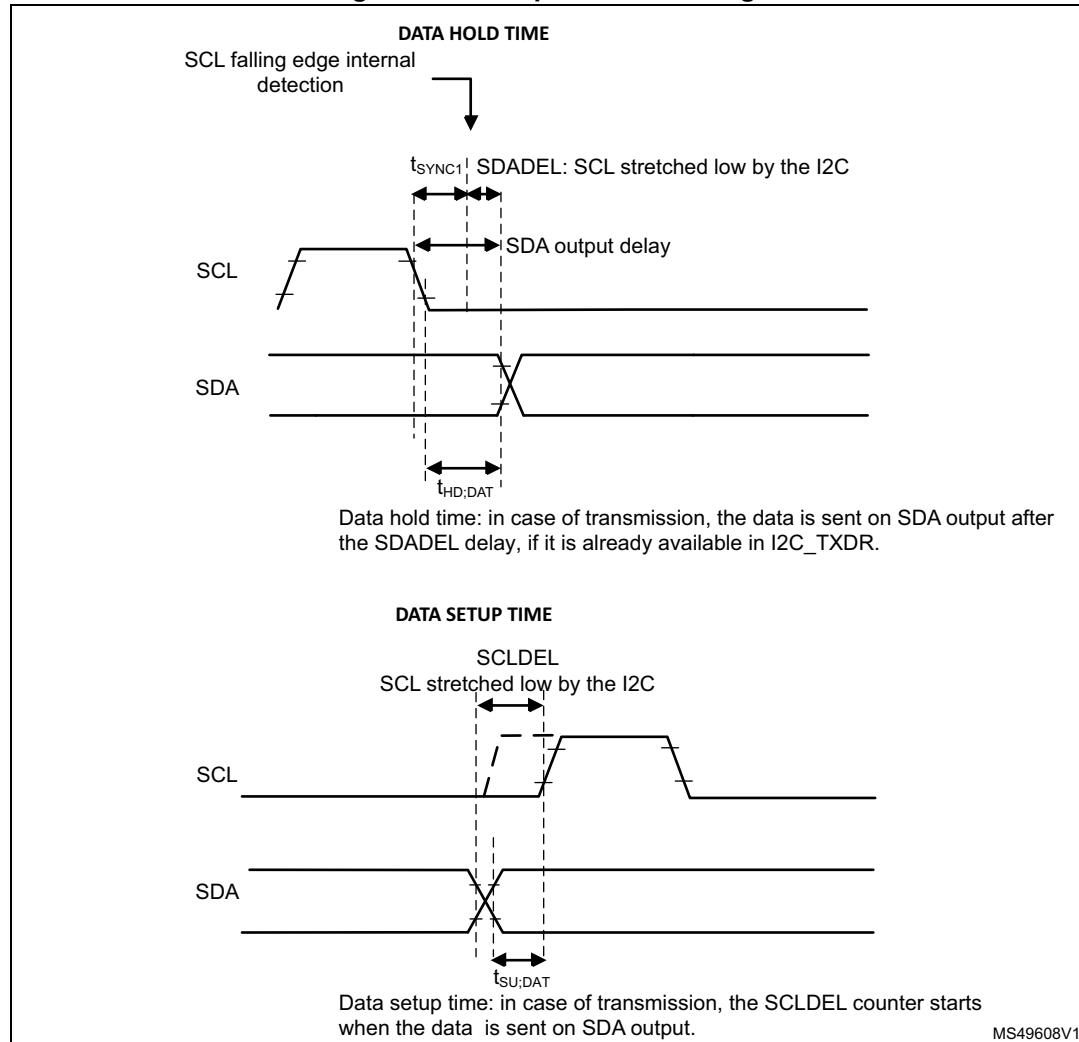
**Caution:** Changing the filter configuration is not allowed when the I2C is enabled.

## I2C timings

The timings must be configured in order to guarantee correct data hold and setup times, used in master and slave modes. This is done by programming the PRESC[3:0], SCLDEL[3:0] and SDADEL[3:0] bits in the I2C\_TIMINGR register.

The STM32CubeMX tool calculates and provides the I2C\_TIMINGR content in the I2C configuration window.

**Figure 220. Setup and hold timings**



- When the SCL falling edge is internally detected, a delay is inserted before sending SDA output. This delay is  $t_{SDADEL} = SDADEL \times t_{PRESC} + t_{I2CCLK}$  where  $t_{PRESC} = (PRESC+1) \times t_{I2CCLK}$ .  
 $t_{SDADEL}$  impacts the hold time  $t_{HD;DAT}$ .

The total SDA output delay is:

$$t_{SYNC1} + \{[SDADEL \times (PRESC+1) + 1] \times t_{I2CCLK}\}$$

$t_{SYNC1}$  duration depends on these parameters:

- SCL falling slope
- When enabled, input delay brought by the analog filter:  $t_{AF(min)} < t_{AF} < t_{AF(max)}$
- When enabled, input delay brought by the digital filter:  $t_{DNF} = DNF \times t_{I2CCLK}$
- Delay due to SCL synchronization to I2CCLK clock (two to three I2CCLK periods)

In order to bridge the undefined region of the SCL falling edge, the user must program SDADEL in such a way that:

$$\{t_f(\max) + t_{HD;DAT}(\min) - t_{AF(min)} - [(DNF+3) \times t_{I2CCLK}]\} / \{(PRESC+1) \times t_{I2CCLK}\} \leq SDADEL$$

$$SDADEL \leq \{t_{HD;DAT}(\max) - t_{AF(max)} - [(DNF+4) \times t_{I2CCLK}]\} / \{(PRESC+1) \times t_{I2CCLK}\}$$

*Note:*  $t_{AF(min)} / t_{AF(max)}$  are part of the equation only when the analog filter is enabled. Refer to device datasheet for  $t_{AF}$  values.

The maximum  $t_{HD;DAT}$  can be 3.45 µs, 0.9 µs and 0.45 µs for Standard-mode, Fast-mode and Fast-mode Plus, but must be less than the maximum of  $t_{VD;DAT}$  by a transition time. This maximum must only be met if the device does not stretch the LOW period ( $t_{LOW}$ ) of the SCL signal. If the clock stretches the SCL, the data must be valid by the set-up time before it releases the clock.

The SDA rising edge is usually the worst case, so in this case the previous equation becomes:

$$SDADEL \leq \{t_{VD;DAT}(\max) - t_r(\max) - 260\text{ ns} - [(DNF+4) \times t_{I2CCLK}]\} / \{(PRESC+1) \times t_{I2CCLK}\}.$$

*Note:* This condition can be violated when  $NOSTRETCH = 0$ , because the device stretches SCL low to guarantee the set-up time, according to the SCLDEL value.

Refer to [Table 90](#) for  $t_f$ ,  $t_r$ ,  $t_{HD;DAT}$  and  $t_{VD;DAT}$  standard values.

- After  $t_{SDADEL}$  delay, or after sending SDA output in case the slave had to stretch the clock because the data was not yet written in I2C\_TXDR register, SCL line is kept at low level during the setup time. This setup time is  $t_{SCLDEL} = (SCLDEL+1) \times t_{PRESC}$  where  $t_{PRESC} = (PRESC+1) \times t_{I2CCLK}$ .  
 $t_{SCLDEL}$  impacts the setup time  $t_{SU;DAT}$ .

In order to bridge the undefined region of the SDA transition (rising edge usually worst case), the user must program SCLDEL in such a way that:

$$\{[t_r(\max) + t_{SU;DAT}(\min)] / [(PRESC+1) \times t_{I2CCLK}]\} - 1 \leq SCLDEL$$

Refer to [Table 90](#) for  $t_r$  and  $t_{SU;DAT}$  standard values.

The SDA and SCL transition time values to be used are the ones in the application. Using the maximum values from the standard increases the constraints for the SDADEL and SCLDEL calculation, but ensures the feature whatever the application.

*Note:* At every clock pulse, after SCL falling edge detection, the I2C master or slave stretches SCL low during at least  $[(SDADEL + SCLDEL + 1) \times (PRESC + 1) + 1] \times t_{I2CCLK}$ , in both

*transmission and reception modes. In transmission mode, if the data is not yet written in I2C\_TXDR when SDADEL counter is finished, the I2C keeps on stretching SCL low until the next data is written. Then new data MSB is sent on SDA output, and SCLDEL counter starts, continuing stretching SCL low to guarantee the data setup time.*

If NOSTRETCH = 1 in slave mode, the SCL is not stretched. Consequently the SDADEL must be programmed in such a way to guarantee also a sufficient setup time.

**Table 90. I<sup>2</sup>C-SMBus specification data setup and hold times**

Symbol	Parameter	Standard-mode (Sm)		Fast-mode (Fm)		Fast-mode Plus (Fm+)		SMBus		Unit
		Min.	Max	Min.	Max	Min.	Max	Min.	Max	
t <sub>HD;DAT</sub>	Data hold time	0	-	0	-	0	-	0.3	-	μs
t <sub>VD;DAT</sub>	Data valid time	-	3.45	-	0.9	-	0.45	-	-	μs
t <sub>SU;DAT</sub>	Data setup time	250	-	100	-	50	-	250	-	ns
t <sub>r</sub>	Rise time of both SDA and SCL signals	-	1000	-	300	-	120	-	1000	ns
t <sub>f</sub>	Fall time of both SDA and SCL signals	-	300	-	300	-	120	-	300	ns

Additionally, in master mode, the SCL clock high and low levels must be configured by programming the PRESC[3:0], SCLH[7:0] and SCLL[7:0] bits in the I2C\_TIMINGR register.

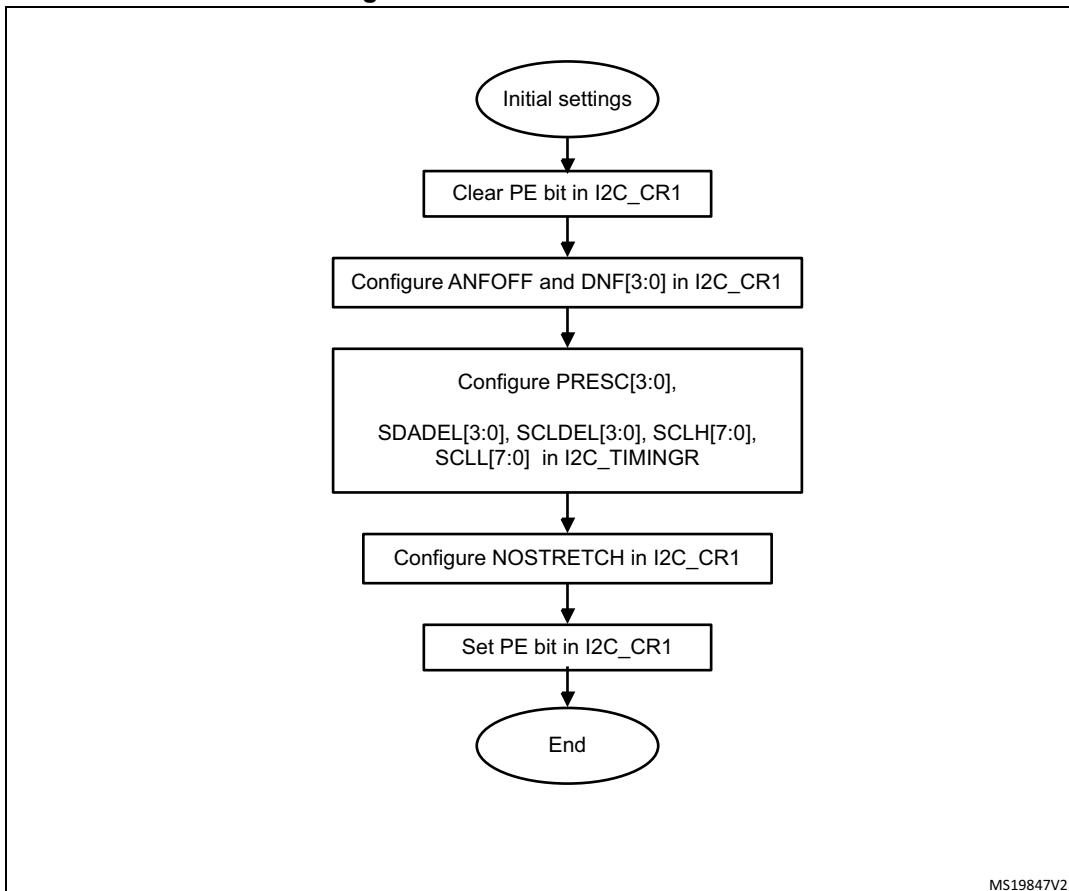
- When the SCL falling edge is internally detected, a delay is inserted before releasing the SCL output.  
This delay is  $t_{SCLL} = (SCLL + 1) \times t_{PRESC}$  where  $t_{PRESC} = (PRESC + 1) \times t_{I2CCLK}$ .  
 $t_{SCLL}$  impacts the SCL low time  $t_{LOW}$ .
- When the SCL rising edge is internally detected, a delay is inserted before forcing the SCL output to low level. This delay is  $t_{SCLH} = (SCLH + 1) \times t_{PRESC}$ , where  $t_{PRESC} = (PRESC+1) \times t_{I2CCLK}$ .  $t_{SCLH}$  impacts the SCL high time  $t_{HIGH}$ .

Refer to [I<sup>2</sup>C master initialization](#) for more details.

**Caution:** Changing the timing configuration is not allowed when the I2C is enabled.

The I2C slave NOSTRETCH mode must also be configured before enabling the peripheral.  
Refer to [I<sup>2</sup>C slave initialization](#) for more details.

**Caution:** Changing the NOSTRETCH configuration is not allowed when the I2C is enabled.

**Figure 221. I2C initialization flow**

#### 26.4.7 Software reset

A software reset can be performed by clearing the PE bit in the I2C\_CR1 register. In that case I2C lines SCL and SDA are released. Internal states machines are reset and communication control bits, as well as status bits come back to their reset value. The configuration registers are not impacted.

Here is the list of impacted register bits:

1. I2C\_CR2 register: START, STOP, NACK
2. I2C\_ISR register: BUSY, TXE, TXIS, RXNE, ADDR, NACKF, TCR, TC, STOPF, BERR, ARLO, OVR

and in addition when the SMBus feature is supported:

1. I2C\_CR2 register: PECBYTE
2. I2C\_ISR register: PECERR, TIMEOUT, ALERT

PE must be kept low during at least three APB clock cycles in order to perform the software reset. This is ensured by writing the following software sequence:

1. Write PE = 0
2. Check PE = 0
3. Write PE = 1

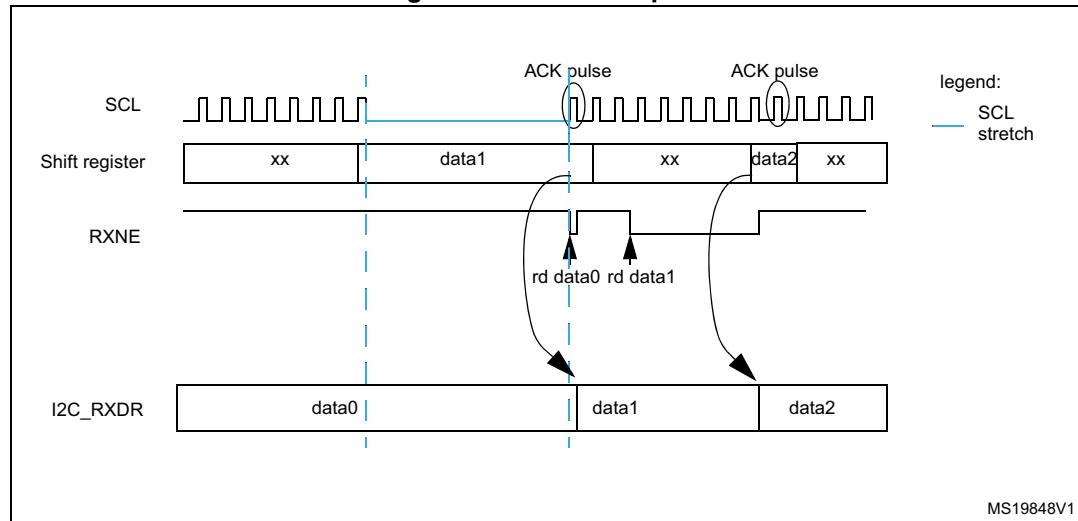
### 26.4.8 Data transfer

The data transfer is managed through transmit and receive data registers and a shift register.

#### Reception

The SDA input fills the shift register. After the eighth SCL pulse (when the complete data byte is received), the shift register is copied into I2C\_RXDR register if it is empty (RXNE = 0). If RXNE = 1, meaning that the previous received data byte has not yet been read, the SCL line is stretched low until I2C\_RXDR is read. The stretch is inserted between the eighth and ninth SCL pulse (before the acknowledge pulse).

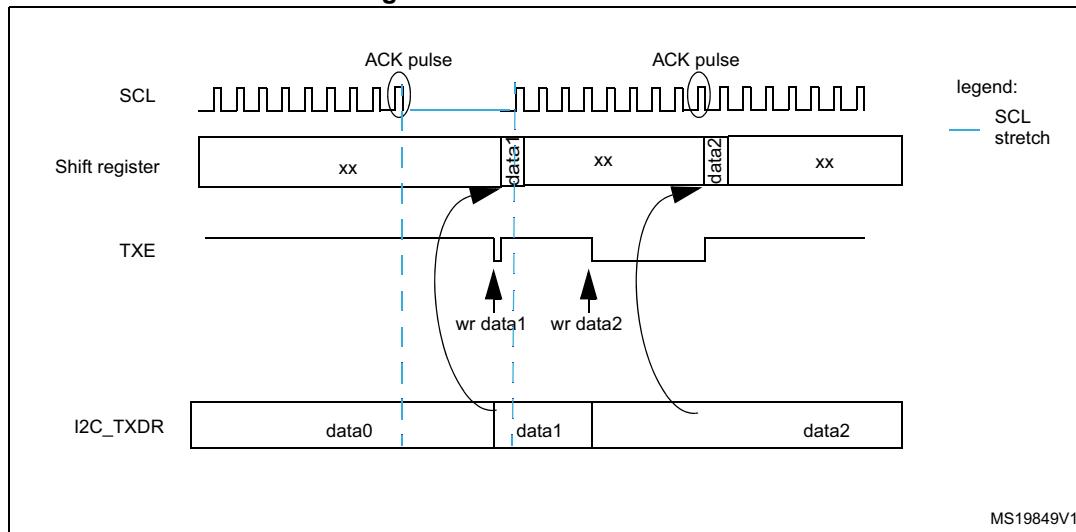
Figure 222. Data reception



## Transmission

If the I2C\_TXDR register is not empty (TXE=0), its content is copied into the shift register after the ninth SCL pulse (the acknowledge pulse). Then the shift register content is shifted out on SDA line. If TXE = 1, meaning that no data is written yet in I2C\_TXDR, SCL line is stretched low until I2C\_TXDR is written. The stretch is done after the ninth SCL pulse.

**Figure 223. Data transmission**



## Hardware transfer management

The I2C has a byte counter embedded in hardware in order to manage byte transfer and to close the communication in various modes such as:

- NACK, STOP and ReSTART generation in master mode
- ACK control in slave receiver mode
- PEC generation/checking when SMBus feature is supported

The byte counter is always used in master mode. By default it is disabled in slave mode, but it can be enabled by software by setting the SBC (slave byte control) bit in the I2C\_CR1 register.

The number of bytes to be transferred is programmed in the NBYTES[7:0] bit field in the I2C\_CR2 register. If the number of bytes to be transferred (NBYTES) is greater than 255, or if a receiver wants to control the acknowledge value of a received data byte, the reload mode must be selected by setting the RELOAD bit in the I2C\_CR2 register. In this mode, the TCR flag is set when the number of bytes programmed in NBYTES is transferred, and an interrupt is generated if TCIE is set. SCL is stretched as long as TCR flag is set. TCR is cleared by software when NBYTES is written to a non-zero value.

When the NBYTES counter is reloaded with the last number of bytes, RELOAD bit must be cleared.

When RELOAD=0 in master mode, the counter can be used in two modes:

- **Automatic end mode** (AUTOEND = '1' in the I2C\_CR2 register). In this mode, the master automatically sends a STOP condition once the number of bytes programmed in the NBYTES[7:0] bit field is transferred.
- **Software end mode** (AUTOEND = '0' in the I2C\_CR2 register). In this mode, software action is expected once the number of bytes programmed in the NBYTES[7:0] bit field is transferred; the TC flag is set and an interrupt is generated if the TCIE bit is set. The SCL signal is stretched as long as the TC flag is set. The TC flag is cleared by software when the START or STOP bit is set in the I2C\_CR2 register. This mode must be used when the master wants to send a RESTART condition.

**Caution:** The AUTOEND bit has no effect when the RELOAD bit is set.

**Table 91. I2C configuration**

Function	SBC bit	RELOAD bit	AUTOEND bit
Master Tx/Rx NBYTES + STOP	x	0	1
Master Tx/Rx + NBYTES + RESTART	x	0	0
Slave Tx/Rx all received bytes ACKed	0	x	x
Slave Rx with ACK control	1	1	x

## 26.4.9 I2C slave mode

### I2C slave initialization

In order to work in slave mode, the user must enable at least one slave address. Two registers I2C\_OAR1 and I2C\_OAR2 are available in order to program the slave own addresses OA1 and OA2.

- OA1 can be configured either in 7-bit mode (by default) or in 10-bit addressing mode by setting the OA1MODE bit in the I2C\_OAR1 register.  
OA1 is enabled by setting the OA1EN bit in the I2C\_OAR1 register.
- If additional slave addresses are required, the second slave address OA2 can be configured. Up to 7 OA2 LSB can be masked by configuring the OA2MSK[2:0] bits in the I2C\_OAR2 register. Therefore for OA2MSK configured from 1 to 6, only OA2[7:2], OA2[7:3], OA2[7:4], OA2[7:5], OA2[7:6] or OA2[7] are compared with the received address. As soon as OA2MSK is not equal to 0, the address comparator for OA2 excludes the I2C reserved addresses (0000 XXX and 1111 XXX), which are not acknowledged. If OA2MSK=7, all received 7-bit addresses are acknowledged (except reserved addresses). OA2 is always a 7-bit address.

These reserved addresses can be acknowledged if they are enabled by the specific enable bit, if they are programmed in the I2C\_OAR1 or I2C\_OAR2 register with OA2MSK=0.

OA2 is enabled by setting the OA2EN bit in the I2C\_OAR2 register.

- The general call address is enabled by setting the GCEN bit in the I2C\_CR1 register.

When the I2C is selected by one of its enabled addresses, the ADDR interrupt status flag is set, and an interrupt is generated if the ADDRIE bit is set.

By default, the slave uses its clock stretching capability, which means that it stretches the SCL signal at low level when needed, in order to perform software actions. If the master does not support clock stretching, the I2C must be configured with NOSTRETCH = 1 in the I2C\_CR1 register.

After receiving an ADDR interrupt, if several addresses are enabled the user must read the ADDCODE[6:0] bits in the I2C\_ISR register in order to check which address matched. DIR flag must also be checked in order to know the transfer direction.

### Slave clock stretching (NOSTRETCH = 0)

In default mode, the I2C slave stretches the SCL clock in the following situations:

- When the ADDR flag is set: the received address matches with one of the enabled slave addresses. This stretch is released when the ADDR flag is cleared by software setting the ADDRCF bit.
- In transmission, if the previous data transmission is completed and no new data is written in I2C\_TXDR register, or if the first data byte is not written when the ADDR flag is cleared (TXE = 1). This stretch is released when the data is written to the I2C\_TXDR register.
- In reception when the I2C\_RXDR register is not read yet and a new data reception is completed. This stretch is released when I2C\_RXDR is read.
- When TCR = 1 in Slave Byte Control mode, reload mode (SBC=1 and RELOAD=1), meaning that the last data byte has been transferred. This stretch is released when then TCR is cleared by writing a non-zero value in the NBYTES[7:0] field.
- After SCL falling edge detection, the I2C stretches SCL low during  $[(SDADEL+SCLDEL+1) \times (PRESC+1) + 1] \times t_{I2CCLK}$ .

### Slave without clock stretching (NOSTRETCH = 1)

When NOSTRETCH = 1 in the I2C\_CR1 register, the I2C slave does not stretch the SCL signal.

- The SCL clock is not stretched while the ADDR flag is set.
- In transmission, the data must be written in the I2C\_TXDR register before the first SCL pulse corresponding to its transfer occurs. If not, an underrun occurs, the OVR flag is set in the I2C\_ISR register and an interrupt is generated if the ERRIE bit is set in the I2C\_CR1 register. The OVR flag is also set when the first data transmission starts and the STOPF bit is still set (has not been cleared). Therefore, if the user clears the STOPF flag of the previous transfer only after writing the first data to be transmitted in the next transfer, he ensures that the OVR status is provided, even for the first data to be transmitted.
- In reception, the data must be read from the I2C\_RXDR register before the ninth SCL pulse (ACK pulse) of the next data byte occurs. If not an overrun occurs, the OVR flag is set in the I2C\_ISR register and an interrupt is generated if the ERRIE bit is set in the I2C\_CR1 register.

### Slave byte control mode

In order to allow byte ACK control in slave reception mode, The Slave byte control mode must be enabled by setting the SBC bit in the I2C\_CR1 register. This is required to be compliant with SMBus standards.

The Reload mode must be selected in order to allow byte ACK control in slave reception mode (RELOAD = 1). To get control of each byte, NBYTES must be initialized to 0x1 in the ADDR interrupt subroutine, and reloaded to 0x1 after each received byte. When the byte is received, the TCR bit is set, stretching the SCL signal low between the eighth and ninth SCL pulses. The user can read the data from the I2C\_RXDR register, and then decide to acknowledge it or not by configuring the ACK bit in the I2C\_CR2 register. The SCL stretch is released by programming NBYTES to a non-zero value: the acknowledge or not-acknowledge is sent and next byte can be received.

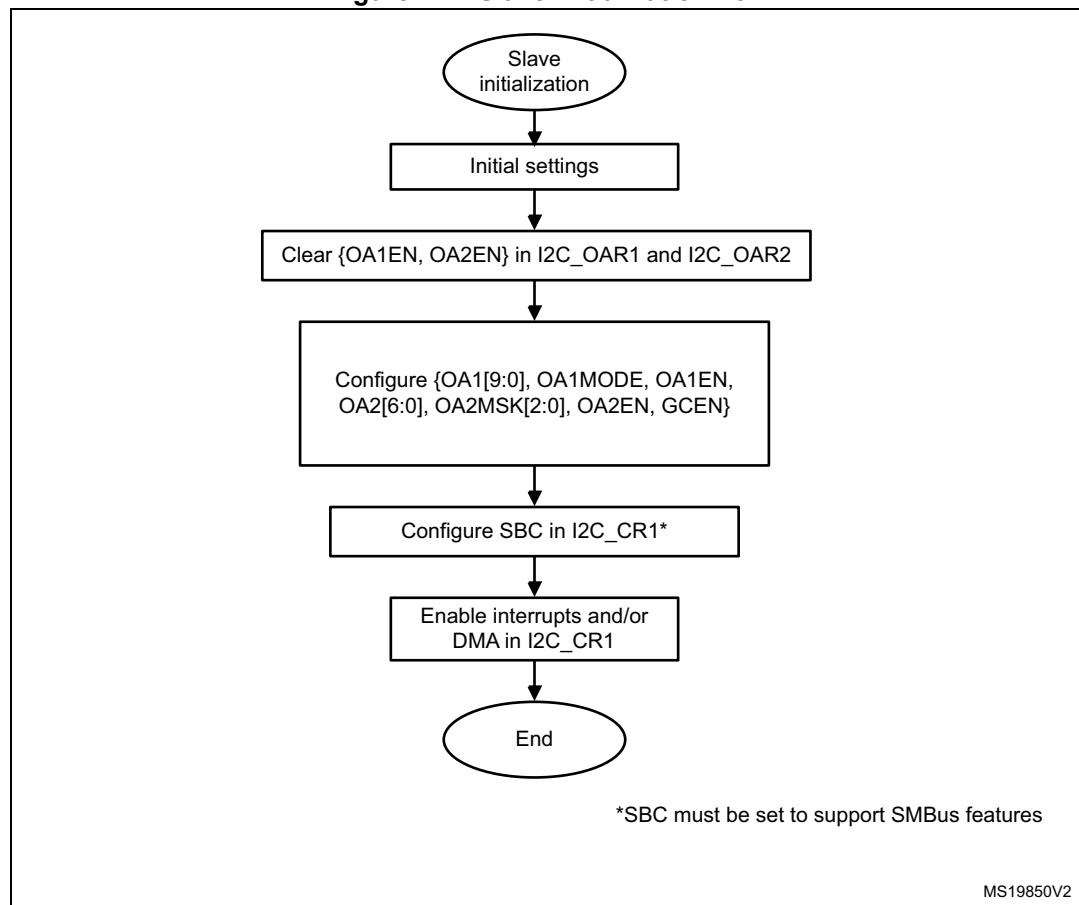
NBYTES can be loaded with a value greater than 0x1, and in this case, the reception flow is continuous during NBYTES data reception.

**Note:** *The SBC bit must be configured when the I2C is disabled, or when the slave is not addressed, or when ADDR = 1.*

*The RELOAD bit value can be changed when ADDR = 1, or when TCR = 1.*

**Caution:** The Slave byte control mode is not compatible with NOSTRETCH mode. Setting SBC when NOSTRETCH = 1 is not allowed.

**Figure 224. Slave initialization flow**



For a code example refer to [A.14.3: I2C configured in slave mode code example](#).

### Slave transmitter

A transmit interrupt status (TXIS) is generated when the I2C\_TXDR register becomes empty. An interrupt is generated if the TXIE bit is set in the I2C\_CR1 register.

The TXIS bit is cleared when the I2C\_TXDR register is written with the next data byte to be transmitted.

When a NACK is received, the NACKF bit is set in the I2C\_ISR register and an interrupt is generated if the NACKIE bit is set in the I2C\_CR1 register. The slave automatically releases the SCL and SDA lines in order to let the master perform a STOP or a RESTART condition. The TXIS bit is not set when a NACK is received.

When a STOP is received and the STOPIE bit is set in the I2C\_CR1 register, the STOPF flag is set in the I2C\_ISR register and an interrupt is generated. In most applications, the SBC bit is usually programmed to '0'. In this case, If TXE = 0 when the slave address is received (ADDR = 1), the user can choose either to send the content of the I2C\_TXDR register as the first data byte, or to flush the I2C\_TXDR register by setting the TXE bit in order to program a new data byte.

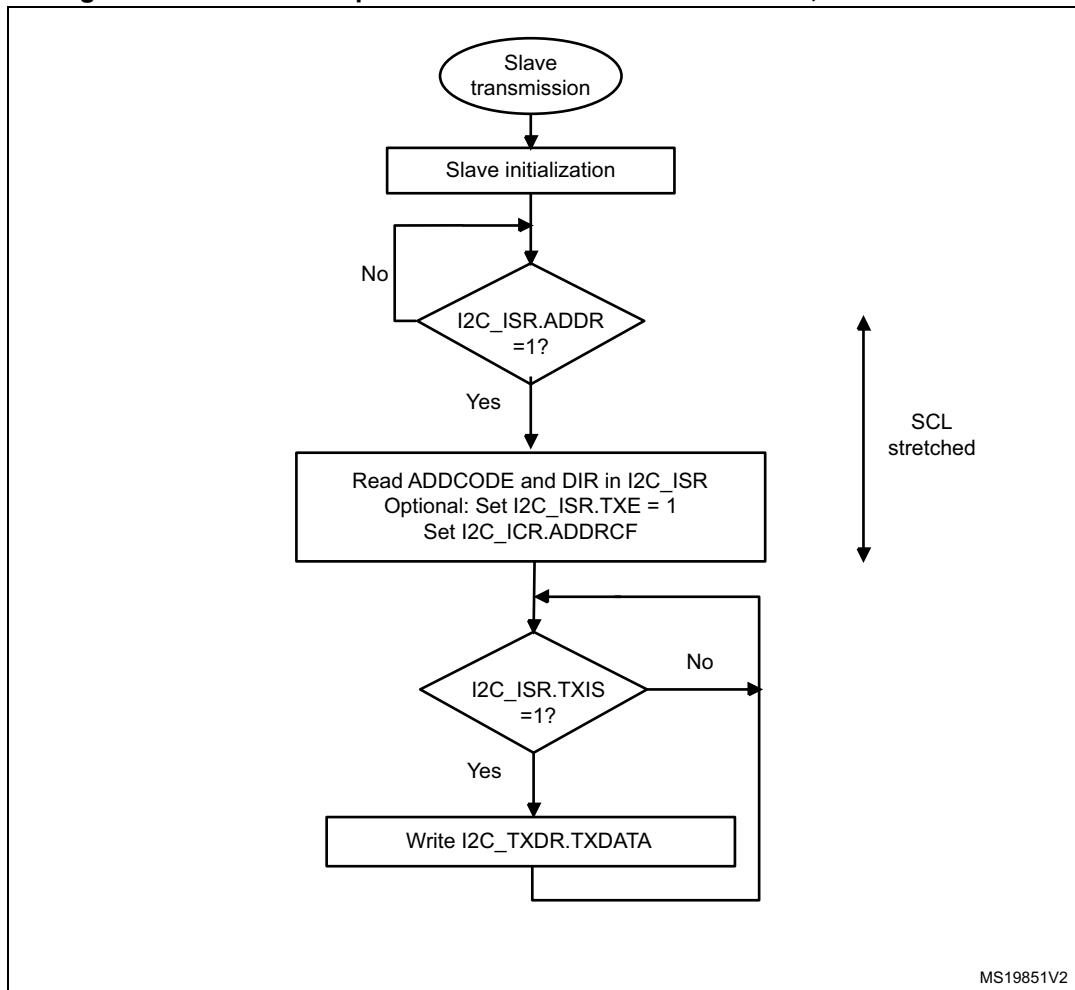
In Slave byte control mode (SBC = 1), the number of bytes to be transmitted must be programmed in NBYTES in the address match interrupt subroutine (ADDR = 1). In this case, the number of TXIS events during the transfer corresponds to the value programmed in NBYTES.

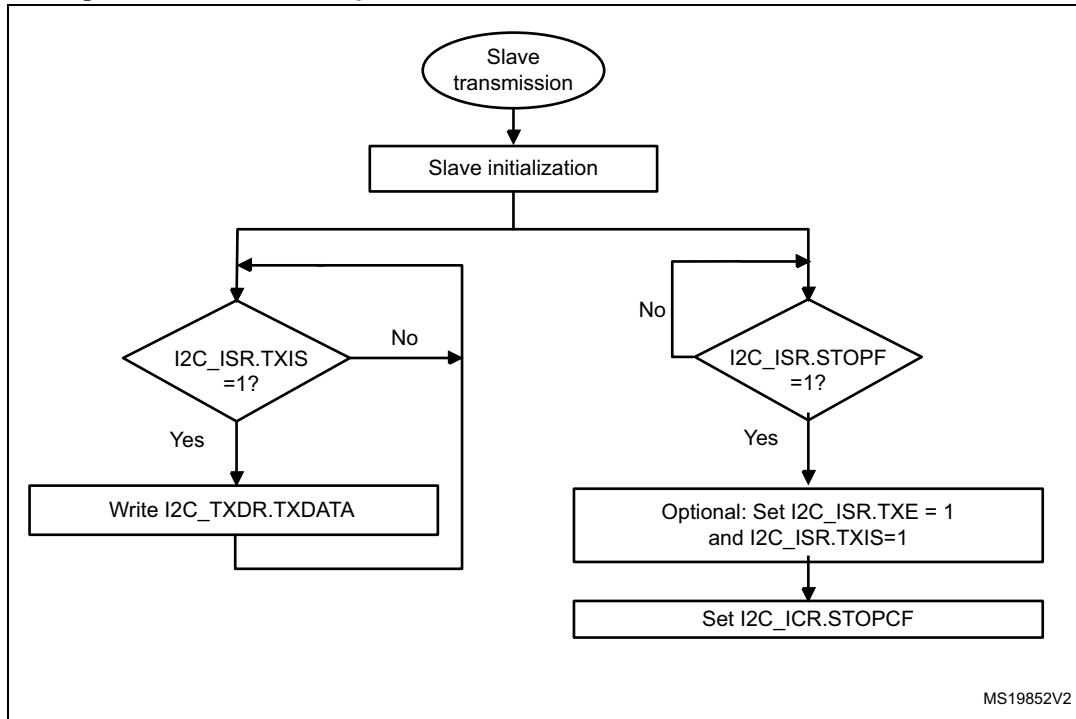
**Caution:** When NOSTRETCH = 1, the SCL clock is not stretched while the ADDR flag is set, so the user cannot flush the I2C\_TXDR register content in the ADDR subroutine, in order to program the first data byte. The first data byte to be sent must be previously programmed in the I2C\_TXDR register:

- This data can be the data written in the last TXIS event of the previous transmission message.
- If this data byte is not the one to be sent, the I2C\_TXDR register can be flushed by setting the TXE bit in order to program a new data byte. The STOPF bit must be cleared only after these actions, in order to guarantee that they are executed before the first data transmission starts, following the address acknowledge.

If STOPF is still set when the first data transmission starts, an underrun error is generated (the OVR flag is set).

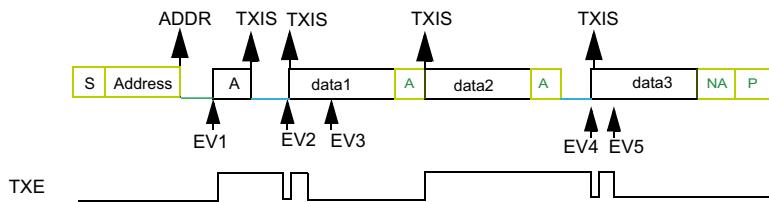
If a TXIS event is needed, (transmit interrupt or transmit DMA request), the user must set the TXIS bit in addition to the TXE bit, in order to generate a TXIS event.

**Figure 225. Transfer sequence flow for I2C slave transmitter, NOSTRETCH = 0**

**Figure 226. Transfer sequence flow for I2C slave transmitter, NOSTRETCH = 1**

**Figure 227. Transfer bus diagrams for I2C slave transmitter (mandatory events only)**

Example I2C slave transmitter 3 bytes with 1st data flushed,  
NOSTRETCH=0:

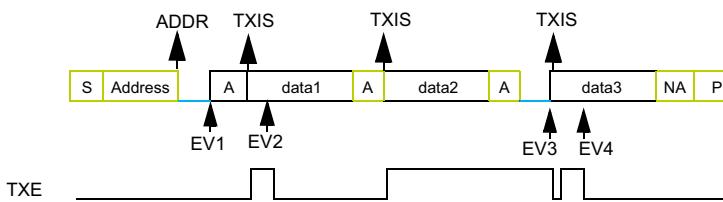


legend:

- transmission
- reception
- SCL stretch

- EV1: ADDR ISR: check ADDCODE and DIR, set TXE, set ADDRCF  
EV2: TXIS ISR: wr data1  
EV3: TXIS ISR: wr data2  
EV4: TXIS ISR: wr data3  
EV5: TXIS ISR: wr data4 (not sent)

Example I2C slave transmitter 3 bytes without 1st data flush,  
NOSTRETCH=0:

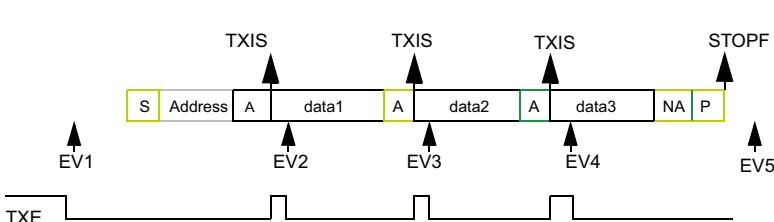


legend :

- transmission
- reception
- SCL stretch

- EV1: ADDR ISR: check ADDCODE and DIR, set ADDRCF  
EV2: TXIS ISR: wr data2  
EV3: TXIS ISR: wr data3  
EV4: TXIS ISR: wr data4 (not sent)

Example I2C slave transmitter 3 bytes, NOSTRETCH=1:



legend:

- transmission
- reception
- SCL stretch

- EV1: wr data1  
EV2: TXIS ISR: wr data2  
EV3: TXIS ISR: wr data3  
EV4: TXIS ISR: wr data4 (not sent)  
EV5: STOPF ISR: (optional: set TXE and TXIS), set STOPCF

MS19853V2

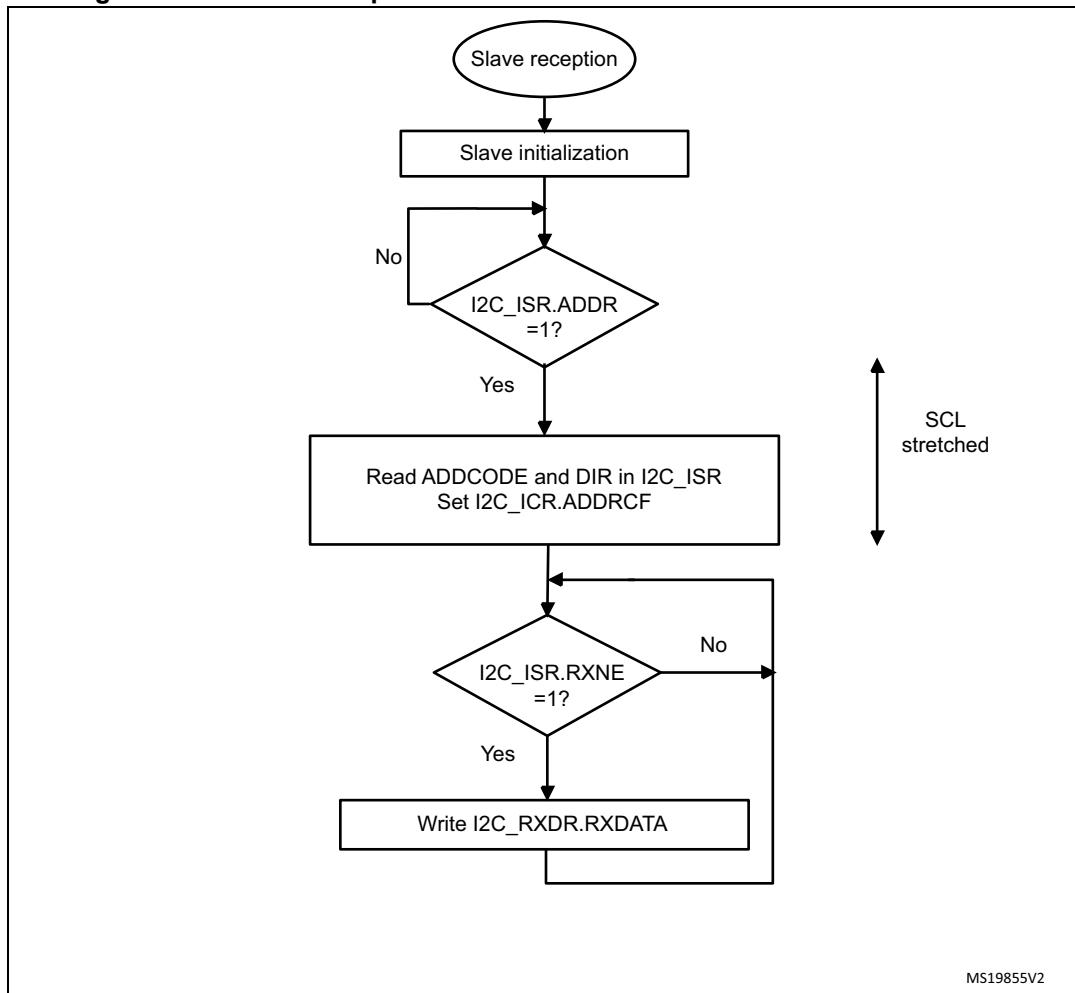
For a code example refer to [A.14.6: I2C slave transmitter code example](#).

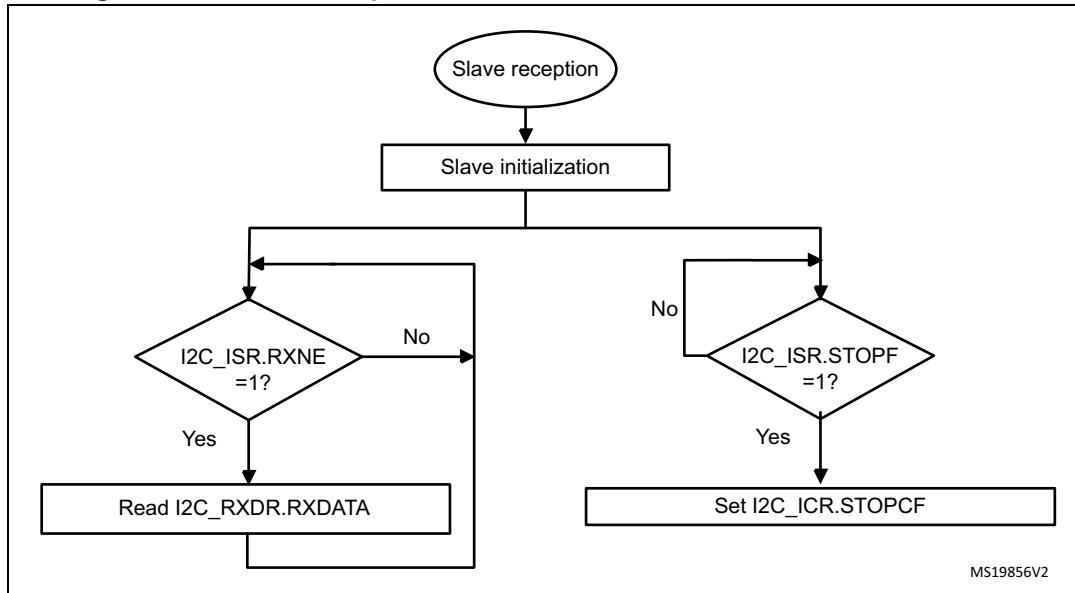
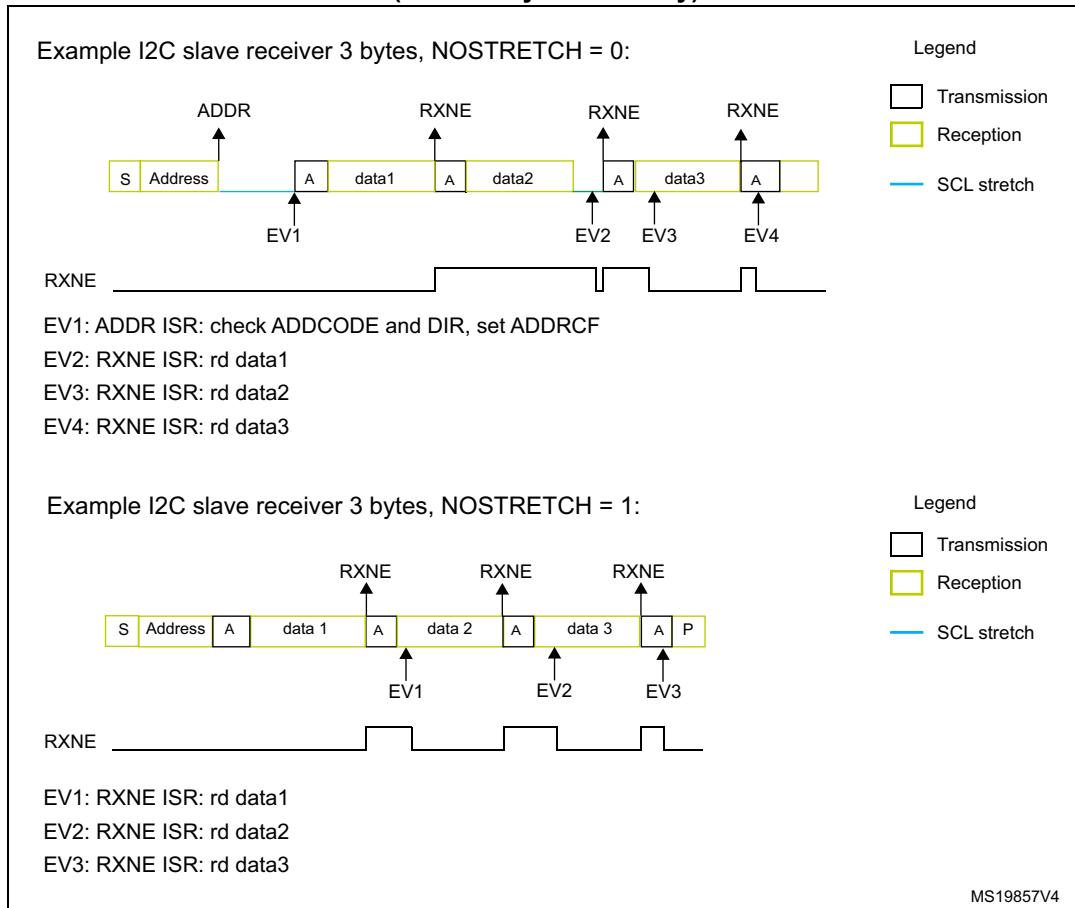
### Slave receiver

RXNE is set in I2C\_ISR when the I2C\_RXDR is full, and generates an interrupt if RXIE is set in I2C\_CR1. RXNE is cleared when I2C\_RXDR is read.

When a STOP is received and STOPIE is set in I2C\_CR1, STOPF is set in I2C\_ISR and an interrupt is generated.

**Figure 228. Transfer sequence flow for slave receiver with NOSTRETCH = 0**



**Figure 229. Transfer sequence flow for slave receiver with NOSTRETCH = 1****Figure 230. Transfer bus diagrams for I2C slave receiver  
(mandatory events only)**

For a code example refer to [A.14.7: I2C slave receiver code example](#).

## 26.4.10 I2C master mode

### I2C master initialization

Before enabling the peripheral, the I2C master clock must be configured by setting the SCLH and SCLL bits in the I2C\_TIMINGR register.

The STM32CubeMX tool calculates and provides the I2C\_TIMINGR content in the I2C Configuration window.

A clock synchronization mechanism is implemented in order to support multi-master environment and slave clock stretching.

In order to allow clock synchronization:

- The low level of the clock is counted using the SCLL counter, starting from the SCL low level internal detection.
- The high level of the clock is counted using the SCLH counter, starting from the SCL high level internal detection.

The I2C detects its own SCL low level after a  $t_{SYNC1}$  delay depending on the SCL falling edge, SCL input noise filters (analog + digital) and SCL synchronization to the I2CxCLK clock. The I2C releases SCL to high level once the SCLL counter reaches the value programmed in the SCLL[7:0] bits in the I2C\_TIMINGR register.

The I2C detects its own SCL high level after a  $t_{SYNC2}$  delay depending on the SCL rising edge, SCL input noise filters (analog + digital) and SCL synchronization to I2CxCLK clock. The I2C ties SCL to low level once the SCLH counter is reached reaches the value programmed in the SCLH[7:0] bits in the I2C\_TIMINGR register.

Consequently the master clock period is:

$$t_{SCL} = t_{SYNC1} + t_{SYNC2} + \{[(SCLH+1) + (SCLL+1)] \times (PRESC+1) \times t_{I2CCLK}\}$$

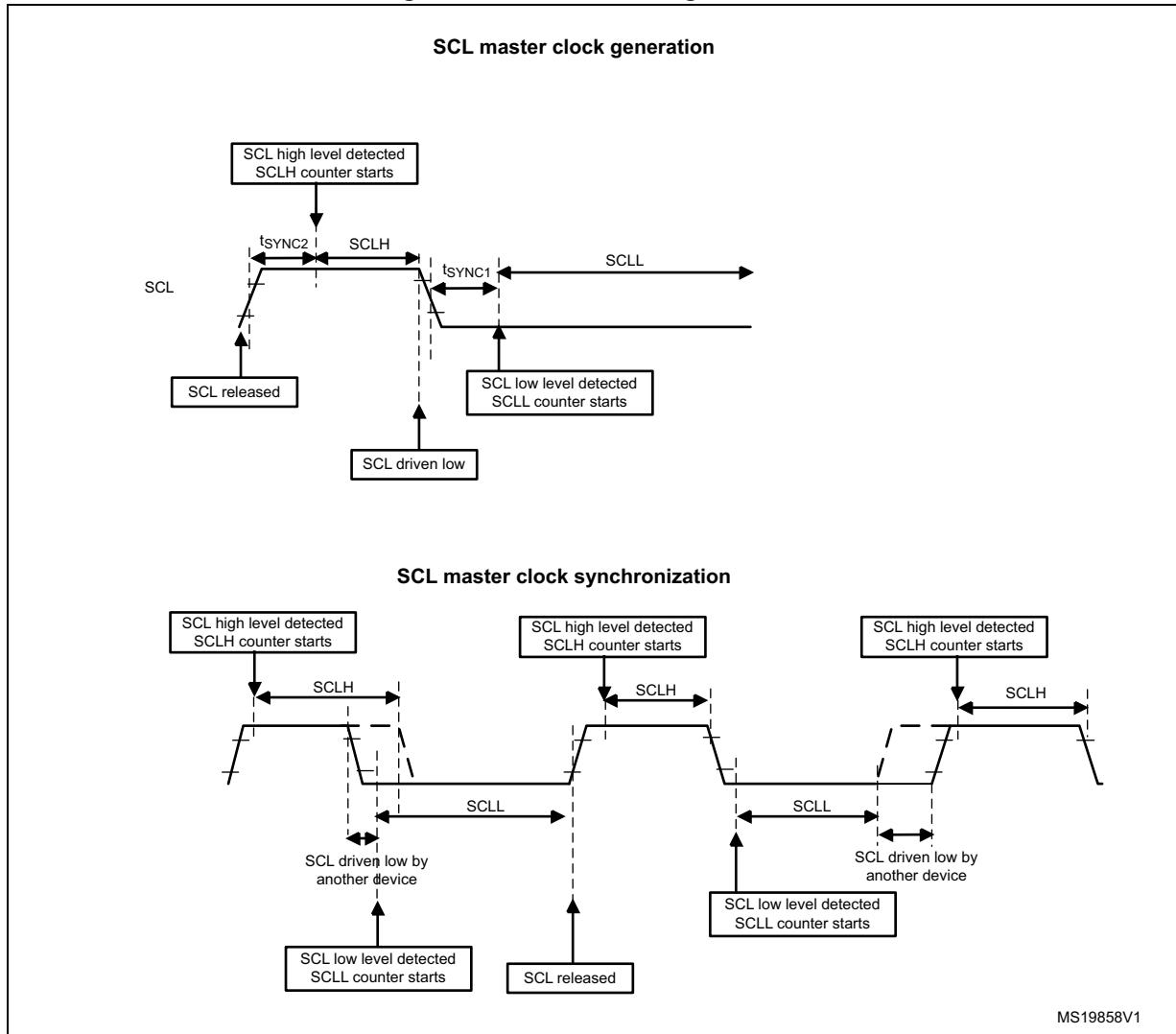
The duration of  $t_{SYNC1}$  depends on these parameters:

- SCL falling slope
- When enabled, input delay induced by the analog filter.
- When enabled, input delay induced by the digital filter: DNF  $\times t_{I2CCLK}$
- Delay due to SCL synchronization with I2CCLK clock (two to three I2CCLK periods)

The duration of  $t_{SYNC2}$  depends on these parameters:

- SCL rising slope
- When enabled, input delay induced by the analog filter.
- When enabled, input delay induced by the digital filter: DNF  $\times t_{I2CCLK}$
- Delay due to SCL synchronization with I2CCLK clock (two to three I2CCLK periods)

Figure 231. Master clock generation



**Caution:** To be I<sup>2</sup>C or SMBus compliant, the master clock must respect the timings given in the table below.

**Table 92. I<sup>2</sup>C-SMBus specification clock timings**

Symbol	Parameter	Standard-mode (Sm)		Fast-mode (Fm)		Fast-mode Plus (Fm+)		SMBus		Unit
		Min	Max	Min	Max	Min	Max	Min	Max	
f <sub>SCL</sub>	SCL clock frequency	-	100	-	400	-	1000	-	100	kHz
t <sub>HD:STA</sub>	Hold time (repeated) START condition	4.0	-	0.6	-	0.26	-	4.0	-	μs
t <sub>SU:STA</sub>	Set-up time for a repeated START condition	4.7	-	0.6	-	0.26	-	4.7	-	
t <sub>SU:STO</sub>	Set-up time for STOP condition	4.0	-	0.6	-	0.26	-	4.0	-	
t <sub>BUF</sub>	Bus free time between a STOP and START condition	4.7	-	1.3	-	0.5	-	4.7	-	
t <sub>LOW</sub>	Low period of the SCL clock	4.7	-	1.3	-	0.5	-	4.7	-	
t <sub>HIGH</sub>	Period of the SCL clock	4.0	-	0.6	-	0.26	-	4.0	50	
t <sub>r</sub>	Rise time of both SDA and SCL signals	-	1000	-	300	-	120	-	1000	ns
t <sub>f</sub>	Fall time of both SDA and SCL signals	-	300	-	300	-	120	-	300	

**Note:** SCLL is also used to generate the t<sub>BUF</sub> and t<sub>SU:STA</sub> timings.

SCLH is also used to generate the t<sub>HD:STA</sub> and t<sub>SU:STO</sub> timings.

Refer to [Section 26.4.11: I2C\\_TIMINGR register configuration examples](#) for examples of I2C\_TIMINGR settings vs. I2CCLK frequency.

### Master communication initialization (address phase)

In order to initiate the communication, the user must program the following parameters for the addressed slave in the I2C\_CR2 register:

- Addressing mode (7-bit or 10-bit): ADD10
- Slave address to be sent: SADD[9:0]
- Transfer direction: RD\_WRN
- In case of 10-bit address read: HEAD10R bit. HEAD10R must be configure to indicate if the complete address sequence must be sent, or only the header in case of a direction change.
- The number of bytes to be transferred: NBYTES[7:0]. If the number of bytes is equal to or greater than 255 bytes, NBYTES[7:0] must initially be filled with 0xFF.

The user must then set the START bit in I2C\_CR2 register. Changing all the above bits is not allowed when START bit is set.

Then the master automatically sends the START condition followed by the slave address as soon as it detects that the bus is free (BUSY = 0) and after a delay of t<sub>BUF</sub>.

In case of an arbitration loss, the master automatically switches back to slave mode and can acknowledge its own address if it is addressed as a slave.

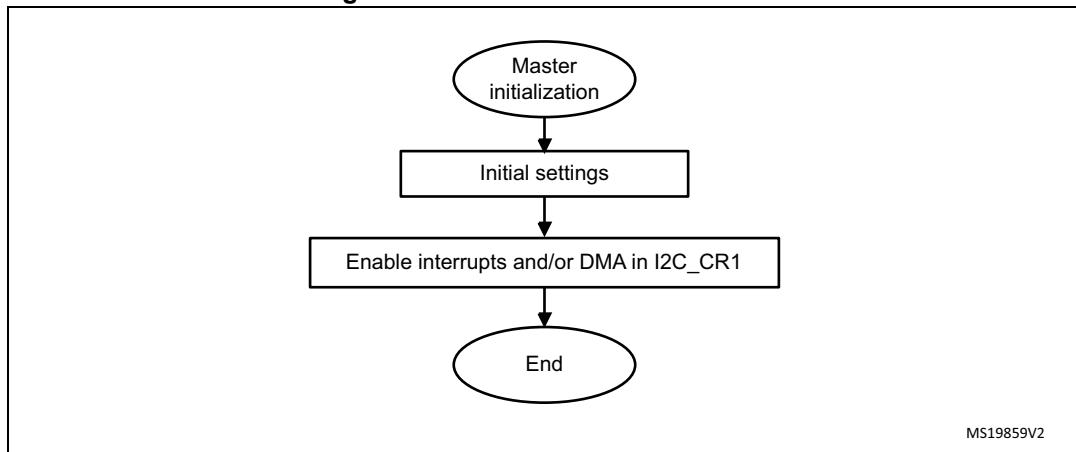
**Note:** The START bit is reset by hardware when the slave address has been sent on the bus, whatever the received acknowledge value. The START bit is also reset by hardware if an arbitration loss occurs.

In 10-bit addressing mode, when the slave address first 7 bits are NACKed by the slave, the master re-launches automatically the slave address transmission until ACK is received. In this case ADDRCF must be set if a NACK is received from the slave, in order to stop sending the slave address.

If the I2C is addressed as a slave (ADDR = 1) while the START bit is set, the I2C switches to slave mode and the START bit is cleared, when the ADDRCF bit is set.

**Note:** The same procedure is applied for a repeated start condition. In this case BUSY = 1.

**Figure 232. Master initialization flow**

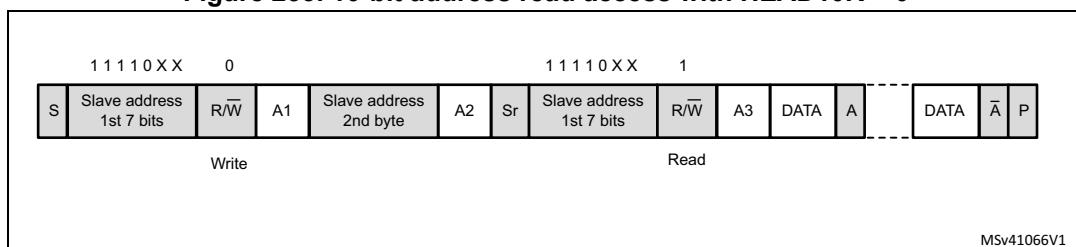


For code examples refer to [A.14.1: I2C configured in master mode to receive code example](#) and [A.14.2: I2C configured in master mode to transmit code example](#).

#### Initialization of a master receiver addressing a 10-bit address slave

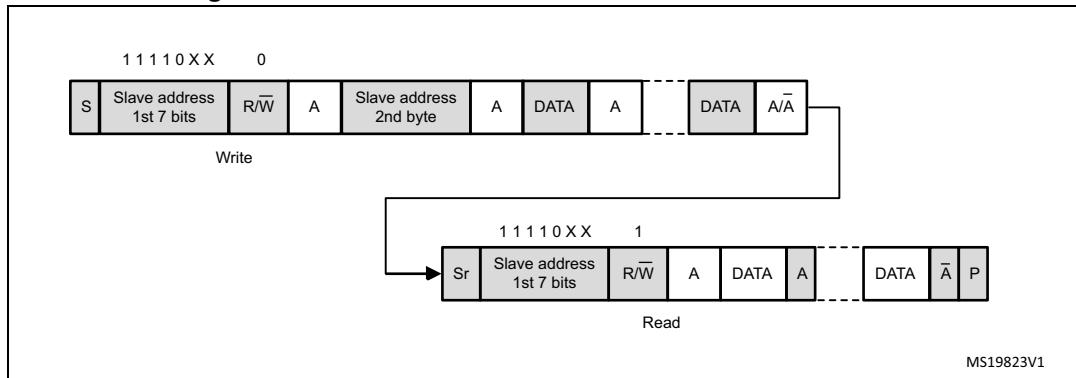
- If the slave address is in 10-bit format, the user can choose to send the complete read sequence by clearing the HEAD10R bit in the I2C\_CR2 register. In this case the master automatically sends the following complete sequence after the START bit is set:  
(Re)Start + Slave address 10-bit header Write + Slave address second byte + REStart + Slave address 10-bit header Read

**Figure 233. 10-bit address read access with HEAD10R = 0**



- If the master addresses a 10-bit address slave, transmits data to this slave and then reads data from the same slave, a master transmission flow must be done first. Then a repeated start is set with the 10 bit slave address configured with HEAD10R = 1. In this case the master sends this sequence: ReStart + Slave address 10-bit header Read.

**Figure 234. 10-bit address read access with HEAD10R = 1**



### Master transmitter

In the case of a write transfer, the TXIS flag is set after each byte transmission, after the ninth SCL pulse when an ACK is received.

A TXIS event generates an interrupt if the TXIE bit is set in the I2C\_CR1 register. The flag is cleared when the I2C\_TXDR register is written with the next data byte to be transmitted.

The number of TXIS events during the transfer corresponds to the value programmed in NBYTES[7:0]. If the total number of data bytes to be sent is greater than 255, reload mode must be selected by setting the RELOAD bit in the I2C\_CR2 register. In this case, when NBYTES data have been transferred, the TCR flag is set and the SCL line is stretched low until NBYTES[7:0] is written to a non-zero value.

The TXIS flag is not set when a NACK is received.

- When RELOAD=0 and NBYTES data have been transferred:
  - In automatic end mode (AUTOEND = 1), a STOP is automatically sent.
  - In software end mode (AUTOEND = 0), the TC flag is set and the SCL line is stretched low in order to perform software actions:
    - A RESTART condition can be requested by setting the START bit in the I2C\_CR2 register with the proper slave address configuration, and number of bytes to be transferred. Setting the START bit clears the TC flag and the START condition is sent on the bus.
    - A STOP condition can be requested by setting the STOP bit in the I2C\_CR2 register. Setting the STOP bit clears the TC flag and the STOP condition is sent on the bus.
- If a NACK is received: the TXIS flag is not set, and a STOP condition is automatically sent after the NACK reception. the NACKF flag is set in the I2C\_ISR register, and an interrupt is generated if the NACKIE bit is set.

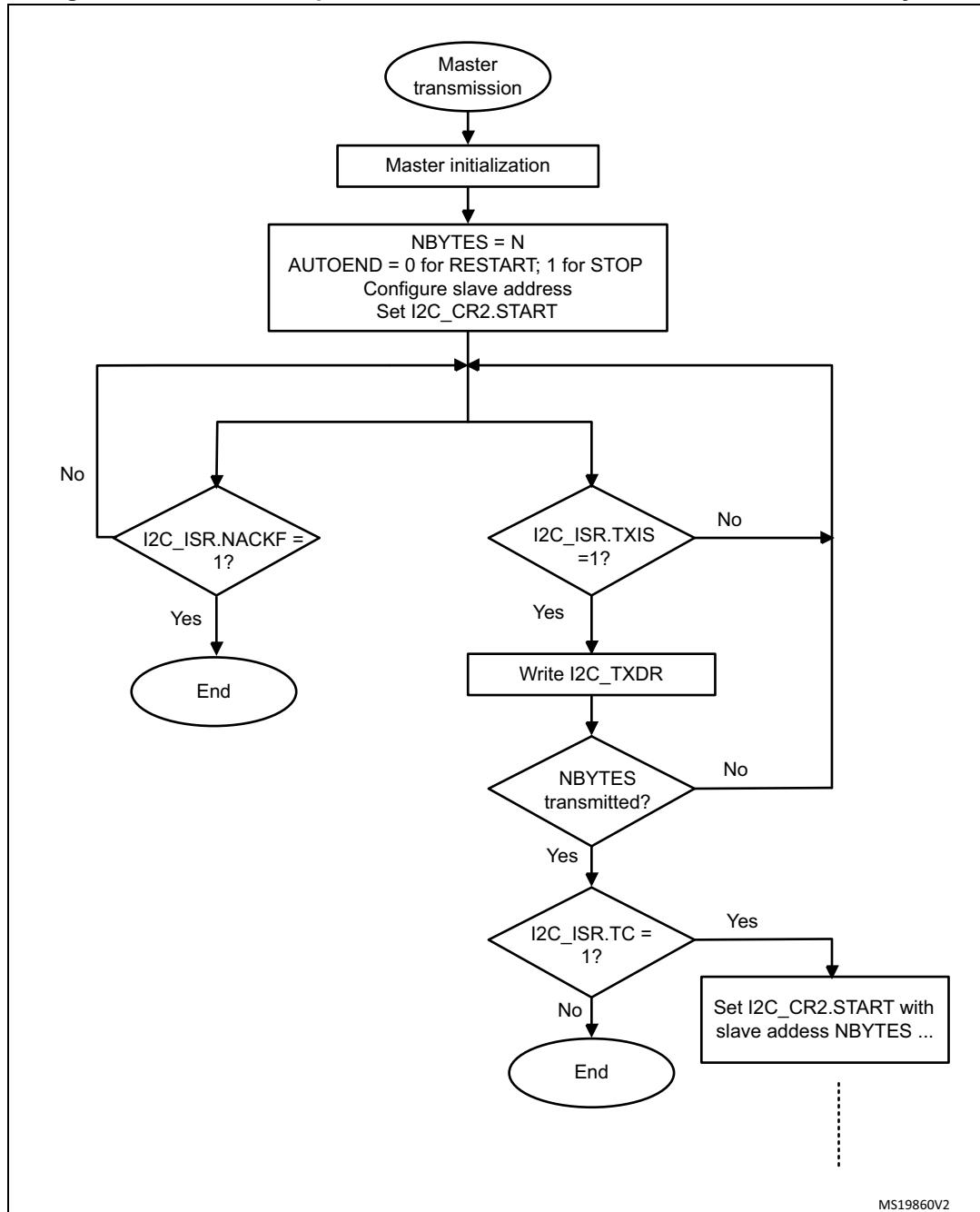
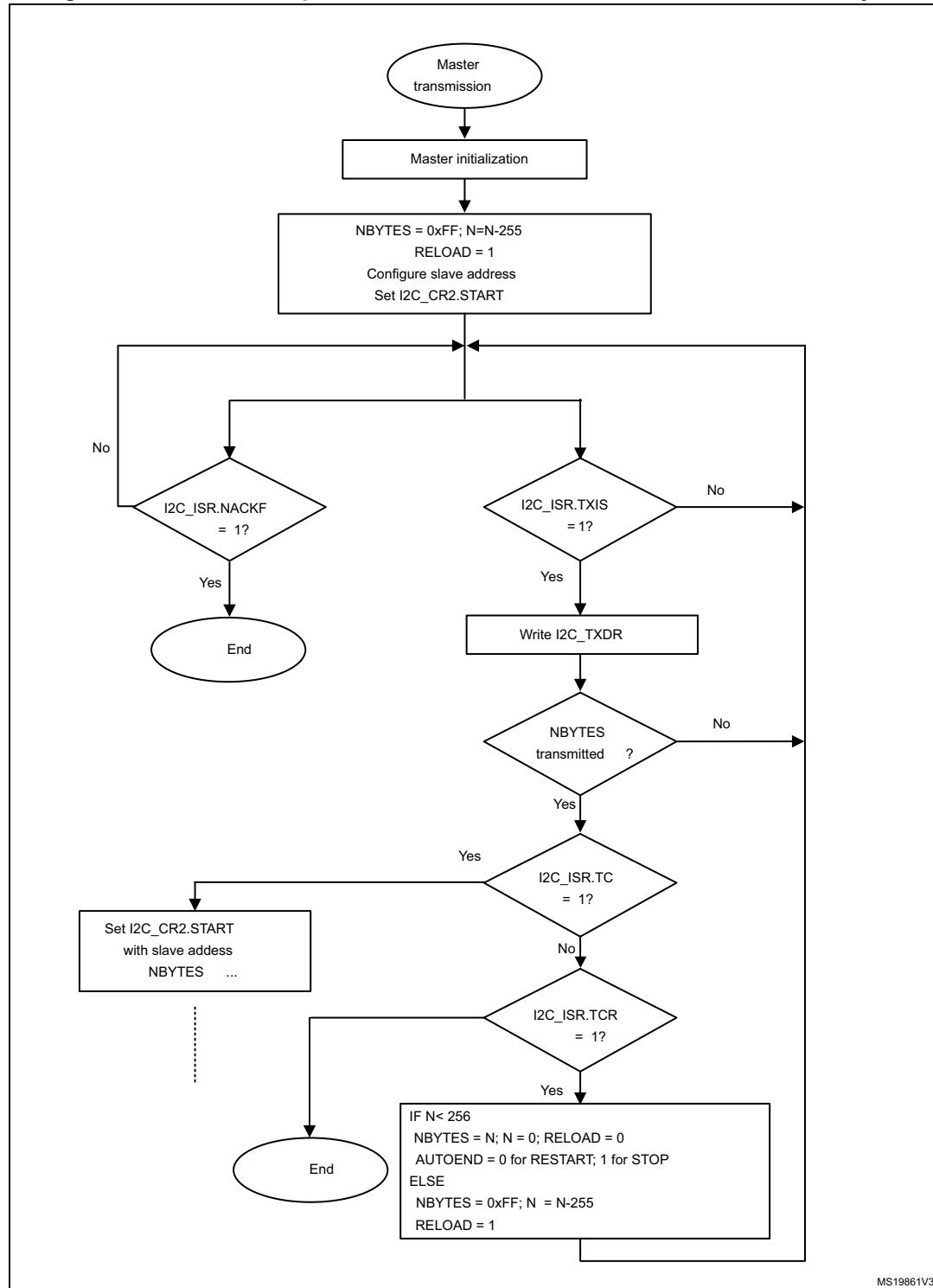
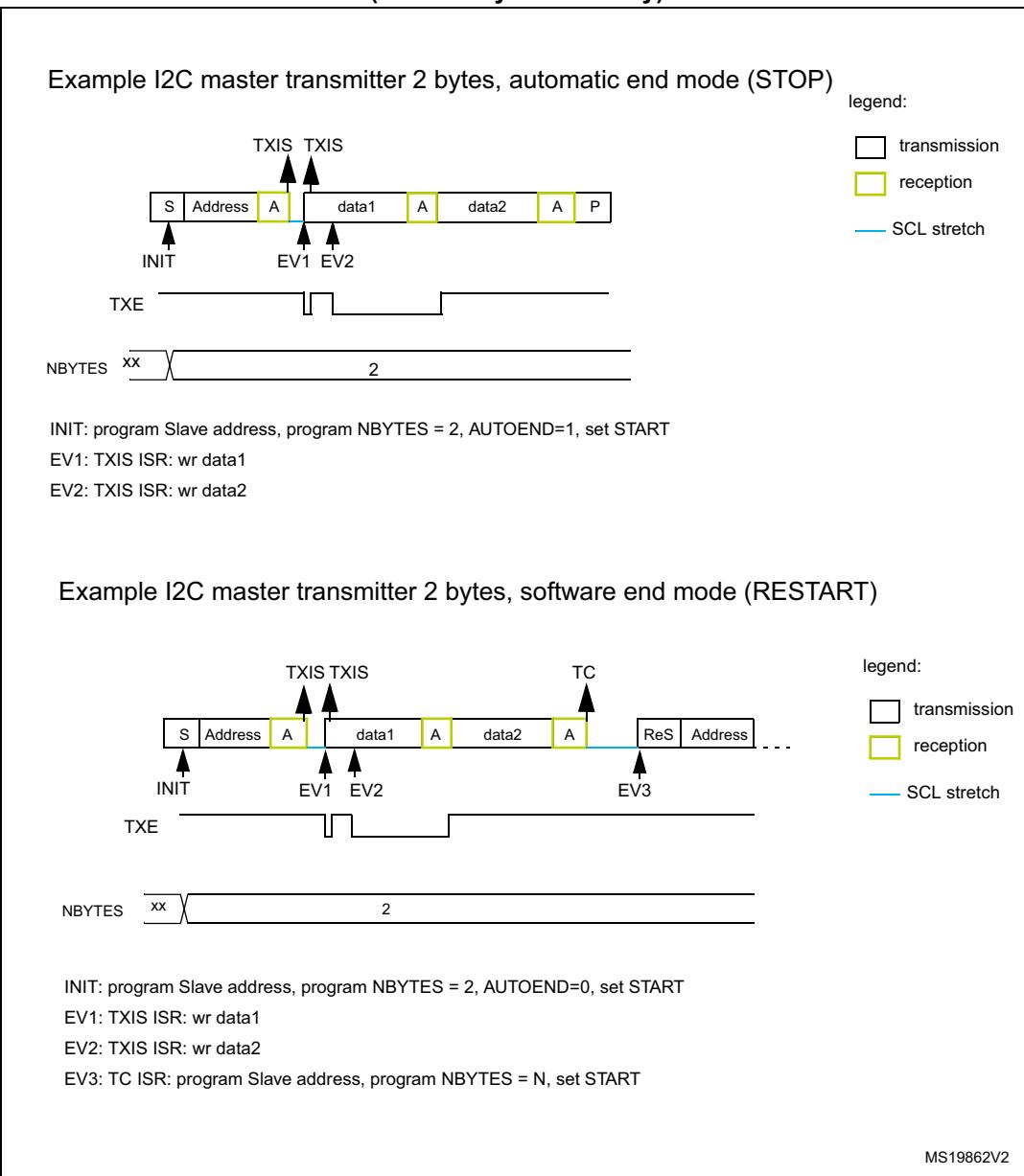
**Figure 235. Transfer sequence flow for I2C master transmitter for  $N \leq 255$  bytes**

Figure 236. Transfer sequence flow for I2C master transmitter for N &gt; 255 bytes



MS19861V3

**Figure 237. Transfer bus diagrams for I2C master transmitter  
(mandatory events only)**



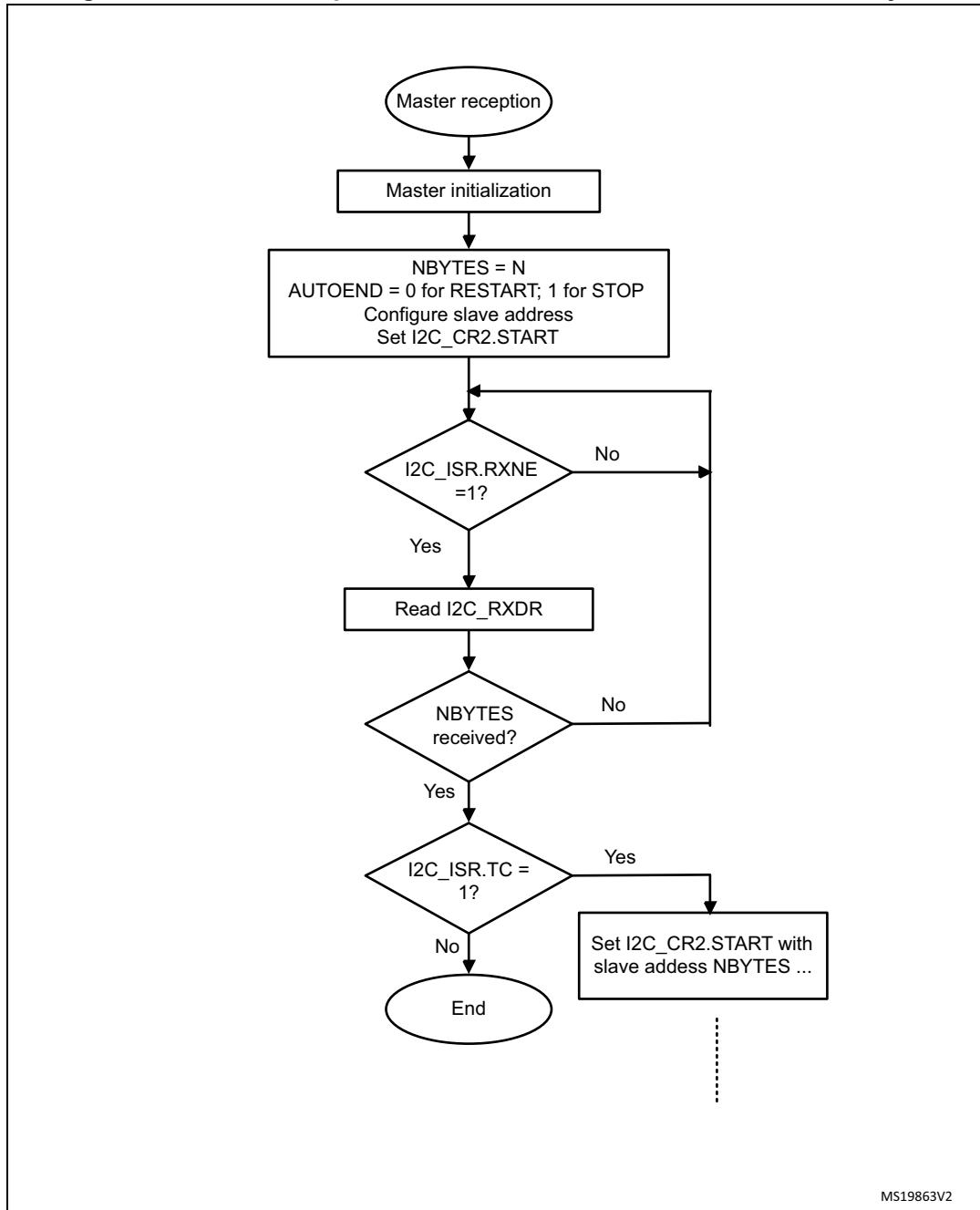
For a code example refer to [A.14.4: I2C master transmitter code example](#).

### Master receiver

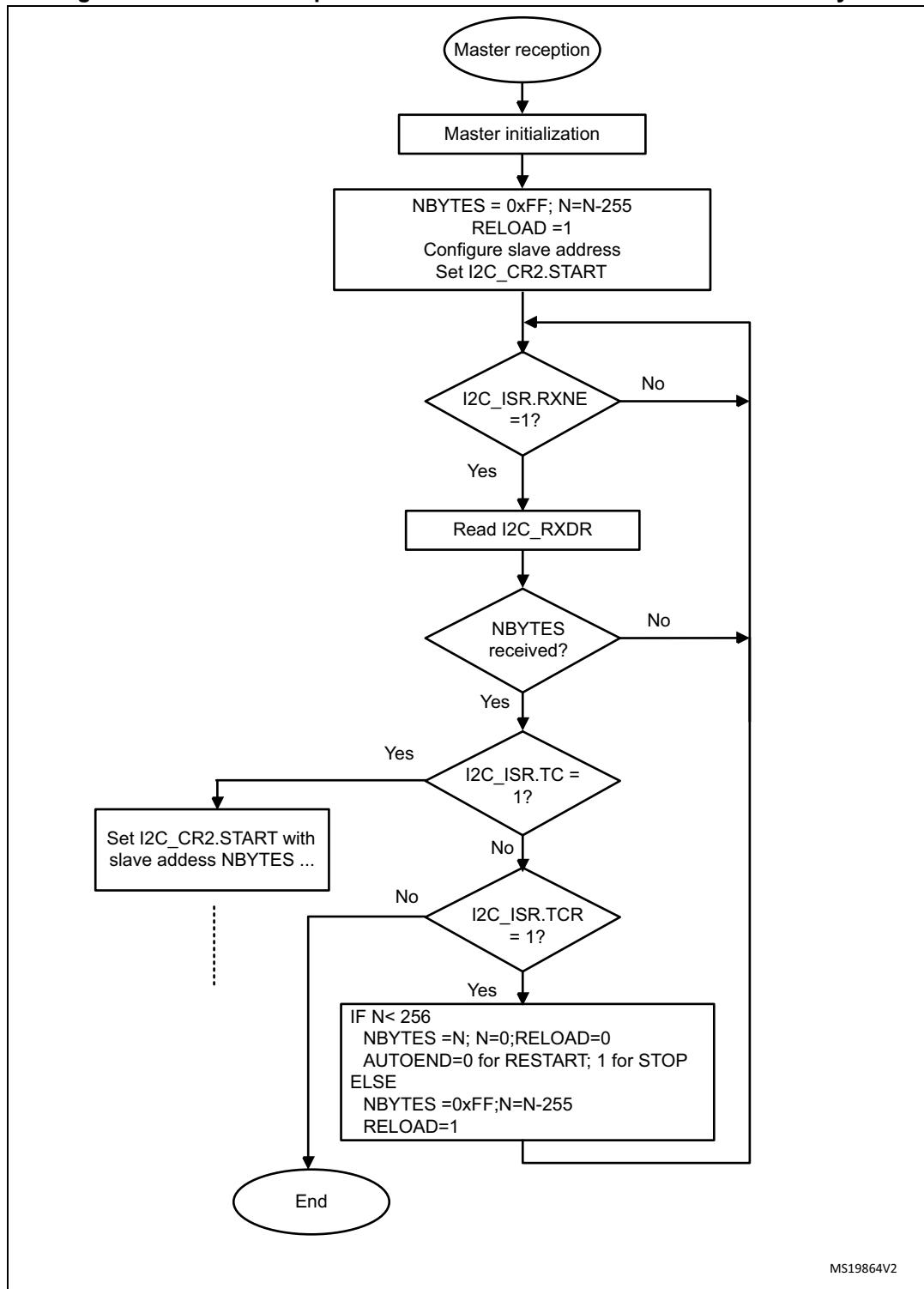
In the case of a read transfer, the RXNE flag is set after each byte reception, after the eighth SCL pulse. An RXNE event generates an interrupt if the RXIE bit is set in the I2C\_CR1 register. The flag is cleared when I2C\_RXDR is read.

If the total number of data bytes to be received is greater than 255, reload mode must be selected by setting the RELOAD bit in the I2C\_CR2 register. In this case, when NBYTES[7:0] data have been transferred, the TCR flag is set and the SCL line is stretched low until NBYTES[7:0] is written to a non-zero value.

- When RELOAD=0 and NBYTES[7:0] data have been transferred:
  - In automatic end mode (AUTOEND=1), a NACK and a STOP are automatically sent after the last received byte.
  - In software end mode (AUTOEND=0), a NACK is automatically sent after the last received byte, the TC flag is set and the SCL line is stretched low in order to allow software actions:  
A RESTART condition can be requested by setting the START bit in the I2C\_CR2 register with the proper slave address configuration, and number of bytes to be transferred. Setting the START bit clears the TC flag and the START condition, followed by slave address, are sent on the bus.  
A STOP condition can be requested by setting the STOP bit in the I2C\_CR2 register. Setting the STOP bit clears the TC flag and the STOP condition is sent on the bus.

**Figure 238. Transfer sequence flow for I2C master receiver for  $N \leq 255$  bytes**

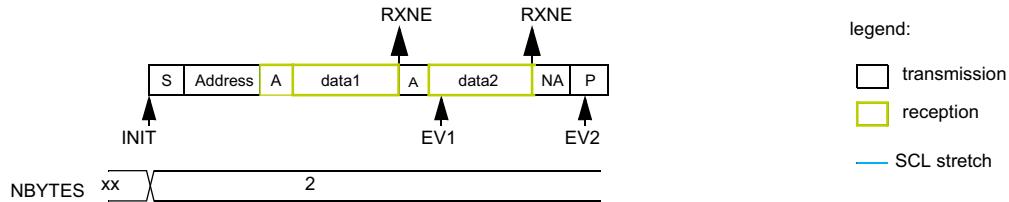
MS19863V2

**Figure 239. Transfer sequence flow for I2C master receiver for N > 255 bytes**

MS19864V2

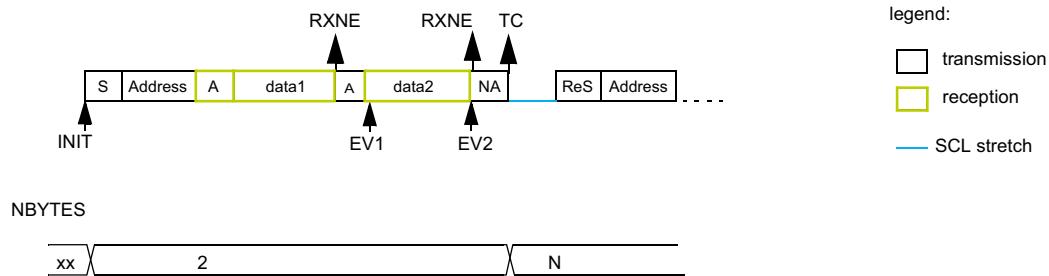
**Figure 240. Transfer bus diagrams for I2C master receiver  
(mandatory events only)**

Example I2C master receiver 2 bytes, automatic end mode (STOP)



INIT: program Slave address, program NBYTES = 2, AUTOEND=1, set START  
EV1: RXNE ISR: rd data1  
EV2: RXNE ISR: rd data2

Example I2C master receiver 2 bytes, software end mode (RESTART)



INIT: program Slave address, program NBYTES = 2, AUTOEND=0, set START  
EV1: RXNE ISR: rd data1  
EV2: RXNE ISR: read data2  
EV3: TC ISR: program Slave address, program NBYTES = N, set START

MS19865V1

For a code example refer to [A.14.5: I2C master receiver code example](#).

### 26.4.11 I2C\_TIMINGR register configuration examples

The tables below provide examples of how to program the I2C\_TIMINGR to obtain timings compliant with the I<sup>2</sup>C specification. In order to get more accurate configuration values, the STM32CubeMX tool (I2C Configuration window) must be used.

**Table 93. Examples of timing settings for  $f_{I2CCCLK} = 8 \text{ MHz}$**

Parameter	Standard-mode (Sm)		Fast-mode (Fm)	Fast-mode Plus (Fm+)
	10 kHz	100 kHz	400 kHz	500 kHz
PRESC	1	1	0	0
SCLL	0xC7	0x13	0x9	0x6
t <sub>SCLL</sub>	200 x 250 ns = 50 $\mu\text{s}$	20 x 250 ns = 5.0 $\mu\text{s}$	10 x 125 ns = 1250 ns	7 x 125 ns = 875 ns
SCLH	0xC3	0xF	0x3	0x3
t <sub>SCLH</sub>	196 x 250 ns = 49 $\mu\text{s}$	16 x 250 ns = 4.0 $\mu\text{s}$	4 x 125 ns = 500 ns	4 x 125 ns = 500 ns
t <sub>SCL</sub> <sup>(1)</sup>	~100 $\mu\text{s}$ <sup>(2)</sup>	~10 $\mu\text{s}$ <sup>(2)</sup>	~2500 ns <sup>(3)</sup>	~2000 ns <sup>(4)</sup>
SDADEL	0x2	0x2	0x1	0x0
t <sub>SDADEL</sub>	2 x 250 ns = 500 ns	2 x 250 ns = 500 ns	1 x 125 ns = 125 ns	0 ns
SCLDEL	0x4	0x4	0x3	0x1
t <sub>SCLDEL</sub>	5 x 250 ns = 1250 ns	5 x 250 ns = 1250 ns	4 x 125 ns = 500 ns	2 x 125 ns = 250 ns

1. SCL period t<sub>SCL</sub> is greater than t<sub>SCLL</sub> + t<sub>SCLH</sub> due to SCL internal detection delay. Values provided for t<sub>SCL</sub> are examples only.

2. t<sub>SYNC1</sub> + t<sub>SYNC2</sub> minimum value is 4 x t<sub>I2CCCLK</sub> = 500 ns. Example with t<sub>SYNC1</sub> + t<sub>SYNC2</sub> = 1000 ns.

3. t<sub>SYNC1</sub> + t<sub>SYNC2</sub> minimum value is 4 x t<sub>I2CCCLK</sub> = 500 ns. Example with t<sub>SYNC1</sub> + t<sub>SYNC2</sub> = 750 ns.

4. t<sub>SYNC1</sub> + t<sub>SYNC2</sub> minimum value is 4 x t<sub>I2CCCLK</sub> = 500 ns. Example with t<sub>SYNC1</sub> + t<sub>SYNC2</sub> = 655 ns.

**Table 94. Examples of timings settings for  $f_{I2CCCLK} = 16 \text{ MHz}$**

Parameter	Standard-mode (Sm)		Fast-mode (Fm)	Fast-mode Plus (Fm+)
	10 kHz	100 kHz	400 kHz	1000 kHz
PRESC	3	3	1	0
SCLL	0xC7	0x13	0x9	0x4
t <sub>SCLL</sub>	200 x 250 ns = 50 $\mu\text{s}$	20 x 250 ns = 5.0 $\mu\text{s}$	10 x 125 ns = 1250 ns	5 x 62.5 ns = 312.5 ns
SCLH	0xC3	0xF	0x3	0x2
t <sub>SCLH</sub>	196 x 250 ns = 49 $\mu\text{s}$	16 x 250 ns = 4.0 $\mu\text{s}$	4 x 125 ns = 500 ns	3 x 62.5 ns = 187.5 ns
t <sub>SCL</sub> <sup>(1)</sup>	~100 $\mu\text{s}$ <sup>(2)</sup>	~10 $\mu\text{s}$ <sup>(2)</sup>	~2500 ns <sup>(3)</sup>	~1000 ns <sup>(4)</sup>
SDADEL	0x2	0x2	0x2	0x0
t <sub>SDADEL</sub>	2 x 250 ns = 500 ns	2 x 250 ns = 500 ns	2 x 125 ns = 250 ns	0 ns
SCLDEL	0x4	0x4	0x3	0x2
t <sub>SCLDEL</sub>	5 x 250 ns = 1250 ns	5 x 250 ns = 1250 ns	4 x 125 ns = 500 ns	3 x 62.5 ns = 187.5 ns

1. SCL period t<sub>SCL</sub> is greater than t<sub>SCLL</sub> + t<sub>SCLH</sub> due to SCL internal detection delay. Values provided for t<sub>SCL</sub> are examples only.

2.  $t_{SYNC1} + t_{SYNC2}$  minimum value is  $4 \times t_{I2CCLK} = 250$  ns. Example with  $t_{SYNC1} + t_{SYNC2} = 1000$  ns.
3.  $t_{SYNC1} + t_{SYNC2}$  minimum value is  $4 \times t_{I2CCLK} = 250$  ns. Example with  $t_{SYNC1} + t_{SYNC2} = 750$  ns.
4.  $t_{SYNC1} + t_{SYNC2}$  minimum value is  $4 \times t_{I2CCLK} = 250$  ns. Example with  $t_{SYNC1} + t_{SYNC2} = 500$  ns.

**Table 95. Examples of timings settings for  $f_{I2CCLK} = 48$  MHz**

Parameter	Standard-mode (Sm)		Fast-mode (Fm)	Fast-mode Plus (Fm+)
	10 kHz	100 kHz	400 kHz	1000 kHz
PRESC	0xB	0xB	5	5
SCLL	0xC7	0x13	0x9	0x3
$t_{SCLL}$	$200 \times 250$ ns = 50 $\mu$ s	$20 \times 250$ ns = 5.0 $\mu$ s	$10 \times 125$ ns = 1250 ns	$4 \times 125$ ns = 500 ns
SCLH	0xC3	0xF	0x3	0x1
$t_{SCLH}$	$196 \times 250$ ns = 49 $\mu$ s	$16 \times 250$ ns = 4.0 $\mu$ s	$4 \times 125$ ns = 500 ns	$2 \times 125$ ns = 250 ns
$t_{SCL}^{(1)}$	$\sim 100$ $\mu$ s <sup>(2)</sup>	$\sim 10$ $\mu$ s <sup>(2)</sup>	$\sim 2500$ ns <sup>(3)</sup>	$\sim 875$ ns <sup>(4)</sup>
SDADEL	0x2	0x2	0x3	0x0
$t_{SDADEL}$	$2 \times 250$ ns = 500 ns	$2 \times 250$ ns = 500 ns	$3 \times 125$ ns = 375 ns	0 ns
SCLDEL	0x4	0x4	0x3	0x1
$t_{SCLDEL}$	$5 \times 250$ ns = 1250 ns	$5 \times 250$ ns = 1250 ns	$4 \times 125$ ns = 500 ns	$2 \times 125$ ns = 250 ns

1. The SCL period  $t_{SCL}$  is greater than  $t_{SCLL} + t_{SCLH}$  due to the SCL internal detection delay. Values provided for  $t_{SCL}$  are only examples.
2.  $t_{SYNC1} + t_{SYNC2}$  minimum value is  $4 \times t_{I2CCLK} = 83.3$  ns. Example with  $t_{SYNC1} + t_{SYNC2} = 1000$  ns
3.  $t_{SYNC1} + t_{SYNC2}$  minimum value is  $4 \times t_{I2CCLK} = 83.3$  ns. Example with  $t_{SYNC1} + t_{SYNC2} = 750$  ns
4.  $t_{SYNC1} + t_{SYNC2}$  minimum value is  $4 \times t_{I2CCLK} = 83.3$  ns. Example with  $t_{SYNC1} + t_{SYNC2} = 250$  ns

### 26.4.12 SMBus specific features

This section is relevant only when the SMBus feature is supported. Refer to [Section 26.3: I2C implementation](#).

#### Introduction

The system management bus (SMBus) is a two-wire interface through which various devices can communicate with each other and with the rest of the system. It is based on I<sup>2</sup>C principles of operation. The SMBus provides a control bus for system and power management related tasks.

This peripheral is compatible with the SMBus specification (<http://smbus.org>).

The system management bus specification refers to three types of devices

- A slave is a device that receives or responds to a command.
- A master is a device that issues commands, generates the clocks and terminates the transfer.
- A host is a specialized master that provides the main interface to the system's CPU. A host must be a master-slave and must support the SMBus host notify protocol. Only one host is allowed in a system.

This peripheral can be configured as master or slave device, and also as a host.

## Bus protocols

There are eleven possible command protocols for any given device. A device may use any or all of the eleven protocols to communicate. The protocols are Quick Command, Send Byte, Receive Byte, Write Byte, Write Word, Read Byte, Read Word, Process Call, Block Read, Block Write and Block Write-Block Read Process Call. These protocols must be implemented by the user software.

For more details on these protocols, refer to SMBus specification (<http://smbus.org>).

### Address resolution protocol (ARP)

SMBus slave address conflicts can be resolved by dynamically assigning a new unique address to each slave device. In order to provide a mechanism to isolate each device for the purpose of address assignment each device must implement a unique device identifier (UDID). This 128-bit number is implemented by software.

This peripheral supports the Address Resolution Protocol (ARP). The SMBus Device Default Address (0b1100 001) is enabled by setting SMBDEN bit in I2C\_CR1 register. The ARP commands must be implemented by the user software.

Arbitration is also performed in slave mode for ARP support.

For more details of the SMBus address resolution protocol, refer to SMBus specification (<http://smbus.org>).

### Received command and data acknowledge control

A SMBus receiver must be able to NACK each received command or data. In order to allow the ACK control in slave mode, the Slave Byte Control mode must be enabled by setting SBC bit in I2C\_CR1 register. Refer to *Slave byte control mode* for more details.

### Host notify protocol

This peripheral supports the host notify protocol by setting the SMBHEN bit in the I2C\_CR1 register. In this case the host acknowledges the SMBus host address (0b0001 000).

When this protocol is used, the device acts as a master and the host as a slave.

### SMBus alert

The SMBus ALERT optional signal is supported. A slave-only device can signal the host through the SMBALERT# pin that it wants to talk. The host processes the interrupt and simultaneously accesses all SMBALERT# devices through the alert response address (0b0001 100). Only the device(s) which pulled SMBALERT# low acknowledges the alert response address.

When configured as a slave device(SMBHEN=0), the SMBA pin is pulled low by setting the ALERTEN bit in the I2C\_CR1 register. The Alert Response Address is enabled at the same time.

When configured as a host (SMBHEN=1), the ALERT flag is set in the I2C\_ISR register when a falling edge is detected on the SMBA pin and ALERTEN=1. An interrupt is generated if the ERRIE bit is set in the I2C\_CR1 register. When ALERTEN=0, the ALERT line is considered high even if the external SMBA pin is low.

*If the SMBus ALERT pin is not needed, the SMBA pin can be used as a standard GPIO if ALERTEN=0.*

## Packet error checking

A packet error checking mechanism has been introduced in the SMBus specification to improve reliability and communication robustness. The packet error checking is implemented by appending a packet error code (PEC) at the end of each message transfer. The PEC is calculated by using the  $C(x) = x^8 + x^2 + x + 1$  CRC-8 polynomial on all the message bytes (including addresses and read/write bits).

The peripheral embeds a hardware PEC calculator and allows a not acknowledge to be sent automatically when the received byte does not match with the hardware calculated PEC.

## Timeouts

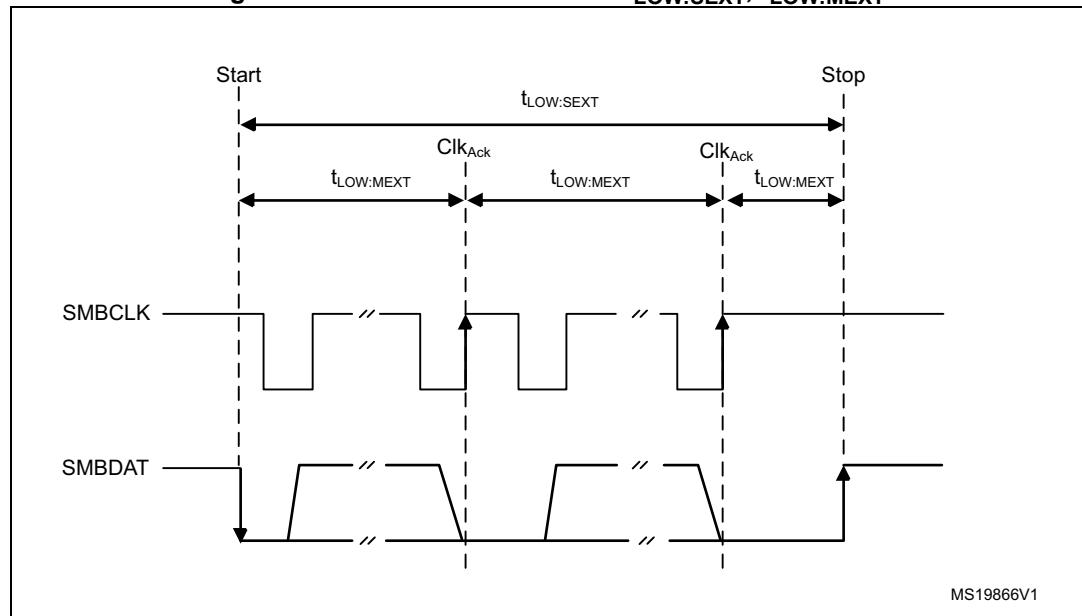
This peripheral embeds hardware timers in order to be compliant with the three timeouts defined in SMBus specification.

**Table 96. SMBus timeout specifications**

Symbol	Parameter	Limits		Unit
		Min	Max	
$t_{TIMEOUT}$	Detect clock low timeout	25	35	ms
$t_{LOW:SEXT}^{(1)}$	Cumulative clock low extend time (slave device)	-	25	
$t_{LOW:MEXT}^{(2)}$	Cumulative clock low extend time (master device)	-	10	

1.  $t_{LOW:SEXT}$  is the cumulative time a given slave device is allowed to extend the clock cycles in one message from the initial START to the STOP. It is possible that, another slave device or the master also extends the clock causing the combined clock low extend time to be greater than  $t_{LOW:SEXT}$ . Therefore, this parameter is measured with the slave device as the sole target of a full-speed master.
2.  $t_{LOW:MEXT}$  is the cumulative time a master device is allowed to extend its clock cycles within each byte of a message as defined from START-to-ACK, ACK-to-ACK, or ACK-to-STOP. It is possible that a slave device or another master also extends the clock causing the combined clock low time to be greater than  $t_{LOW:MEXT}$  on a given byte. Therefore, this parameter is measured with a full speed slave device as the sole target of the master.

**Figure 241. Timeout intervals for  $t_{LOW:SEXT}$ ,  $t_{LOW:MEXT}$**



### Bus idle detection

A master can assume that the bus is free if it detects that the clock and data signals have been high for  $t_{IDLE}$  greater than  $t_{HIGH,MAX}$ . (refer to [Table 90](#))

This timing parameter covers the condition where a master has been dynamically added to the bus and may not have detected a state transition on the SMBCLK or SMBDAT lines. In this case, the master must wait long enough to ensure that a transfer is not currently in progress. The peripheral supports a hardware bus idle detection.

### 26.4.13 SMBus initialization

This section is relevant only when SMBus feature is supported. Refer to [Section 26.3: I2C implementation](#).

In addition to I2C initialization, some other specific initialization must be done in order to perform SMBus communication:

#### Received command and data acknowledge control (Slave mode)

A SMBus receiver must be able to NACK each received command or data. In order to allow ACK control in slave mode, the Slave byte control mode must be enabled by setting the SBC bit in the I2C\_CR1 register. Refer to [Slave byte control mode](#) for more details.

#### Specific address (Slave mode)

The specific SMBus addresses must be enabled if needed. Refer to [Bus idle detection](#) for more details.

- The SMBus device default address (0b1100 001) is enabled by setting the SMBDEN bit in the I2C\_CR1 register.
- The SMBus host address (0b0001 000) is enabled by setting the SMBHEN bit in the I2C\_CR1 register.
- The alert response address (0b0001100) is enabled by setting the ALERTEN bit in the I2C\_CR1 register.

#### Packet error checking

PEC calculation is enabled by setting the PECEN bit in the I2C\_CR1 register. Then the PEC transfer is managed with the help of a hardware byte counter: NBYTES[7:0] in the I2C\_CR2 register. The PECEN bit must be configured before enabling the I2C.

The PEC transfer is managed with the hardware byte counter, so the SBC bit must be set when interfacing the SMBus in slave mode. The PEC is transferred after NBYTES - 1 data have been transferred when the PECPBYTE bit is set and the RELOAD bit is cleared. If RELOAD is set, PECPBYTE has no effect.

**Caution:** Changing the PECEN configuration is not allowed when the I2C is enabled.

**Table 97. SMBus with PEC configuration**

Mode	SBC bit	RELOAD bit	AUTOEND bit	PECPBYTE bit
Master Tx/Rx NBYTES + PEC+ STOP	x	0	1	1
Master Tx/Rx NBYTES + PEC + ReSTART	x	0	0	1
Slave Tx/Rx with PEC	1	0	x	1

### Timeout detection

The timeout detection is enabled by setting the TIMOUTEN and TEXTEN bits in the I2C\_TIMEOUTTR register. The timers must be programmed in such a way that they detect a timeout before the maximum time given in the SMBus specification.

- $t_{TIMEOUT}$  check

In order to enable the  $t_{TIMEOUT}$  check, the 12-bit TIMEOUTA[11:0] bits must be programmed with the timer reload value in order to check the  $t_{TIMEOUT}$  parameter. The TIDLE bit must be configured to '0' in order to detect the SCL low level timeout.

Then the timer is enabled by setting the TIMOUTEN in the I2C\_TIMEOUTTR register.

If SCL is tied low for a time greater than  $(TIMEOUTA+1) \times 2048 \times t_{I2CCLK}$ , the TIMEOUT flag is set in the I2C\_ISR register.

Refer to [Table 98](#).

**Caution:** Changing the TIMEOUTA[11:0] bits and TIDLE bit configuration is not allowed when the TIMOUTEN bit is set.

- $t_{LOW:SEXT}$  and  $t_{LOW:MEXT}$  check

Depending on if the peripheral is configured as a master or as a slave, The 12-bit TIMEOUTB timer must be configured in order to check  $t_{LOW:SEXT}$  for a slave and  $t_{LOW:MEXT}$  for a master. As the standard specifies only a maximum, the user can choose the same value for the both.

Then the timer is enabled by setting the TEXTEN bit in the I2C\_TIMEOUTTR register.

If the SMBus peripheral performs a cumulative SCL stretch for a time greater than  $(TIMEOUTB+1) \times 2048 \times t_{I2CCLK}$ , and in the timeout interval described in [Bus idle detection](#) section, the TIMEOUT flag is set in the I2C\_ISR register.

Refer to [Table 99](#)

**Caution:** Changing the TIMEOUTB configuration is not allowed when the TEXTEN bit is set.

### Bus idle detection

In order to enable the  $t_{IDLE}$  check, the 12-bit TIMEOUTA[11:0] field must be programmed with the timer reload value in order to obtain the  $t_{IDLE}$  parameter. The TIDLE bit must be configured to '1' in order to detect both SCL and SDA high level timeout.

Then the timer is enabled by setting the TIMOUTEN bit in the I2C\_TIMEOUTTR register.

If both the SCL and SDA lines remain high for a time greater than  $(TIMEOUTA+1) \times 4 \times t_{I2CCLK}$ , the TIMEOUT flag is set in the I2C\_ISR register.

Refer to [Table 100](#).

**Caution:** Changing the TIMEOUTA and TIDLE configuration is not allowed when the TIMOUTEN is set.

### 26.4.14 SMBus: I2C\_TIMEOUTR register configuration examples

This section is relevant only when SMBus feature is supported. Refer to [Section 26.3: I2C implementation](#).

- Configuring the maximum duration of  $t_{TIMEOUT}$  to 25 ms:

**Table 98. Examples of TIMEOUTA settings for various I2CCLK frequencies  
(max  $t_{TIMEOUT} = 25$  ms)**

$f_{I2CCLK}$	TIMEOUTA[11:0] bits	TIDLE bit	TIMEOUTEN bit	$t_{TIMEOUT}$
8 MHz	0x61	0	1	$98 \times 2048 \times 125$ ns = 25 ms
16 MHz	0xC3	0	1	$196 \times 2048 \times 62.5$ ns = 25 ms
48 MHz	0x249	0	1	$586 \times 2048 \times 20.08$ ns = 25 ms

- Configuring the maximum duration of  $t_{LOW:SEXT}$  and  $t_{LOW:MEXT}$  to 8 ms:

**Table 99. Examples of TIMEOUTB settings for various I2CCLK frequencies**

$f_{I2CCLK}$	TIMEOUTB[11:0] bits	TEXTEN bit	$t_{LOW:EXT}$
8 MHz	0x1F	1	$32 \times 2048 \times 125$ ns = 8 ms
16 MHz	0x3F	1	$64 \times 2048 \times 62.5$ ns = 8 ms
48 MHz	0xBB	1	$188 \times 2048 \times 20.08$ ns = 8 ms

- Configuring the maximum duration of  $t_{IDLE}$  to 50  $\mu$ s

**Table 100. Examples of TIMEOUTA settings for various I2CCLK frequencies  
(max  $t_{IDLE} = 50$   $\mu$ s)**

$f_{I2CCLK}$	TIMEOUTA[11:0] bits	TIDLE bit	TIMEOUTEN bit	$t_{TIDLE}$
8 MHz	0x63	1	1	$100 \times 4 \times 125$ ns = 50 $\mu$ s
16 MHz	0xC7	1	1	$200 \times 4 \times 62.5$ ns = 50 $\mu$ s
48 MHz	0x257	1	1	$600 \times 4 \times 20.08$ ns = 50 $\mu$ s

### 26.4.15 SMBus slave mode

This section is relevant only when the SMBus feature is supported. Refer to [Section 26.3: I2C implementation](#).

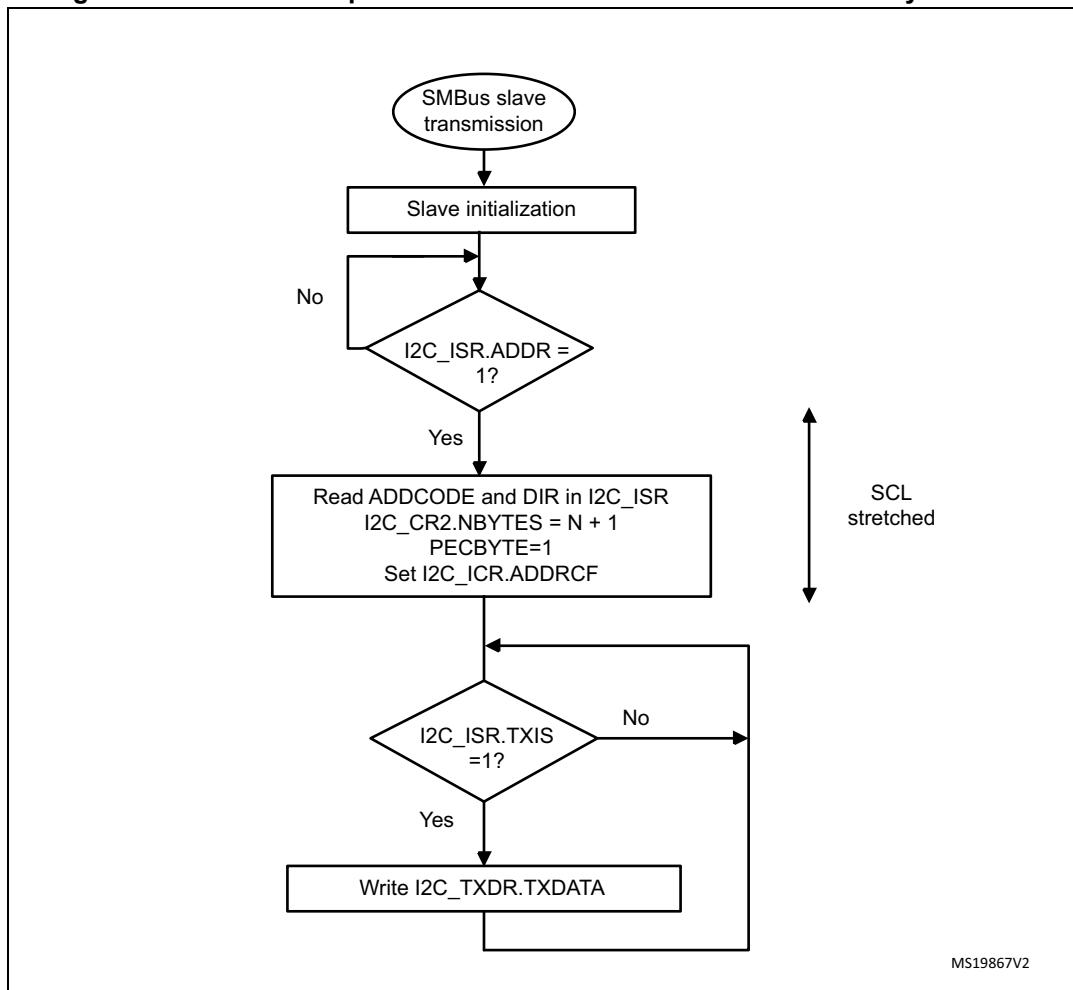
In addition to I2C slave transfer management (refer to [Section 26.4.9: I2C slave mode](#)) some additional software flows are provided to support the SMBus.

#### SMBus slave transmitter

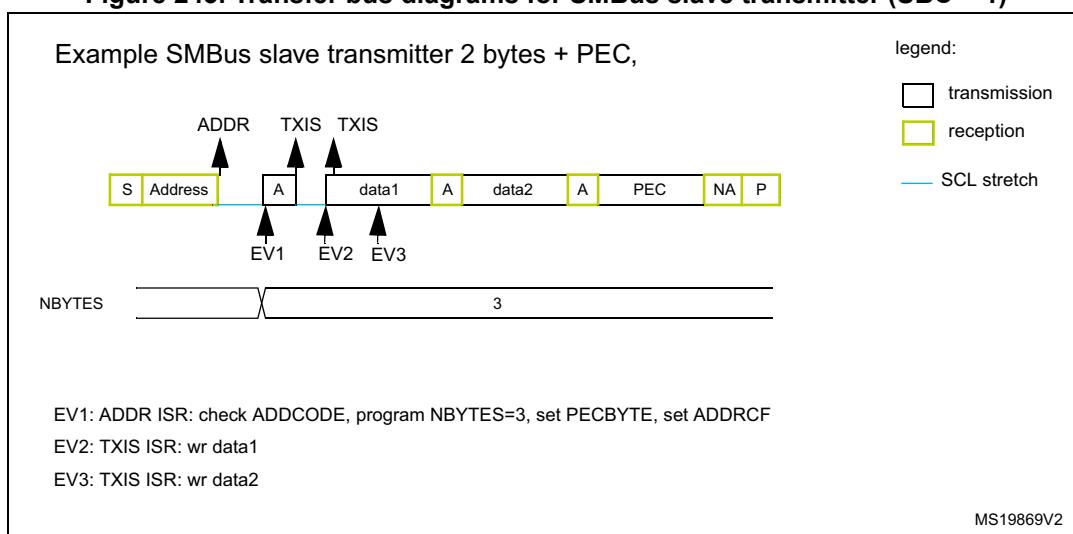
When the IP is used in SMBus, SBC must be programmed to '1' in order to allow the PEC transmission at the end of the programmed number of data bytes. When the PECPBYTE bit is set, the number of bytes programmed in NBYTES[7:0] includes the PEC transmission. In that case the total number of TXIS interrupts is NBYTES - 1 and the content of the I2C\_PECR register is automatically transmitted if the master requests an extra byte after the NBYTES - 1 data transfer.

**Caution:** The PECBYTE bit has no effect when the RELOAD bit is set.

**Figure 242. Transfer sequence flow for SMBus slave transmitter N bytes + PEC**



**Figure 243. Transfer bus diagrams for SMBus slave transmitter (SBC = 1)**



### SMBus Slave receiver

When the I2C is used in SMBus mode, SBC must be programmed to '1' in order to allow the PEC checking at the end of the programmed number of data bytes. In order to allow the ACK control of each byte, the reload mode must be selected (RELOAD=1). Refer to [Slave byte control mode](#) for more details.

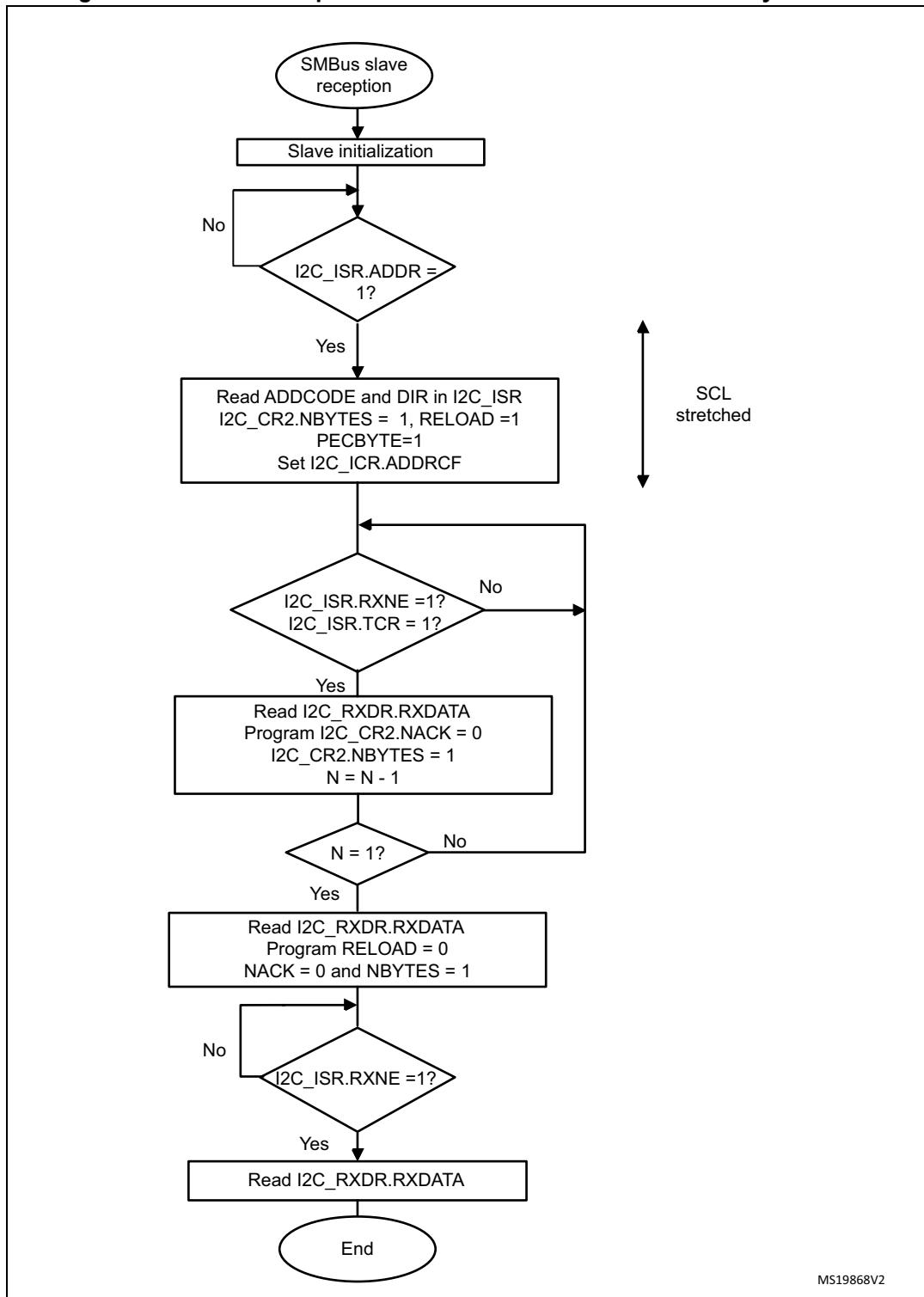
In order to check the PEC byte, the RELOAD bit must be cleared and the PECPBYTE bit must be set. In this case, after NBYTES - 1 data have been received, the next received byte is compared with the internal I2C\_PECR register content. A NACK is automatically generated if the comparison does not match, and an ACK is automatically generated if the comparison matches, whatever the ACK bit value. Once the PEC byte is received, it is copied into the I2C\_RXDR register like any other data, and the RXNE flag is set.

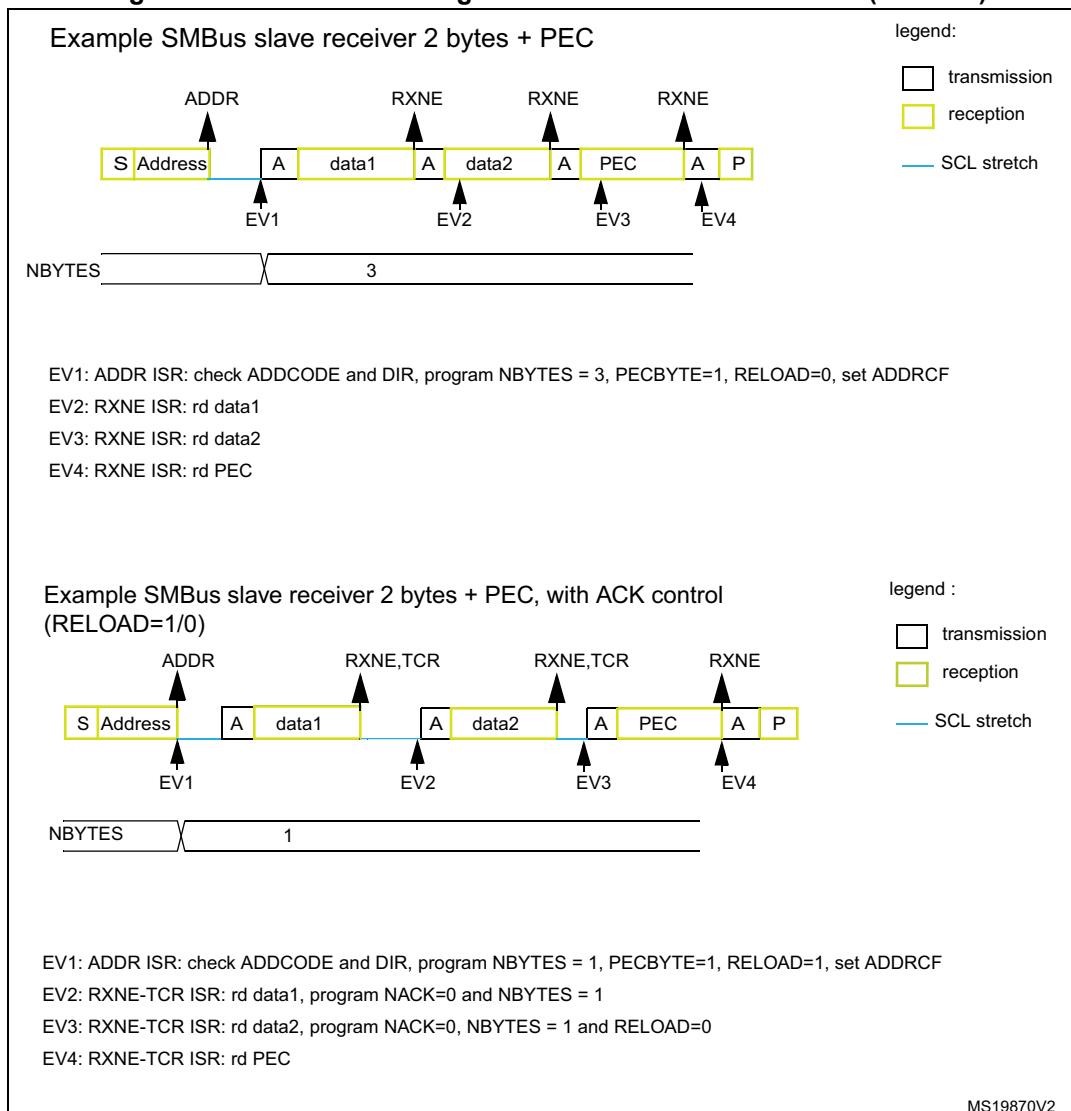
In the case of a PEC mismatch, the PECPERR flag is set and an interrupt is generated if the ERRIE bit is set in the I2C\_CR1 register.

If no ACK software control is needed, the user can program PECPBYTE=1 and, in the same write operation, program NBYTES with the number of bytes to be received in a continuous flow. After NBYTES - 1 are received, the next received byte is checked as being the PEC.

**Caution:** The PECPBYTE bit has no effect when the RELOAD bit is set.

Figure 244. Transfer sequence flow for SMBus slave receiver N bytes + PEC



**Figure 245. Bus transfer diagrams for SMBus slave receiver (SBC = 1)**

MS19870V2

This section is relevant only when the SMBus feature is supported. Refer to [Section 26.3: I2C implementation](#).

In addition to I2C master transfer management (refer to [Section 26.4.10: I2C master mode](#)), some additional software flows are provided to support the SMBus.

### SMBus master transmitter

When the SMBus master wants to transmit the PEC, the PECBYTE bit must be set and the number of bytes must be programmed in the NBYTES[7:0] field, before setting the START bit. In this case the total number of TXIS interrupts is NBYTES - 1. So if the PECBYTE bit is set when NBYTES = 0x1, the content of the I2C\_PECR register is automatically transmitted.

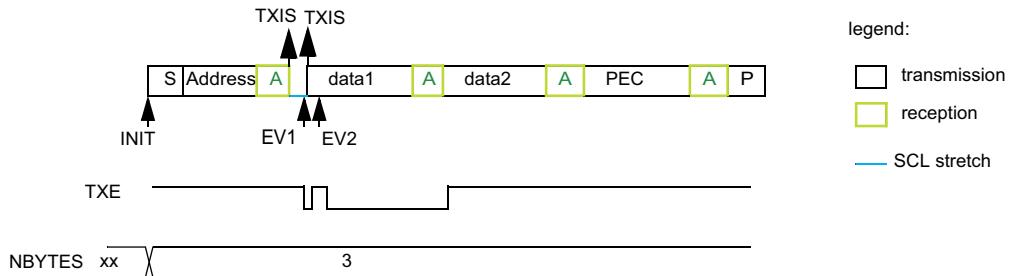
If the SMBus master wants to send a STOP condition after the PEC, automatic end mode must be selected (AUTOEND = 1). In this case, the STOP condition automatically follows the PEC transmission.

When the SMBus master wants to send a RESTART condition after the PEC, software mode must be selected (AUTOEND=0). In this case, once NBYTES - 1 have been transmitted, the I2C\_PECR register content is transmitted and the TC flag is set after the PEC transmission, stretching the SCL line low. The RESTART condition must be programmed in the TC interrupt subroutine.

**Caution:** The PECBYTE bit has no effect when the RELOAD bit is set.

**Figure 246. Bus transfer diagrams for SMBus master transmitter**

Example SMBus master transmitter 2 bytes + PEC, automatic end mode (STOP)

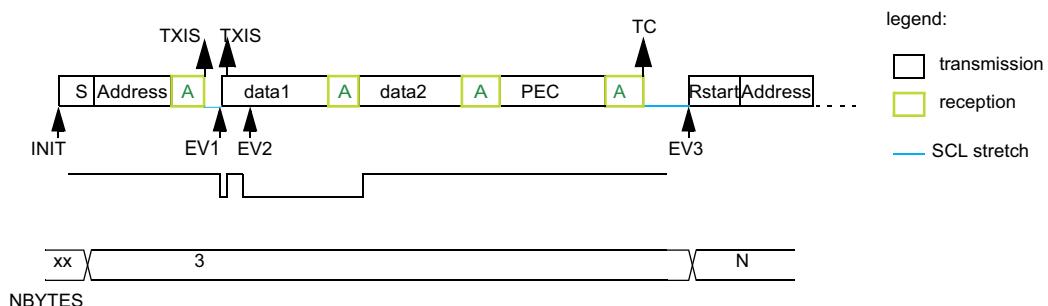


INIT: program Slave address, program NBYTES = 3, AUTOEND=1, set PECBYTE, set START

EV1: TXIS ISR: wr data1

EV2: TXIS ISR: wr data2

Example SMBus master transmitter 2 bytes + PEC, software end mode (RESTART)



INIT: program Slave address, program NBYTES = 3, AUTOEND=0, set PECBYTE, set START

EV1: TXIS ISR: wr data1

EV2: TXIS ISR: wr data2

EV3: TC ISR: program Slave address, program NBYTES = N, set START

MS19871V2

### SMBus master receiver

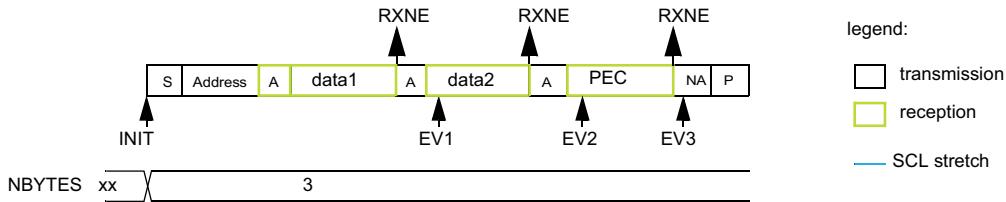
When the SMBus master wants to receive the PEC followed by a STOP at the end of the transfer, automatic end mode can be selected (AUTOEND = 1). The PECBYTE bit must be set and the slave address must be programmed, before setting the START bit. In this case, after NBYTES - 1 data have been received, the next received byte is automatically checked versus the I2C\_PECR register content. A NACK response is given to the PEC byte, followed by a STOP condition.

When the SMBus master receiver wants to receive the PEC byte followed by a RESTART condition at the end of the transfer, software mode must be selected (AUTOEND=0). The PECBYTE bit must be set and the slave address must be programmed, before setting the START bit. In this case, after NBYTES - 1 data have been received, the next received byte is automatically checked versus the I2C\_PECR register content. The TC flag is set after the PEC byte reception, stretching the SCL line low. The RESTART condition can be programmed in the TC interrupt subroutine.

**Caution:** The PECBYTE bit has no effect when the RELOAD bit is set.

**Figure 247. Bus transfer diagrams for SMBus master receiver**

Example SMBus master receiver 2 bytes + PEC, automatic end mode (STOP)



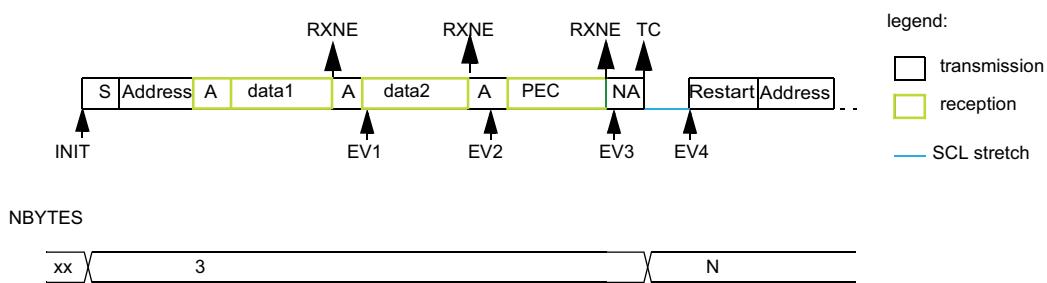
INIT: program Slave address, program NBYTES = 3, AUTOEND=1, set PECPBYTE, set START

EV1: RXNE ISR: rd data1

EV2: RXNE ISR: rd data2

EV3: RXNE ISR: rd PEC

Example SMBus master receiver 2 bytes + PEC, software end mode (RESTART)



INIT: program Slave address, program NBYTES = 3, AUTOEND=0, set PECPBYTE, set START

EV1: RXNE ISR: rd data1

EV2: RXNE ISR: rd data2

EV3: RXNE ISR: read PEC

EV4: TC ISR: program Slave address, program NBYTES = N, set START

MS19872V2

## 26.4.16 Wake-up from Stop mode on address match

This section is relevant only when wake-up from Stop mode feature is supported. Refer to [Section 26.3: I2C implementation](#).

The I2C is able to wake-up the MCU from Stop mode (APB clock is off), when it is addressed. All addressing modes are supported.

Wake-up from Stop mode is enabled by setting the WUPEN bit in the I2C\_CR1 register. The HSI oscillator must be selected as the clock source for I2CCLK in order to allow wake-up from Stop mode.

During Stop mode, the HSI is switched off. When a START is detected, the I2C interface switches the HSI on, and stretches SCL low until HSI is woken up.

HSI is then used for the address reception.

In case of an address match, the I2C stretches SCL low during MCU wake-up time. The stretch is released when ADDR flag is cleared by software, and the transfer goes on normally.

If the address does not match, the HSI is switched off again and the MCU is not woken up.

**Note:** *If the I2C clock is the system clock, or if WUPEN = 0, the HSI is not switched on after a START is received.*

*Only an ADDR interrupt can wake-up the MCU. Therefore do not enter Stop mode when the I2C is performing a transfer as a master, or as an addressed slave after the ADDR flag is set. This can be managed by clearing SLEEPDEEP bit in the ADDR interrupt routine and setting it again only after the STOPF flag is set.*

**Caution:** The digital filter is not compatible with the wake-up from Stop mode feature. If the DNF bit is not equal to 0, setting the WUPEN bit has no effect.

**Caution:** This feature is available only when the I2C clock source is the HSI oscillator.

**Caution:** Clock stretching must be enabled (NOSTRETCH = 0) to ensure proper operation of the wake-up from Stop mode feature.

**Caution:** If wake-up from Stop mode is disabled (WUPEN = 0), the I2C peripheral must be disabled before entering Stop mode (PE = 0).

#### 26.4.17 Error conditions

The following errors are the conditions which may cause communication to fail.

##### Bus error (BERR)

A bus error is detected when a START or a STOP condition is detected and is not located after a multiple of nine SCL clock pulses. A START or a STOP condition is detected when a SDA edge occurs while SCL is high.

The bus error flag is set only if the I2C is involved in the transfer as master or addressed slave (i.e not during the address phase in slave mode).

In case of a misplaced START or RESTART detection in slave mode, the I2C enters address recognition state like for a correct START condition.

When a bus error is detected, the BERR flag is set in the I2C\_ISR register, and an interrupt is generated if the ERRIE bit is set in the I2C\_CR1 register.

##### Arbitration lost (ARLO)

An arbitration loss is detected when a high level is sent on the SDA line, but a low level is sampled on the SCL rising edge.

- In master mode, arbitration loss is detected during the address phase, data phase and data acknowledge phase. In this case, the SDA and SCL lines are released, the START control bit is cleared by hardware and the master switches automatically to slave mode.
- In slave mode, arbitration loss is detected during data phase and data acknowledge phase. In this case, the transfer is stopped, and the SCL and SDA lines are released.

When an arbitration loss is detected, the ARLO flag is set in the I2C\_ISR register, and an interrupt is generated if the ERRIE bit is set in the I2C\_CR1 register.

### Overrun/underrun error (OVR)

An overrun or underrun error is detected in slave mode when NOSTRETCH = 1 and:

- In reception when a new byte is received and the RXDR register has not been read yet.  
The new received byte is lost, and a NACK is automatically sent as a response to the new byte.
- In transmission:
  - When STOPF=1 and the first data byte must be sent. The content of the I2C\_TXDR register is sent if TXE = 0, 0xFF if not.
  - When a new byte must be sent and the I2C\_TXDR register has not been written yet, 0xFF is sent.

When an overrun or underrun error is detected, the OVR flag is set in the I2C\_ISR register, and an interrupt is generated if the ERRIE bit is set in the I2C\_CR1 register.

### Packet error checking error (PECERR)

This section is relevant only when the SMBus feature is supported. Refer to [Section 26.3: I2C implementation](#).

A PEC error is detected when the received PEC byte does not match with the I2C\_PECR register content. A NACK is automatically sent after the wrong PEC reception.

When a PEC error is detected, the PECERR flag is set in the I2C\_ISR register, and an interrupt is generated if the ERRIE bit is set in the I2C\_CR1 register.

### Timeout error (TIMEOUT)

This section is relevant only when the SMBus feature is supported. Refer to [Section 26.3: I2C implementation](#).

A timeout error occurs for any of these conditions:

- TIDLE = 0 and SCL remained low for the time defined in the TIMEOUTA[11:0] bits: this is used to detect a SMBus timeout.
- TIDLE = 1 and both SDA and SCL remained high for the time defined in the TIMEOUTA [11:0] bits: this is used to detect a bus idle condition.
- Master cumulative clock low extend time reached the time defined in the TIMEOUTB[11:0] bits (SMBus  $t_{LOW:MEXT}$  parameter)
- Slave cumulative clock low extend time reached the time defined in TIMEOUTB[11:0] bits (SMBus  $t_{LOW:SEXT}$  parameter)

When a timeout violation is detected in master mode, a STOP condition is automatically sent.

When a timeout violation is detected in slave mode, SDA and SCL lines are automatically released.

When a timeout error is detected, the TIMEOUT flag is set in the I2C\_ISR register, and an interrupt is generated if the ERRIE bit is set in the I2C\_CR1 register.

### Alert (ALERT)

This section is relevant only when the SMBus feature is supported. Refer to [Section 26.3: I2C implementation](#).

The ALERT flag is set when the I2C interface is configured as a Host (`SMBHEN = 1`), the alert pin detection is enabled (`ALERTEN = 1`) and a falling edge is detected on the SMBA pin. An interrupt is generated if the `ERRIE` bit is set in the `I2C_CR1` register.

## 26.4.18 DMA requests

### Transmission using DMA

DMA (direct memory access) can be enabled for transmission by setting the `TXDMAEN` bit in the `I2C_CR1` register. Data is loaded from an SRAM area configured using the DMA peripheral (see [Section 11: Direct memory access controller \(DMA\) on page 359](#)) to the `I2C_TXDR` register whenever the `TXIS` bit is set.

Only the data are transferred with DMA.

- In master mode: the initialization, the slave address, direction, number of bytes and START bit are programmed by software (the transmitted slave address cannot be transferred with DMA). When all data are transferred using DMA, the DMA must be initialized before setting the START bit. The end of transfer is managed with the `NBYTES` counter. Refer to [Master transmitter](#).

For a code example refer to [A.14.8: I2C configured in master mode to transmit with DMA code example](#).

- In slave mode:
  - With `NOSTRETCH = 0`, when all data are transferred using DMA, the DMA must be initialized before the address match event, or in `ADDR` interrupt subroutine, before clearing `ADDR`.
  - With `NOSTRETCH = 1`, the DMA must be initialized before the address match event.
- For instances supporting SMBus: the PEC transfer is managed with `NBYTES` counter. Refer to [SMBus slave transmitter](#) and [SMBus master transmitter](#).

*Note:* If DMA is used for transmission, the `TXIE` bit does not need to be enabled.

### Reception using DMA

DMA (direct memory access) can be enabled for reception by setting the `RXDMAEN` bit in the `I2C_CR1` register. Data is loaded from the `I2C_RXDR` register to an SRAM area configured using the DMA peripheral (refer to [Section 10: Direct memory access controller \(DMA\) on page 188](#)) whenever the `RXNE` bit is set. Only the data (including PEC) are transferred with DMA.

- In Master mode, the initialization, the slave address, direction, number of bytes and START bit are programmed by software. When all data are transferred using DMA, the DMA must be initialized before setting the START bit. The end of transfer is managed with the `NBYTES` counter..
- In Slave mode with `NOSTRETCH = 0`, when all data are transferred using DMA, the DMA must be initialized before the address match event, or in the `ADDR` interrupt subroutine, before clearing the `ADDR` flag.
- If SMBus is supported (see [Section 26.3: I2C implementation](#)): the PEC transfer is managed with the `NBYTES` counter. Refer to [SMBus Slave receiver](#) and [SMBus master receiver](#).

*Note:* If DMA is used for reception, the `RXIE` bit does not need to be enabled.

For a code example refer to [A.14.9: I2C configured in slave mode to receive with DMA code example](#).

#### 26.4.19 Debug mode

When the microcontroller enters debug mode (core halted), the SMBus timeout either continues to work normally or stops, depending on the DBG\_I2Cx\_SMBUS\_TIMEOUT configuration bits in the DBG module.

### 26.5 I2C low-power modes

Table 101. Effect of low-power modes on the I2C

Mode	Description
Sleep	No effect. I2C interrupts cause the device to exit the Sleep mode.
Stop <sup>(1)</sup>	The I2C registers content is kept. If WUPEN = 1 and I2C is clocked by an internal oscillator (HSI): the address recognition is functional. The I2C address match condition causes the device to exit the Stop mode. If WUPEN = 0: the I2C must be disabled before entering Stop mode
Standby	The I2C peripheral is powered down and must be reinitialized after exiting Standby mode.

1. Refer to [Section 26.3](#) for information about the Stop modes supported by each instance. If wake-up from a specific Stop mode is not supported, the instance must be disabled before entering this Stop mode.

## 26.6 I2C interrupts

The table below gives the list of I2C interrupt requests.

**Table 102. I2C Interrupt requests**

Interrupt acronym	Interrupt event	Event flag	Enable control bit	Interrupt clear method	Exit the Sleep mode	Exit the Stop mode	Exit the Standby mode
I2C	I2C_EV	Receive buffer not empty	RXNE	RXIE	Read I2C_RXDR register	Yes	No
		Transmit buffer interrupt status	TXIS	TXIE	Write I2C_TXDR register		
		Stop detection interrupt flag	STOPF	STOPIE	Write STOPCF=1		
		Transfer complete reload	TCR	TCIE	Write I2C_CR2 with NBYTES[7:0] ≠ 0		
		Transfer complete	TC		Write START=1 or STOP=1		
		Address matched	ADDR	ADDRIE	Write ADDRCF=1		Yes <sup>(1)</sup>
		NACK reception	NACKF	NACKIE	Write NACKCF=1		No
I2C	I2C_ER	Bus error	BERR	ERRIE	Write BERRCF=1	Yes	No
		Arbitration loss	ARLO		Write ARLOCF=1		
		Overrun/Underrun	OVR		Write OVRCF=1		
		PEC error	PECERR		Write PECERRCF=1		
		Timeout/t <sub>LOW</sub> error	TIMEOUT		Write TIMEOUTCF=1		
		SMBus alert	ALERT		Write ALERTCF=1		

1. The ADDR match event can wake up the device from Stop mode only if the I2C instance supports the Wake-up from Stop mode feature. Refer to [Section 26.3: I2C implementation](#).

## 26.7 I2C registers

Refer to [Section 1.2 on page 42](#) for a list of abbreviations used in register descriptions.

The peripheral registers are accessed by words (32-bit).

### 26.7.1 I2C control register 1 (I2C\_CR1)

Address offset: 0x000

Reset value: 0x0000 0000

Access: No wait states, except if a write access occurs while a write access to this register is ongoing. In this case, wait states are inserted in the second write access until the previous one is completed. The latency of the second write access can be up to  $2 \times \text{PCLK1} + 6 \times \text{I2CCLK}$ .

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PECEN	ALERT EN	SMBD EN	SMBH EN	GCEN	WUPE N	NOSTR ETCH	SBC
								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RXDMA EN	TXDMA EN	Res.	ANF OFF	DNF[3:0]				ERRIE	TCIE	STOP IE	NACK IE	ADDR IE	RXIE	TXIE	PE
rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 Reserved, must be kept at reset value.

Bit 23 **PECEN**: PEC enable

- 0: PEC calculation disabled
- 1: PEC calculation enabled

*Note:* If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'. Refer to [Section 26.3: I2C implementation](#).

Bit 22 **ALERTEN**: SMBus alert enable

- 0: The SMBus alert pin (SMBA) is not supported in host mode (SMBHEN = 1). In device mode (SMBHEN=0), the SMBA pin is released and the Alert Response Address header is disabled (0001100x followed by NACK).
- 1: The SMBus alert pin is supported in host mode (SMBHEN = 1). In device mode (SMBHEN=0), the SMBA pin is driven low and the Alert Response Address header is enabled (0001100x followed by ACK).

*Note:* When ALERTEN=0, the SMBA pin can be used as a standard GPIO.

*If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'.* Refer to [Section 26.3: I2C implementation](#).

Bit 21 **SMBDEN**: SMBus device default address enable

- 0: Device default address disabled. Address 0b1100001x is NACKed.
- 1: Device default address enabled. Address 0b1100001x is ACKed.

*Note:* If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'. Refer to [Section 26.3: I2C implementation](#).

Bit 20 **SMBHEN**: SMBus host address enable

- 0: Host address disabled. Address 0b0001000x is NACKed.
- 1: Host address enabled. Address 0b0001000x is ACKed.

*Note:* If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'. Refer to [Section 26.3: I2C implementation](#).

Bit 19 **GCEN**: General call enable

- 0: General call disabled. Address 0b00000000 is NACKed.
- 1: General call enabled. Address 0b00000000 is ACKed.

Bit 18 **WUPEN**: Wake-up from Stop mode enable

- 0: Wake-up from Stop mode disable.
- 1: Wake-up from Stop mode enable.

*Note: If the Wake-up from Stop mode feature is not supported, this bit is reserved and forced by hardware to '0'. Refer to Section 26.3: I2C implementation.*

*Note: WUPEN can be set only when DNF = '0000'*

Bit 17 **NOSTRETCH**: Clock stretching disable

This bit is used to disable clock stretching in slave mode. It must be kept cleared in master mode.

- 0: Clock stretching enabled
- 1: Clock stretching disabled

*Note: This bit can only be programmed when the I2C is disabled (PE = 0).*

Bit 16 **SBC**: Slave byte control

This bit is used to enable hardware byte control in slave mode.

- 0: Slave byte control disabled
- 1: Slave byte control enabled

Bit 15 **RXDMAEN**: DMA reception requests enable

- 0: DMA mode disabled for reception
- 1: DMA mode enabled for reception

Bit 14 **TXDMAEN**: DMA transmission requests enable

- 0: DMA mode disabled for transmission
- 1: DMA mode enabled for transmission

Bit 13 Reserved, must be kept at reset value.

Bit 12 **ANFOFF**: Analog noise filter OFF

- 0: Analog noise filter enabled
- 1: Analog noise filter disabled

*Note: This bit can only be programmed when the I2C is disabled (PE = 0).*

Bits 11:8 **DNF[3:0]**: Digital noise filter

These bits are used to configure the digital noise filter on SDA and SCL input. The digital filter, filters spikes with a length of up to  $DNF[3:0] * t_{I2CCLK}$

0000: Digital filter disabled

0001: Digital filter enabled and filtering capability up to 1  $t_{I2CCLK}$

..  
1111: digital filter enabled and filtering capability up to 15  $t_{I2CCLK}$

*Note: If the analog filter is also enabled, the digital filter is added to the analog filter.*

*This filter can only be programmed when the I2C is disabled (PE = 0).*

Bit 7 **ERRIE**: Error interrupts enable

- 0: Error detection interrupts disabled
- 1: Error detection interrupts enabled

*Note: Any of these errors generate an interrupt:*

- Arbitration Loss (ARLO)*
- Bus Error detection (BERR)*
- Overrun/Underrun (OVR)*
- Timeout detection (TIMEOUT)*
- PEC error detection (PECERR)*
- Alert pin event detection (ALERT)*

Bit 6 **TCIE**: Transfer complete interrupt enable

- 0: Transfer complete interrupt disabled
- 1: Transfer complete interrupt enabled

*Note: Any of these events generate an interrupt:*

- Transfer complete (TC)*
- Transfer complete reload (TCR)*

Bit 5 **STOPIE**: Stop detection Interrupt enable

- 0: Stop detection (STOPF) interrupt disabled
- 1: Stop detection (STOPF) interrupt enabled

Bit 4 **NACKIE**: Not acknowledge received Interrupt enable

- 0: Not acknowledge (NACKF) received interrupts disabled
- 1: Not acknowledge (NACKF) received interrupts enabled

Bit 3 **ADDRIE**: Address match Interrupt enable (slave only)

- 0: Address match (ADDR) interrupts disabled
- 1: Address match (ADDR) interrupts enabled

Bit 2 **RXIE**: RX Interrupt enable

- 0: Receive (RXNE) interrupt disabled
- 1: Receive (RXNE) interrupt enabled

Bit 1 **TXIE**: TX Interrupt enable

- 0: Transmit (TXIS) interrupt disabled
- 1: Transmit (TXIS) interrupt enabled

Bit 0 **PE**: Peripheral enable

- 0: Peripheral disable
- 1: Peripheral enable

*Note: When PE = 0, the I2C SCL and SDA lines are released. Internal state machines and status bits are put back to their reset value. When cleared, PE must be kept low for at least three APB clock cycles.*

## 26.7.2 I2C control register 2 (I2C\_CR2)

Address offset: 0x04

Reset value: 0x0000 0000

Access: No wait states, except if a write access occurs while a write access to this register is ongoing. In this case, wait states are inserted in the second write access until the previous one is completed. The latency of the second write access can be up to 2 x PCLK1 + 6 x I2CCLK.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	PEC BYTE	AUTO END	RE LOAD	NBYTES[7:0]							
					rs	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NACK	STOP	START	HEAD1 OR	ADD10	RD_ WRN	SADD[9:0]									
rs	rs	rs	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bit 26 **PECBYTE**: Packet error checking byte

This bit is set by software, and cleared by hardware when the PEC is transferred, or when a STOP condition or an Address matched is received, also when PE = 0.

0: No PEC transfer.

1: PEC transmission/reception is requested

*Note: Writing '0' to this bit has no effect.*

*This bit has no effect when RELOAD is set.*

*This bit has no effect in slave mode when SBC = 0.*

*If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'. Refer to [Section 26.3: I2C implementation](#).*

Bit 25 **AUTOEND**: Automatic end mode (master mode)

This bit is set and cleared by software.

0: software end mode: TC flag is set when NBYTES data are transferred, stretching SCL low.

1: Automatic end mode: a STOP condition is automatically sent when NBYTES data are transferred.

*Note: This bit has no effect in slave mode or when the RELOAD bit is set.*

Bit 24 **RELOAD**: NBYTES reload mode

This bit is set and cleared by software.

0: The transfer is completed after the NBYTES data transfer (STOP or RESTART follows).

1: The transfer is not completed after the NBYTES data transfer (NBYTES is reloaded). TCR flag is set when NBYTES data are transferred, stretching SCL low.

Bits 23:16 **NBYTES[7:0]**: Number of bytes

The number of bytes to be transmitted/received is programmed there. This field is don't care in slave mode with SBC = 0.

*Note: Changing these bits when the START bit is set is not allowed.*

**Bit 15 NACK:** NACK generation (slave mode)

The bit is set by software, cleared by hardware when the NACK is sent, or when a STOP condition or an Address matched is received, or when PE = 0.

- 0: an ACK is sent after current received byte.
- 1: a NACK is sent after current received byte.

*Note: Writing '0' to this bit has no effect.*

*This bit is used in slave mode only: in master receiver mode, NACK is automatically generated after last byte preceding STOP or RESTART condition, whatever the NACK bit value.*

*When an overrun occurs in slave receiver NOSTRETCH mode, a NACK is automatically generated whatever the NACK bit value.*

*When hardware PEC checking is enabled (PECBYTE = 1), the PEC acknowledge value does not depend on the NACK value.*

**Bit 14 STOP:** Stop generation (master mode)

The bit is set by software, cleared by hardware when a STOP condition is detected, or when PE = 0.

**In Master mode:**

- 0: No Stop generation.
- 1: Stop generation after current byte transfer.

*Note: Writing '0' to this bit has no effect.*

**Bit 13 START:** Start generation

This bit is set by software, and cleared by hardware after the Start followed by the address sequence is sent, by an arbitration loss, by a timeout error detection, or when PE = 0. It can also be cleared by software by writing '1' to the ADDRCF bit in the I2C\_ICR register.

- 0: No Start generation.
- 1: Restart/Start generation:

If the I2C is already in master mode with AUTOEND = 0, setting this bit generates a Repeated start condition when RELOAD=0, after the end of the NBYTES transfer.

Otherwise setting this bit generates a START condition once the bus is free.

*Note: Writing '0' to this bit has no effect.*

*The START bit can be set even if the bus is BUSY or I2C is in slave mode.*

*This bit has no effect when RELOAD is set.*

**Bit 12 HEAD10R:** 10-bit address header only read direction (master receiver mode)

0: The master sends the complete 10 bit slave address read sequence: Start + 2 bytes 10bit address in write direction + Restart + 1st 7 bits of the 10 bit address in read direction.

- 1: The master only sends the 1st 7 bits of the 10 bit address, followed by Read direction.

*Note: Changing this bit when the START bit is set is not allowed.*

**Bit 11 ADD10:** 10-bit addressing mode (master mode)

- 0: The master operates in 7-bit addressing mode,
- 1: The master operates in 10-bit addressing mode

*Note: Changing this bit when the START bit is set is not allowed.*

**Bit 10 RD\_WRN:** Transfer direction (master mode)

- 0: Master requests a write transfer.
- 1: Master requests a read transfer.

*Note: Changing this bit when the START bit is set is not allowed.*

Bits 9:0 **SADD[9:0]**: Slave address (master mode)

**In 7-bit addressing mode (ADD10 = 0):**

SADD[7:1] must be written with the 7-bit slave address to be sent. The bits SADD[9], SADD[8] and SADD[0] are don't care.

**In 10-bit addressing mode (ADD10 = 1):**

SADD[9:0] must be written with the 10-bit slave address to be sent.

*Note: Changing these bits when the START bit is set is not allowed.*

### 26.7.3 I2C own address 1 register (I2C\_OAR1)

Address offset: 0x08

Reset value: 0x0000 0000

Access: No wait states, except if a write access occurs while a write access to this register is ongoing. In this case, wait states are inserted in the second write access until the previous one is completed. The latency of the second write access can be up to 2 x PCLK1 + 6 x I2CCLK.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OA1EN	Res.	Res.	Res.	Res.	OA1 MODE	OA1[9:0]									
rw					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bit 15 **OA1EN**: Own address 1 enable

- 0: Own address 1 disabled. The received slave address OA1 is NACKed.
- 1: Own address 1 enabled. The received slave address OA1 is ACKed.

Bits 14:11 Reserved, must be kept at reset value.

Bit 10 **OA1MODE**: Own address 1 10-bit mode

- 0: Own address 1 is a 7-bit address.
- 1: Own address 1 is a 10-bit address.

*Note: This bit can be written only when OA1EN = 0.*

Bits 9:0 **OA1[9:0]**: Interface own slave address

7-bit addressing mode: OA1[7:1] contains the 7-bit own slave address. The bits OA1[9], OA1[8] and OA1[0] are don't care.

10-bit addressing mode: OA1[9:0] contains the 10-bit own slave address.

*Note: These bits can be written only when OA1EN = 0.*

### 26.7.4 I2C own address 2 register (I2C\_OAR2)

Address offset: 0x0C

Reset value: 0x0000 0000

Access: No wait states, except if a write access occurs while a write access to this register is ongoing. In this case, wait states are inserted in the second write access, until the previous one is completed. The latency of the second write access can be up to 2x PCLK1 + 6 x I2CCLK.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OA2EN	Res.	Res.	Res.	Res.	OA2MSK[2:0]			OA2[7:1]							
rw					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bit 15 **OA2EN**: Own address 2 enable

- 0: Own address 2 disabled. The received slave address OA2 is NACKed.
- 1: Own address 2 enabled. The received slave address OA2 is ACKed.

Bits 14:11 Reserved, must be kept at reset value.

Bits 10:8 **OA2MSK[2:0]**: Own address 2 masks

- 000: No mask
- 001: OA2[1] is masked and don't care. Only OA2[7:2] are compared.
- 010: OA2[2:1] are masked and don't care. Only OA2[7:3] are compared.
- 011: OA2[3:1] are masked and don't care. Only OA2[7:4] are compared.
- 100: OA2[4:1] are masked and don't care. Only OA2[7:5] are compared.
- 101: OA2[5:1] are masked and don't care. Only OA2[7:6] are compared.
- 110: OA2[6:1] are masked and don't care. Only OA2[7] is compared.
- 111: OA2[7:1] are masked and don't care. No comparison is done, and all (except reserved) 7-bit received addresses are acknowledged.

*Note: These bits can be written only when OA2EN = 0.*

*As soon as OA2MSK is not equal to 0, the reserved I2C addresses (0b0000xxx and 0b1111xxx) are not acknowledged even if the comparison matches.*

Bits 7:1 **OA2[7:1]**: Interface address

7-bit addressing mode: 7-bit address

*Note: These bits can be written only when OA2EN = 0.*

Bit 0 Reserved, must be kept at reset value.

### 26.7.5 I2C timing register (I2C\_TIMINGR)

Address offset: 0x10

Reset value: 0x0000 0000

Access: No wait states

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PRESC[3:0]				Res.	Res.	Res.	Res.	SCLDEL[3:0]				SDADEL[3:0]			
rw	rw	rw	rw					rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SCLH[7:0]								SCLL[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:28 **PRESC[3:0]**: Timing prescaler

This field is used to prescale I2CCLK in order to generate the clock period  $t_{PRESC}$  used for data setup and hold counters (refer to [I2C timings](#)) and for SCL high and low level counters (refer to [I2C master initialization](#)).

$$t_{PRESC} = (\text{PRESC}+1) \times t_{I2CCLK}$$

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

Bits 23:20 **SCLDEL[3:0]**: Data setup time

This field is used to generate a delay  $t_{SCLDEL}$  between SDA edge and SCL rising edge. In master mode and in slave mode with NOSTRETCH = 0, the SCL line is stretched low during  $t_{SCLDEL}$ .

$$t_{SCLDEL} = (\text{SCLDEL}+1) \times t_{PRESC}$$

Note:  $t_{SCLDEL}$  is used to generate  $t_{SU:DAT}$  timing.

Bits 19:16 **SDADEL[3:0]**: Data hold time

This field is used to generate the delay  $t_{SDADEL}$  between SCL falling edge and SDA edge. In master mode and in slave mode with NOSTRETCH = 0, the SCL line is stretched low during  $t_{SDADEL}$ .

$$t_{SDADEL} = \text{SDADEL} \times t_{PRESC}$$

Note:  $t_{SDADEL}$  is used to generate  $t_{HD:DAT}$  timing.

Bits 15:8 **SCLH[7:0]**: SCL high period (master mode)

This field is used to generate the SCL high period in master mode.

$$t_{SCLH} = (\text{SCLH}+1) \times t_{PRESC}$$

Note:  $t_{SCLH}$  is also used to generate  $t_{SU:STO}$  and  $t_{HD:STA}$  timing.

Bits 7:0 **SCLL[7:0]**: SCL low period (master mode)

This field is used to generate the SCL low period in master mode.

$$t_{SCLL} = (\text{SCLL}+1) \times t_{PRESC}$$

Note:  $t_{SCLL}$  is also used to generate  $t_{BUF}$  and  $t_{SU:STA}$  timings.

Note: This register must be configured when the I2C is disabled ( $PE = 0$ ).

Note: The STM32CubeMX tool calculates and provides the I2C\_TIMINGR content in the I2C Configuration window.

## 26.7.6 I2C timeout register (I2C\_TIMEOUTR)

Address offset: 0x14

Reset value: 0x0000 0000

Access: No wait states, except if a write access occurs while a write access to this register is ongoing. In this case, wait states are inserted in the second write access until the previous one is completed. The latency of the second write access can be up to  $2 \times \text{PCLK1} + 6 \times \text{I2CCLK}$ .

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TEXTEN	Res.	Res.	Res.	TIMEOUTB[11:0]												
TIMOUTEN	Res.	Res.	TIDLE	TIMEOUTA[11:0]												
rw			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Bit 31 **TEXTEN**: Extended clock timeout enable

0: Extended clock timeout detection is disabled

1: Extended clock timeout detection is enabled. When a cumulative SCL stretch for more than  $t_{\text{LOW:EXT}}$  is done by the I2C interface, a timeout error is detected (TIMEOUT=1).

Bits 30:28 Reserved, must be kept at reset value.

Bits 27:16 **TIMEOUTB[11:0]**: Bus timeout B

This field is used to configure the cumulative clock extension timeout:

In master mode, the master cumulative clock low extend time ( $t_{\text{LOW:MEXT}}$ ) is detected

In slave mode, the slave cumulative clock low extend time ( $t_{\text{LOW:SEXT}}$ ) is detected

$t_{\text{LOW:EXT}} = (\text{TIMEOUTB}+1) \times 2048 \times t_{\text{I2CCLK}}$

*Note: These bits can be written only when TEXTEN=0.*

Bit 15 **TIMOUTEN**: Clock timeout enable

0: SCL timeout detection is disabled

1: SCL timeout detection is enabled: when SCL is low for more than  $t_{\text{TIMEOUT}}$  (TIDLE=0) or high for more than  $t_{\text{IDLE}}$  (TIDLE=1), a timeout error is detected (TIMEOUT=1).

Bits 14:13 Reserved, must be kept at reset value.

Bit 12 **TIDLE**: Idle clock timeout detection

0: TIMEOUTA is used to detect SCL low timeout

1: TIMEOUTA is used to detect both SCL and SDA high timeout (bus idle condition)

*Note: This bit can be written only when TIMOUTEN=0.*

Bits 11:0 **TIMEOUTA[11:0]**: Bus Timeout A

This field is used to configure:

The SCL low timeout condition  $t_{\text{TIMEOUT}}$  when TIDLE=0

$t_{\text{TIMEOUT}} = (\text{TIMEOUTA}+1) \times 2048 \times t_{\text{I2CCLK}}$

The bus idle condition (both SCL and SDA high) when TIDLE=1

$t_{\text{IDLE}} = (\text{TIMEOUTA}+1) \times 4 \times t_{\text{I2CCLK}}$

*Note: These bits can be written only when TIMOUTEN=0.*

**Note:** If the SMBus feature is not supported, this register is reserved and forced by hardware to "0x00000000". Refer to [Section 26.3: I2C implementation](#).

### 26.7.7 I2C interrupt and status register (I2C\_ISR)

Address offset: 0x18

Reset value: 0x0000 0001

Access: No wait states

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16							
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ADDCODE[6:0]														DIR
								r	r	r	r	r	r	r	r							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0							
BUSY	Res.	ALERT	TIME OUT	PEC ERR	OVR	ARLO	BERR	TCR	TC	STOPF	NACKF	ADDR	RXNE	TXIS	TXE							
r		r	r	r	r	r	r	r	r	r	r	r	r	rs	rs							

Bits 31:24 Reserved, must be kept at reset value.

Bits 23:17 ADDCODE[6:0]: Address match code (Slave mode)

These bits are updated with the received address when an address match event occurs (ADDR = 1).

In the case of a 10-bit address, ADDCODE provides the 10-bit header followed by the 2 MSBs of the address.

Bit 16 DIR: Transfer direction (Slave mode)

This flag is updated when an address match event occurs (ADDR = 1).

0: Write transfer, slave enters receiver mode.

1: Read transfer, slave enters transmitter mode.

Bit 15 BUSY: Bus busy

This flag indicates that a communication is in progress on the bus. It is set by hardware when a START condition is detected. It is cleared by hardware when a STOP condition is detected, or when PE = 0.

Bit 14 Reserved, must be kept at reset value.

Bit 13 ALERT: SMBus alert

This flag is set by hardware when SMBHEN=1 (SMBus host configuration), ALERTEN=1 and a SMBALERT event (falling edge) is detected on SMBA pin. It is cleared by software by setting the ALERTCF bit.

*Note: This bit is cleared by hardware when PE = 0.*

*If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'.*

*Refer to Section 26.3: I2C implementation.*

Bit 12 TIMEOUT: Timeout or t<sub>LOW</sub> detection flag

This flag is set by hardware when a timeout or extended clock timeout occurred. It is cleared by software by setting the TIMEOUTCF bit.

*Note: This bit is cleared by hardware when PE = 0.*

*If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'.*

*Refer to Section 26.3: I2C implementation.*

Bit 11 **PECERR**: PEC Error in reception

This flag is set by hardware when the received PEC does not match with the PEC register content. A NACK is automatically sent after the wrong PEC reception. It is cleared by software by setting the PECCF bit.

*Note: This bit is cleared by hardware when PE = 0.*

*If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'. Refer to Section 26.3: I2C implementation.*

Bit 10 **OVR**: Overrun/Underrun (slave mode)

This flag is set by hardware in slave mode with NOSTRETCH = 1, when an overrun/underrun error occurs. It is cleared by software by setting the OVRCF bit.

*Note: This bit is cleared by hardware when PE = 0.*

Bit 9 **ARLO**: Arbitration lost

This flag is set by hardware in case of arbitration loss. It is cleared by software by setting the ARLOCF bit.

*Note: This bit is cleared by hardware when PE = 0.*

Bit 8 **BERR**: Bus error

This flag is set by hardware when a misplaced Start or STOP condition is detected whereas the peripheral is involved in the transfer. The flag is not set during the address phase in slave mode. It is cleared by software by setting the BERRCF bit.

*Note: This bit is cleared by hardware when PE = 0.*

Bit 7 **TCR**: Transfer Complete Reload

This flag is set by hardware when RELOAD=1 and NBYTES data have been transferred. It is cleared by software when NBYTES is written to a non-zero value.

*Note: This bit is cleared by hardware when PE = 0.*

*This flag is only for master mode, or for slave mode when the SBC bit is set.*

Bit 6 **TC**: Transfer Complete (master mode)

This flag is set by hardware when RELOAD=0, AUTOEND=0 and NBYTES data have been transferred. It is cleared by software when START bit or STOP bit is set.

*Note: This bit is cleared by hardware when PE = 0.*

Bit 5 **STOPF**: Stop detection flag

This flag is set by hardware when a STOP condition is detected on the bus and the peripheral is involved in this transfer:

- either as a master, provided that the STOP condition is generated by the peripheral.
- or as a slave, provided that the peripheral has been addressed previously during this transfer.

It is cleared by software by setting the STOPCF bit.

*Note: This bit is cleared by hardware when PE = 0.*

Bit 4 **NACKF**: Not Acknowledge received flag

This flag is set by hardware when a NACK is received after a byte transmission. It is cleared by software by setting the NACKCF bit.

*Note: This bit is cleared by hardware when PE = 0.*

Bit 3 **ADDR**: Address matched (slave mode)

This bit is set by hardware as soon as the received slave address matched with one of the enabled slave addresses. It is cleared by software by setting the ADDRRCF bit.

*Note: This bit is cleared by hardware when PE = 0.*

Bit 2 **RXNE**: Receive data register not empty (receivers)

This bit is set by hardware when the received data is copied into the I2C\_RXDR register, and is ready to be read. It is cleared when I2C\_RXDR is read.

*Note: This bit is cleared by hardware when PE = 0.*

Bit 1 **TXIS**: Transmit interrupt status (transmitters)

This bit is set by hardware when the I2C\_TXDR register is empty and the data to be transmitted must be written in the I2C\_TXDR register. It is cleared when the next data to be sent is written in the I2C\_TXDR register.

This bit can be written to '1' by software when NOSTRETCH = 1 only, in order to generate a TXIS event (interrupt if TXIE=1 or DMA request if TXDMAEN = 1).

*Note: This bit is cleared by hardware when PE = 0.*

Bit 0 **TXE**: Transmit data register empty (transmitters)

This bit is set by hardware when the I2C\_TXDR register is empty. It is cleared when the next data to be sent is written in the I2C\_TXDR register.

This bit can be written to '1' by software in order to flush the transmit data register I2C\_TXDR.

*Note: This bit is set by hardware when PE = 0.*

## 26.7.8 I2C interrupt clear register (I2C\_ICR)

Address offset: 0x1C

Reset value: 0x0000 0000

Access: No wait states

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	ALERT CF	TIMOU TCF	PECCF	OVRCF	ARLOC F	BERRCF	Res.	Res.	STOPCF	NACKCF	ADDR CF	Res.	Res.	Res.
		w	w	w	w	w	w			w	w	w			

Bits 31:14 Reserved, must be kept at reset value.

Bit 13 **ALERTCF**: Alert flag clear

Writing 1 to this bit clears the ALERT flag in the I2C\_ISR register.

*Note: If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'. Refer to Section 26.3: I2C implementation.*

Bit 12 **TIMOUTCF**: Timeout detection flag clear

Writing 1 to this bit clears the TIMEOUT flag in the I2C\_ISR register.

*Note: If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'. Refer to Section 26.3: I2C implementation.*

Bit 11 **PECCF**: PEC Error flag clear

Writing 1 to this bit clears the PECERR flag in the I2C\_ISR register.

*Note: If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'. Refer to Section 26.3: I2C implementation.*

- Bit 10 **OVRCF**: Overrun/Underrun flag clear  
Writing 1 to this bit clears the OVR flag in the I2C\_ISR register.
- Bit 9 **ARLOCF**: Arbitration lost flag clear  
Writing 1 to this bit clears the ARLO flag in the I2C\_ISR register.
- Bit 8 **BERRCF**: Bus error flag clear  
Writing 1 to this bit clears the BERRF flag in the I2C\_ISR register.
- Bits 7:6 Reserved, must be kept at reset value.
- Bit 5 **STOPCF**: STOP detection flag clear  
Writing 1 to this bit clears the STOPF flag in the I2C\_ISR register.
- Bit 4 **NACKCF**: Not Acknowledge flag clear  
Writing 1 to this bit clears the NACKF flag in I2C\_ISR register.
- Bit 3 **ADDRCF**: Address matched flag clear  
Writing 1 to this bit clears the ADDR flag in the I2C\_ISR register. Writing 1 to this bit also clears the START bit in the I2C\_CR2 register.
- Bits 2:0 Reserved, must be kept at reset value.

### 26.7.9 I2C PEC register (I2C\_PECR)

Address offset: 0x20

Reset value: 0x0000 0000

Access: No wait states

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	PEC[7:0]														
								r	r	r	r	r	r	r	r

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

Bits 7:0 **PEC[7:0]**: Packet error checking register

This field contains the internal PEC when PECEN=1.

The PEC is cleared by hardware when PE = 0.

**Note:** If the SMBus feature is not supported, this register is reserved and forced by hardware to “0x00000000”. Refer to [Section 26.3: I2C implementation](#).

### 26.7.10 I2C receive data register (I2C\_RXDR)

Address offset: 0x24

Reset value: 0x0000 0000

Access: No wait states

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

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

Bits 7:0 **RXDATA[7:0]**: 8-bit receive data

Data byte received from the I<sup>2</sup>C bus

### 26.7.11 I2C transmit data register (I2C\_TXDR)

Address offset: 0x28

Reset value: 0x0000 0000

Access: No wait states

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

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

Bits 7:0 **TXDATA[7:0]**: 8-bit transmit data

Data byte to be transmitted to the I<sup>2</sup>C bus

*Note:* These bits can be written only when TXE = 1.

### 26.7.12 I2C register map

The table below provides the I2C register map and reset values.

**Table 103. I2C register map and reset values**

**Table 103. I2C register map and reset values (continued)**

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x28	I2C_TXDR	Res.	TXDATA[7:0]																														
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 27 Universal synchronous/asynchronous receiver transmitter (USART/UART)

### 27.1 Introduction

The universal synchronous asynchronous receiver transmitter (USART) offers a flexible means of Full-duplex data exchange with external equipment requiring an industry standard NRZ asynchronous serial data format. The USART offers a very wide range of baud rates using a programmable baud rate generator.

It supports synchronous one-way communication and Half-duplex Single-wire communication, as well as multiprocessor communications. It also supports the LIN (Local Interconnect Network), Smartcard protocol and IrDA (Infrared Data Association) SIR ENDEC specifications and Modem operations (CTS/RTS).

High speed data communication is possible by using the DMA (direct memory access) for multibuffer configuration.

### 27.2 USART main features

- Full-duplex asynchronous communications
- NRZ standard format (mark/space)
- Configurable oversampling method by 16 or 8 to give flexibility between speed and clock tolerance
- A common programmable transmit and receive baud rate of up to 6 Mbit/s when the clock frequency is 48 MHz and oversampling is by 8
- Dual clock domain allowing:
  - USART functionality and wake-up from Stop mode
  - Convenient baud rate programming independent from the PCLK reprogramming
- Auto baud rate detection
- Programmable data word length (7, 8 or 9 bits)
- Programmable data order with MSB-first or LSB-first shifting
- Configurable stop bits (1 or 2 stop bits)
- Synchronous mode and clock output for synchronous communications
- Single-wire Half-duplex communications
- Continuous communications using DMA
- Received/transmitted bytes are buffered in reserved SRAM using centralized DMA
- Separate enable bits for transmitter and receiver
- Separate signal polarity control for transmission and reception
- Swappable Tx/Rx pin configuration
- Hardware flow control for modem and RS-485 transceiver

- Communication control/error detection flags
- Parity control:
  - Transmits parity bit
  - Checks parity of received data byte
- Fourteen interrupt sources with flags
- Multiprocessor communications
  - The USART enters Mute mode if the address does not match.
- Wake-up from Mute mode (by idle line detection or address mark detection)

## 27.3 USART extended features

- LIN master synchronous break send capability and LIN slave break detection capability
  - 13-bit break generation and 10/11-bit break detection when USART is hardware configured for LIN
- IrDA SIR encoder decoder supporting 3/16 bit duration for normal mode
- Smartcard mode
  - Supports the T=0 and T=1 asynchronous protocols for smartcards as defined in the ISO/IEC 7816-3 standard
  - 0.5 and 1.5 stop bits for smartcard operation
- Support for ModBus communication
  - Timeout feature
  - CR/LF character recognition

## 27.4 USART implementation

Table 104. STM32F0xx USART features<sup>(1)</sup>

USART modes/ features	STM32F03x	STM32F05x		STM32F04x		STM32F07x		STM32F09x		
	USART1	USART1	USART2	USART1	USART2	USART1/ USART2	USART3/USART4	USART1/USART2/ USART3	USART4	USART5/USART6/ USART7/ USART8
Hardware flow control for modem	X	X	X	X	X	X	X	X	X	-
Continuous communication using DMA	X	X	X	X	X	X	X	X	X	X
Multiprocessor communication	X	X	X	X	X	X	X	X	X	X
Synchronous mode	X	X	X	X	X	X	X	X	X	X
Smartcard mode	X <sup>(2)</sup>	X <sup>(2)</sup>	-	X <sup>(3)</sup>	-	X <sup>(3)</sup>	-	X <sup>(3)</sup>	-	-
Single-wire Half-duplex communication	X	X	X	X	X	X	X	X	X	X
IrDA SIR ENDEC block	X	X	-	X	-	X	-	X	-	-
LIN mode	X	X	-	X	-	X	-	X	-	-
Dual clock domain and wake-up from Stop mode	X	X	-	X	-	X	-	X	-	-
Receiver timeout interrupt	X	X	-	X	-	X	-	X	-	-
Modbus communication	X	X	-	X	-	X	-	X	-	-
Auto baud rate detection (supported modes)	2 (Modes 0/1)	2 (Modes 0/1)	-	4	-	4	-	4	-	-
Driver Enable	X	X	X	X	X	X	X	X	X	X
USART data length	8 and 9 bits			7 <sup>(4)</sup> , 8 and 9 bits						

1. X = supported.

2. CK output is disabled when UE bit = 0.

3. CK is always available when CLKEN = 1, regardless of the UE bit value.

4. In 7-bit data length mode, Smartcard mode, LIN master mode and Auto baud rate (0x7F and 0x55 frames) detection are not supported.

## 27.5 USART functional description

Any USART bidirectional communication requires a minimum of two pins: Receive data In (RX) and Transmit data Out (TX):

- **RX:** Receive data Input.  
This is the serial data input. Oversampling techniques are used for data recovery by discriminating between valid incoming data and noise.
- **TX:** Transmit data Output.  
When the transmitter is disabled, the output pin returns to its I/O port configuration. When the transmitter is enabled and nothing is to be transmitted, the TX pin is at high level. In Single-wire and Smartcard modes, this I/O is used to transmit and receive the data.

Serial data are transmitted and received through these pins in normal USART mode. The frames are comprised of:

- An Idle Line prior to transmission or reception
- A start bit
- A data word (7, 8 or 9 bits) least significant bit first
- 0.5, 1, 1.5, 2 stop bits indicating that the frame is complete
- The USART interface uses a baud rate generator
- A status register (USART\_ISR)
- Receive and transmit data registers (USART\_RDR, USART\_TDR)
- A baud rate register (USART\_BRR)
- A guard-time register (USART\_GTPR) in case of Smartcard mode.

Refer to [Section 27.8: USART registers on page 744](#) for the definitions of each bit.

The following pin is required to interface in synchronous mode and Smartcard mode:

- **CK:** Clock output. This pin outputs the transmitter data clock for synchronous transmission corresponding to SPI master mode (no clock pulses on start bit and stop bit, and a software option to send a clock pulse on the last data bit). In parallel, data can be received synchronously on RX. This can be used to control peripherals that have shift registers (e.g. LCD drivers). The clock phase and polarity are software programmable. In Smartcard mode, CK output can provide the clock to the smartcard.

The following pins are required in RS232 Hardware flow control mode:

- **CTS:** Clear To Send blocks the data transmission at the end of the current transfer (when high)
- **RTS:** Request to send indicates that the USART is ready to receive data (when low).

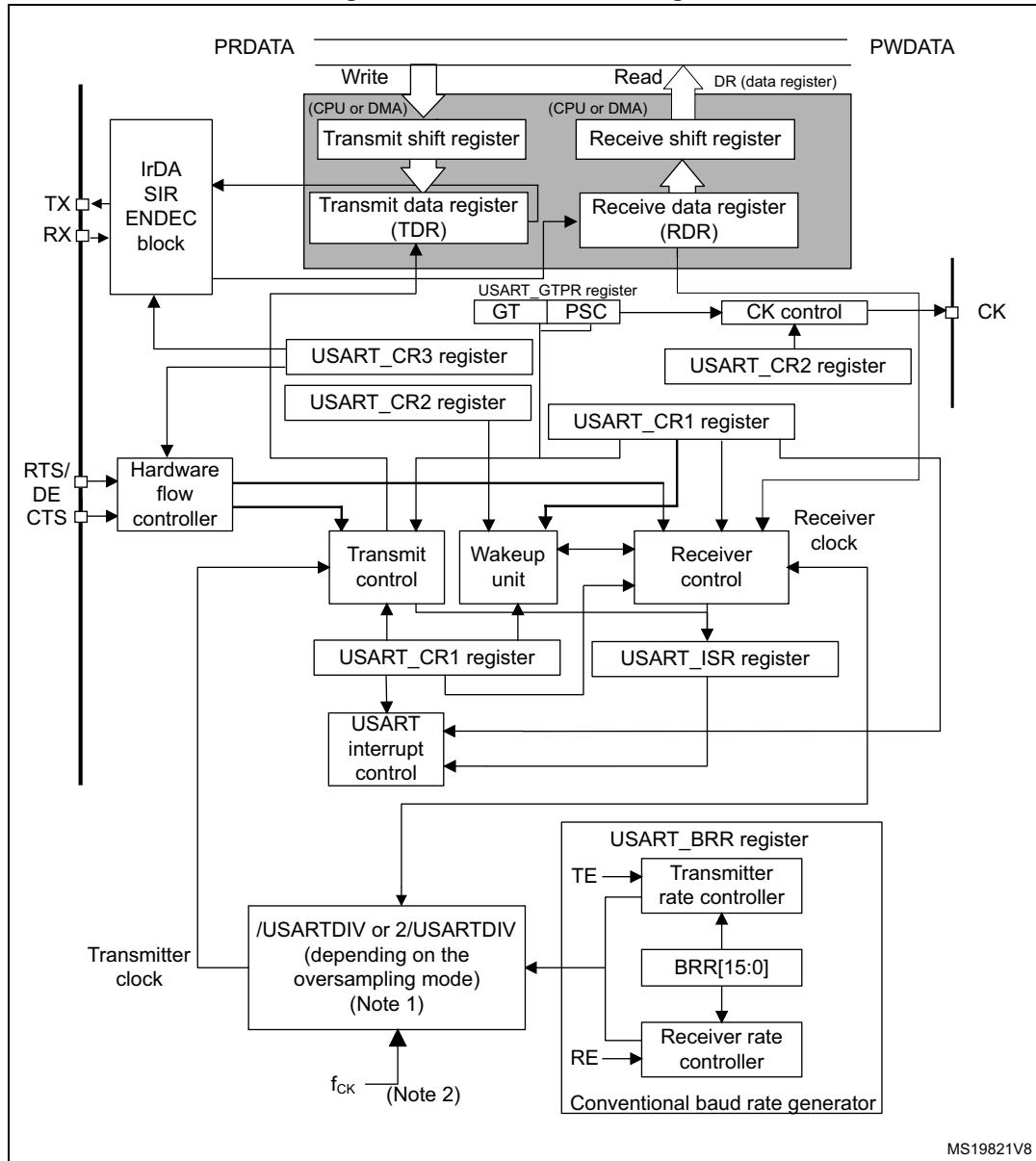
The following pin is required in RS485 Hardware control mode:

- **DE:** Driver Enable activates the transmission mode of the external transceiver.

*Note:*

*DE and RTS share the same pin.*

Figure 248. USART block diagram



1. For details on coding USARTDIV in the USART\_BRR register, refer to [Section 27.5.4: USART baud rate generation](#).
2.  $f_{CK}$  can be  $f_{LSE}$ ,  $f_{HSI}$ ,  $f_{PCLK}$ ,  $f_{SYS}$ .

### 27.5.1 USART character description

The word length can be selected as being either 7 or 8 or 9 bits by programming the M[1:0] bits in the USART\_CR1 register (see [Figure 249](#)).

- 7-bit character length: M[1:0] = 10
- 8-bit character length: M[1:0] = 00
- 9-bit character length: M[1:0] = 01

**Note:** *The 7-bit mode is supported only on some USARTs. In addition, not all modes are supported in 7-bit data length mode. Refer to [Section 27.4: USART implementation](#) for additional information.*

By default, the signal (TX or RX) is in low state during the start bit. It is in high state during the stop bit.

These values can be inverted, separately for each signal, through polarity configuration control.

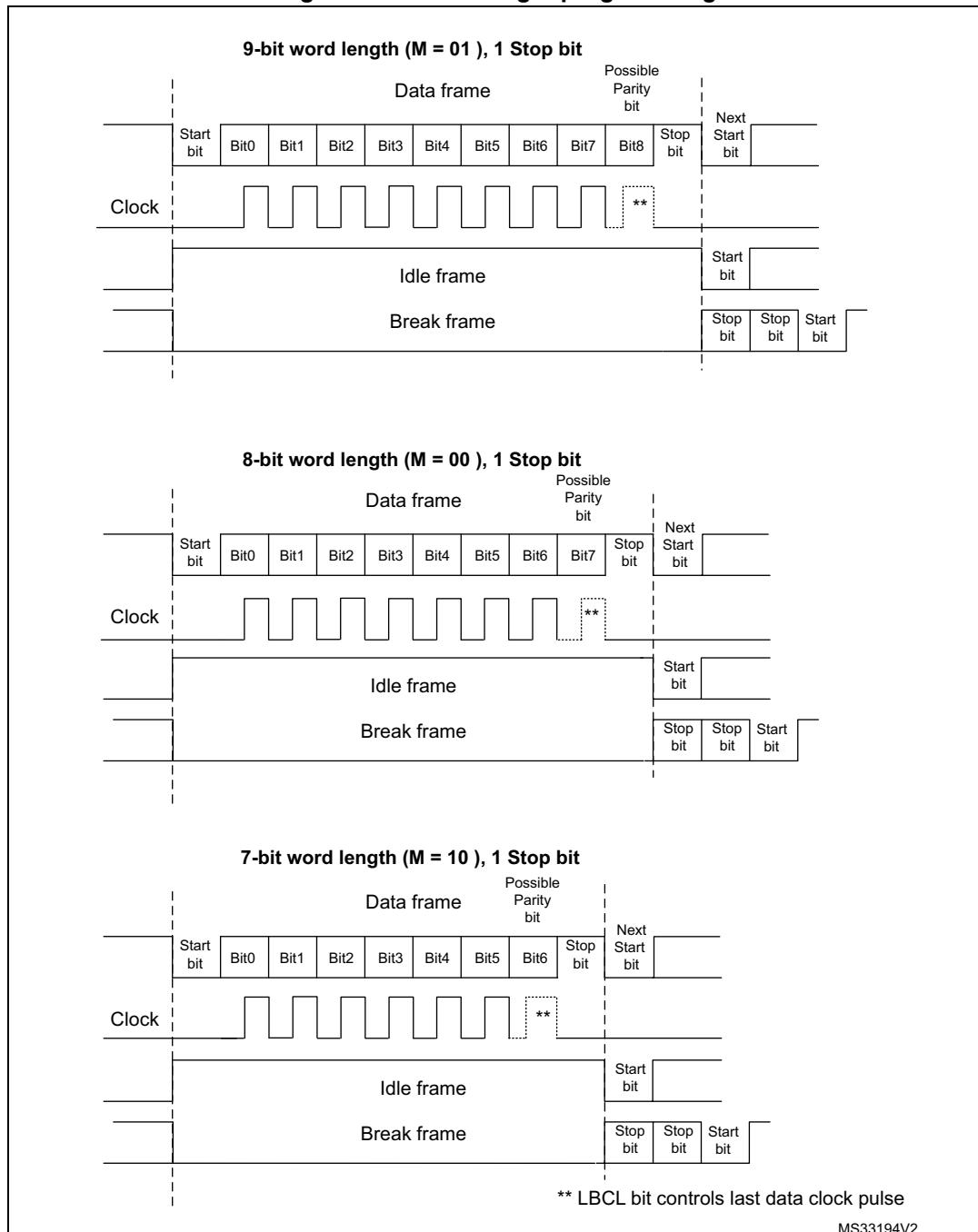
An **Idle character** is interpreted as an entire frame of “1”s (the number of “1”s includes the number of stop bits).

A **Break character** is interpreted on receiving “0”s for a frame period. At the end of the break frame, the transmitter inserts 2 stop bits.

Transmission and reception are driven by a common baud rate generator, the clock for each is generated when the enable bit is set respectively for the transmitter and receiver.

The details of each block is given below.

Figure 249. Word length programming



## 27.5.2 USART transmitter

The transmitter can send data words of either 7, 8 or 9 bits depending on the M bits status. The Transmit Enable bit (TE) must be set in order to activate the transmitter function. The data in the transmit shift register is output on the TX pin and the corresponding clock pulses are output on the CK pin.

### Character transmission

During an USART transmission, data shifts out least significant bit first (default configuration) on the TX pin. In this mode, the USART\_TDR register consists of a buffer (TDR) between the internal bus and the transmit shift register (see [Figure 248](#)).

Every character is preceded by a start bit which is a logic level low for one bit period. The character is terminated by a configurable number of stop bits.

The following stop bits are supported by USART: 0.5, 1, 1.5 and 2 stop bits.

*Note:* *The TE bit must be set before writing the data to be transmitted to the USART\_TDR.*

*The TE bit should not be reset during transmission of data. Resetting the TE bit during the transmission will corrupt the data on the TX pin as the baud rate counters will get frozen. The current data being transmitted is lost.*

*An idle frame is sent after the TE bit is enabled.*

### Configurable stop bits

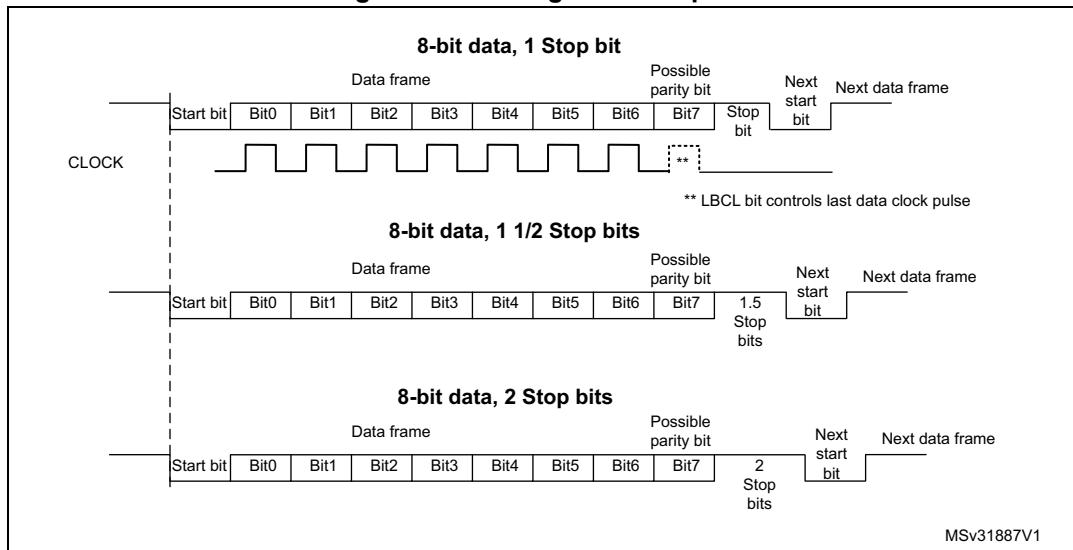
The number of stop bits to be transmitted with every character can be programmed in Control register 2, bits 13,12.

- **1 stop bit:** This is the default value of number of stop bits.
- **2 stop bits:** This is supported by normal USART, Single-wire and Modem modes.
- **1.5 stop bits:** To be used in Smartcard mode.
- **0.5 stop bit:** To be used when receiving data in Smartcard mode.

An idle frame transmission will include the stop bits.

A break transmission is 10 low bits (when M[1:0] = 00) or 11 low bits (when M[1:0] = 01) or 9 low bits (when M[1:0] = 10) followed by 2 stop bits (see [Figure 250](#)). It is not possible to transmit long breaks (break of length greater than 9/10/11 low bits).

Figure 250. Configurable stop bits



### Character transmission procedure

1. Program the M bits in USART\_CR1 to define the word length.
2. Select the desired baud rate using the USART\_BRR register.
3. Program the number of stop bits in USART\_CR2.
4. Enable the USART by writing the UE bit in USART\_CR1 register to 1.
5. Select DMA enable (DMAT) in USART\_CR3 if multibuffer communication is to take place. Configure the DMA register as explained in multibuffer communication.
6. Set the TE bit in USART\_CR1 to send an idle frame as first transmission.
7. Write the data to send in the USART\_TDR register (this clears the TXE bit). Repeat this for each data to be transmitted in case of single buffer.
8. After writing the last data into the USART\_TDR register, wait until TC=1. This indicates that the transmission of the last frame is complete. This is required for instance when the USART is disabled or enters the Halt mode to avoid corrupting the last transmission.

For code example, refer to [A.19.1: USART transmitter configuration code example](#).

### Single byte communication

Clearing the TXE bit is always performed by a write to the transmit data register.

The TXE bit is set by hardware and it indicates:

- The data has been moved from the USART\_TDR register to the shift register and the data transmission has started.
- The USART\_TDR register is empty.
- The next data can be written in the USART\_TDR register without overwriting the previous data.

For code example, refer to [A.19.2: USART transmit byte code example](#).

This flag generates an interrupt if the TXEIE bit is set.

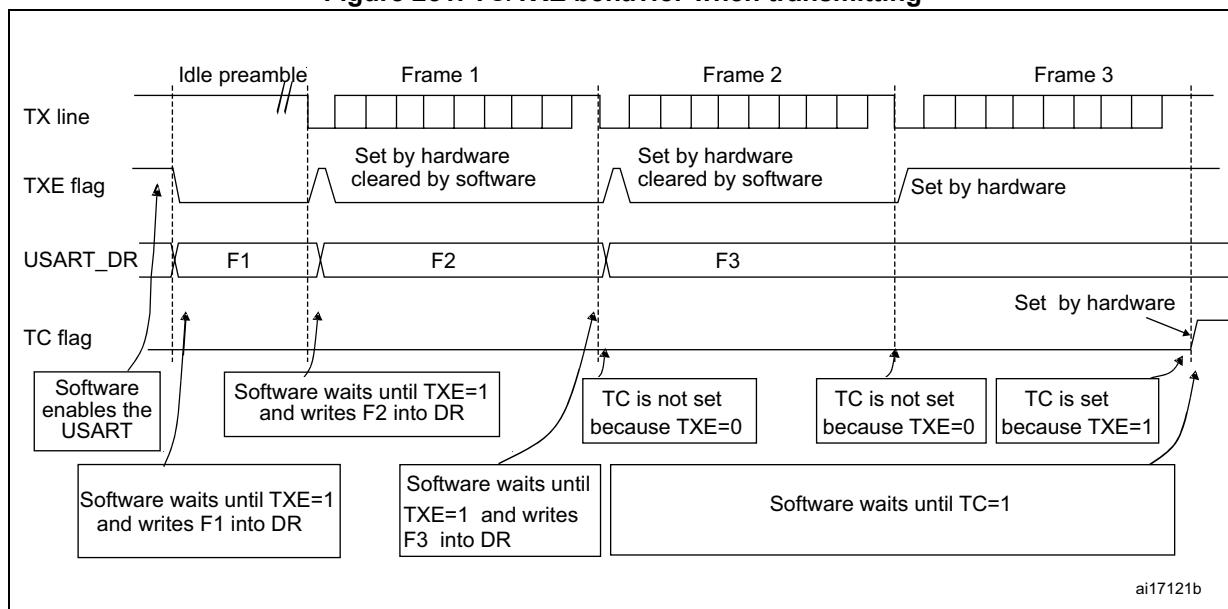
When a transmission is taking place, a write instruction to the USART\_TDR register stores the data in the TDR register; next, the data is copied in the shift register at the end of the currently ongoing transmission.

When no transmission is taking place, a write instruction to the USART\_TDR register places the data in the shift register, the data transmission starts, and the TXE bit is set.

If a frame is transmitted (after the stop bit) and the TXE bit is set, the TC bit goes high. An interrupt is generated if the TCIE bit is set in the USART\_CR1 register.

After writing the last data in the USART\_TDR register, it is mandatory to wait for TC=1 before disabling the USART or causing the microcontroller to enter the low-power mode (see [Figure 251: TC/TXE behavior when transmitting](#)).

**Figure 251. TC/TXE behavior when transmitting**



For code example, refer to [A.19.3: USART transfer complete code example](#).

### Break characters

Setting the SBKRQ bit transmits a break character. The break frame length depends on the M bits (see [Figure 249](#)).

If a '1' is written to the SBKRQ bit, a break character is sent on the TX line after completing the current character transmission. The SBKF bit is set by the write operation and it is reset by hardware when the break character is completed (during the stop bits after the break character). The USART inserts a logic 1 signal (STOP) for the duration of 2 bits at the end of the break frame to guarantee the recognition of the start bit of the next frame.

In the case the application needs to send the break character following all previously inserted data, including the ones not yet transmitted, the software should wait for the TXE flag assertion before setting the SBKRQ bit.

### Idle characters

Setting the TE bit drives the USART to send an idle frame before the first data frame.

### 27.5.3 USART receiver

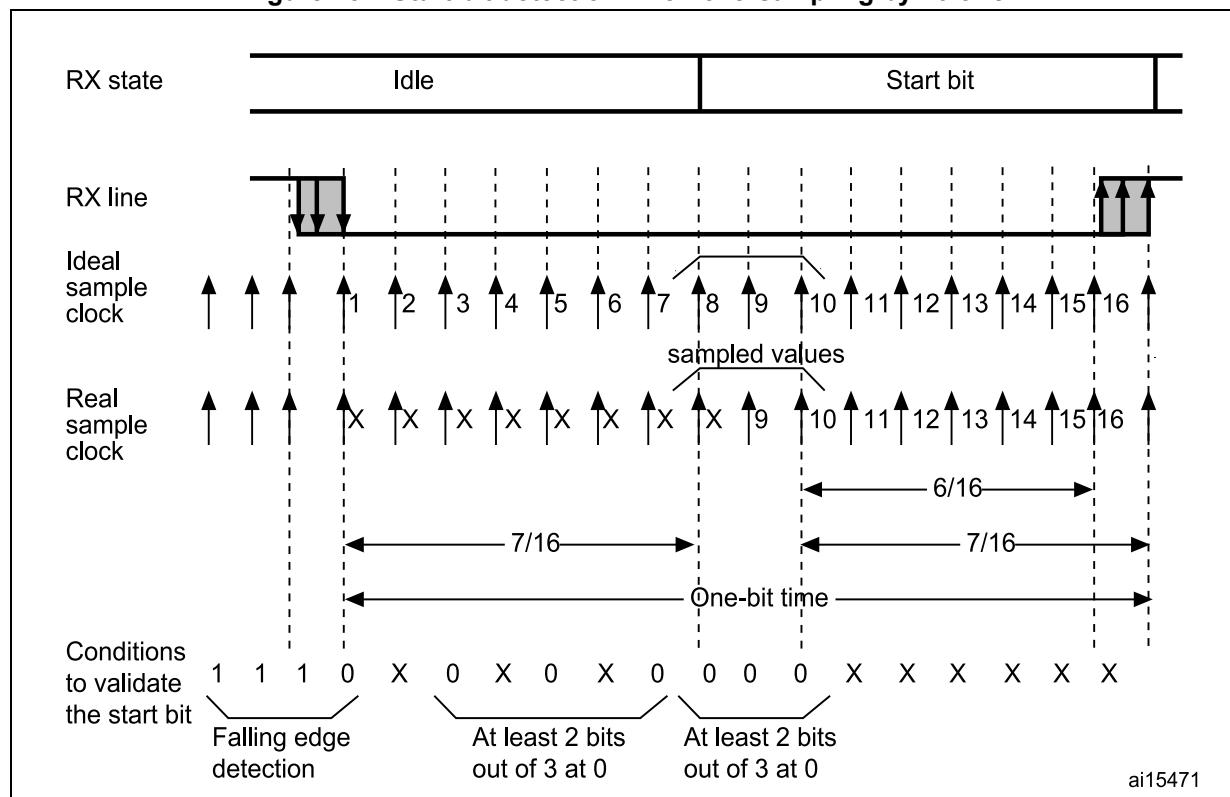
The USART can receive data words of either 7, 8 or 9 bits depending on the M bits in the USART\_CR1 register.

#### Start bit detection

The start bit detection sequence is the same when oversampling by 16 or by 8.

In the USART, the start bit is detected when a specific sequence of samples is recognized. This sequence is: 1 1 1 0 X 0 X 0X 0X 0 X 0X 0.

**Figure 252. Start bit detection when oversampling by 16 or 8**



**Note:** If the sequence is not complete, the start bit detection aborts and the receiver returns to the idle state (no flag is set), where it waits for a falling edge.

The start bit is confirmed (RXNE flag set, interrupt generated if RXNEIE=1) if the 3 sampled bits are at 0 (first sampling on the 3rd, 5th and 7th bits finds the 3 bits at 0 and second sampling on the 8th, 9th and 10th bits also finds the 3 bits at 0).

The start bit is validated (RXNE flag set, interrupt generated if RXNEIE=1) but the NF noise flag is set if,

- for both samplings, 2 out of the 3 sampled bits are at 0 (sampling on the 3rd, 5th and 7th bits and sampling on the 8th, 9th and 10th bits)
- or
- for one of the samplings (sampling on the 3rd, 5th and 7th bits or sampling on the 8th, 9th and 10th bits), 2 out of the 3 bits are found at 0.

If neither conditions a. or b. are met, the start detection aborts and the receiver returns to the idle state (no flag is set).

### Character reception

During an USART reception, data shifts in least significant bit first (default configuration) through the RX pin. In this mode, the USART\_RDR register consists of a buffer (RDR) between the internal bus and the receive shift register.

#### Character reception procedure

1. Program the M bits in USART\_CR1 to define the word length.
2. Select the desired baud rate using the baud rate register USART\_BRR
3. Program the number of stop bits in USART\_CR2.
4. Enable the USART by writing the UE bit in USART\_CR1 register to 1.
5. Select DMA enable (DMAR) in USART\_CR3 if multibuffer communication is to take place. Configure the DMA register as explained in multibuffer communication.
6. Set the RE bit USART\_CR1. This enables the receiver which begins searching for a start bit.

For code example, refer to [A.19.4: USART receiver configuration code example](#).

When a character is received:

- The RXNE bit is set to indicate that the content of the shift register is transferred to the RDR. In other words, data has been received and can be read (as well as its associated error flags).
- An interrupt is generated if the RXNEIE bit is set.
- The error flags can be set if a frame error, noise or an overrun error has been detected during reception. PE flag can also be set with RXNE.
- In multibuffer, RXNE is set after every byte received and is cleared by the DMA read of the Receive data Register.
- In single buffer mode, clearing the RXNE bit is performed by a software read to the USART\_RDR register. The RXNE flag can also be cleared by writing 1 to the RXFRQ in the USART\_RQR register. The RXNE bit must be cleared before the end of the reception of the next character to avoid an overrun error.

For code example, refer to [A.19.5: USART receive byte code example](#).

### Break character

When a break character is received, the USART handles it as a framing error.

### Idle character

When an idle frame is detected, there is the same procedure as for a received data character plus an interrupt if the IDLEIE bit is set.

## Overrun error

An overrun error occurs when a character is received when RXNE has not been reset. Data can not be transferred from the shift register to the RDR register until the RXNE bit is cleared.

The RXNE flag is set after every byte received. An overrun error occurs if RXNE flag is set when the next data is received or the previous DMA request has not been serviced. When an overrun error occurs:

- The ORE bit is set.
- The RDR content will not be lost. The previous data is available when a read to USART\_RDR is performed.
- The shift register are overwritten. After that point, any data received during overrun is lost.
- An interrupt is generated if either the RXNEIE bit is set or EIE bit is set.
- The ORE bit is reset by setting the ORECF bit in the ICR register.

*Note:*

*The ORE bit, when set, indicates that at least 1 datum has been lost. There are two possibilities:*

- if RXNE=1, then the last valid data is stored in the receive register RDR and can be read,
- if RXNE=0, then it means that the last valid data has already been read and thus there is nothing to be read in the RDR. This case can occur when the last valid data is read in the RDR at the same time as the new (and lost) data is received.

## Selecting the clock source and the proper oversampling method

The choice of the clock source is done through the Clock Control system (see Section Reset and clock control (RCC))). The clock source must be chosen before enabling the USART (by setting the UE bit).

The choice of the clock source must be done according to two criteria:

- Possible use of the USART in low-power mode
- Communication speed.

The clock source frequency is  $f_{CK}$ .

When the dual clock domain with the wake-up from Stop mode is supported, the clock source can be one of the following sources: PCLK (default), LSE, HSI or SYSCLK. Otherwise, the USART clock source is PCLK.

Choosing LSE or HSI as clock source may allow the USART to receive data while the MCU is in low-power mode. Depending on the received data and wake-up mode selection, the USART wakes up the MCU, when needed, in order to transfer the received data by software reading the USART\_RDR register or by DMA.

For the other clock sources, the system must be active in order to allow USART communication.

The communication speed range (specially the maximum communication speed) is also determined by the clock source.

The receiver implements different user-configurable oversampling techniques for data recovery by discriminating between valid incoming data and noise. This allows a trade-off between the maximum communication speed and noise/clock inaccuracy immunity.

The oversampling method can be selected by programming the OVER8 bit in the USART\_CR1 register and can be either 16 or 8 times the baud rate clock ([Figure 253](#) and [Figure 254](#)).

Depending on the application:

- Select oversampling by 8 (OVER8=1) to achieve higher speed (up to  $f_{CK}/8$ ). In this case the maximum receiver tolerance to clock deviation is reduced (refer to [Section 27.5.5: Tolerance of the USART receiver to clock deviation on page 718](#))
- Select oversampling by 16 (OVER8=0) to increase the tolerance of the receiver to clock deviations. In this case, the maximum speed is limited to maximum  $f_{CK}/16$  where  $f_{CK}$  is the clock source frequency.

Programming the ONEBIT bit in the USART\_CR3 register selects the method used to evaluate the logic level. There are two options:

- The majority vote of the three samples in the center of the received bit. In this case, when the 3 samples used for the majority vote are not equal, the NF bit is set
- A single sample in the center of the received bit

Depending on the application:

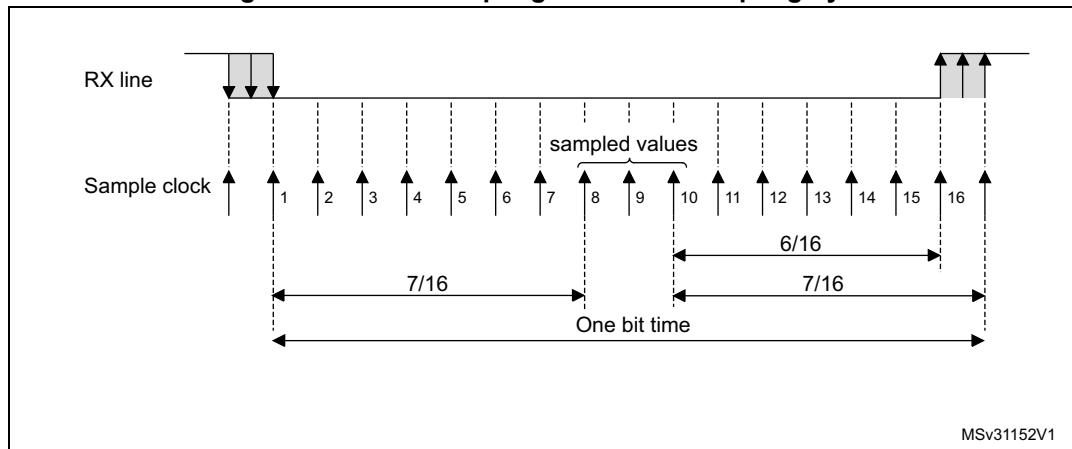
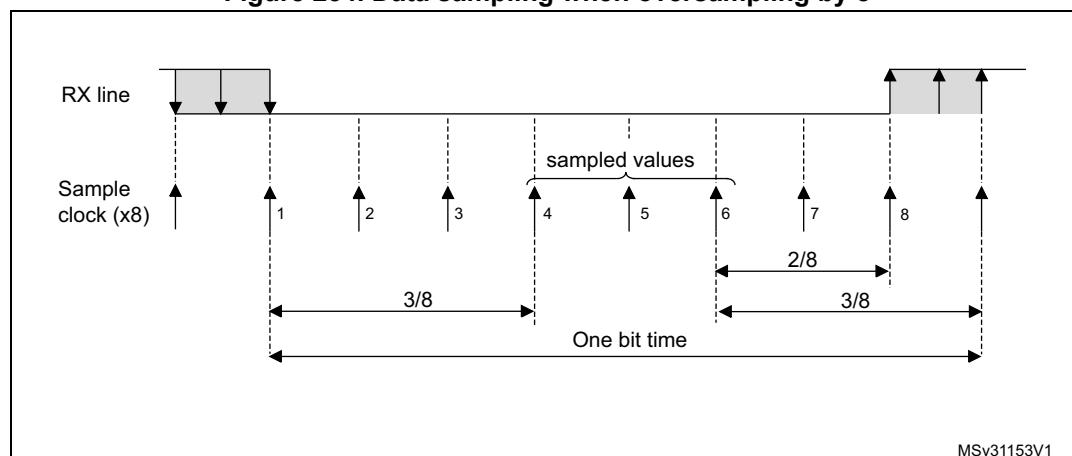
- select the three samples' majority vote method (ONEBIT=0) when operating in a noisy environment and reject the data when a noise is detected (refer to [Figure 105](#)) because this indicates that a glitch occurred during the sampling.
- select the single sample method (ONEBIT=1) when the line is noise-free to increase the receiver's tolerance to clock deviations (see [Section 27.5.5: Tolerance of the USART receiver to clock deviation on page 718](#)). In this case the NF bit will never be set.

When noise is detected in a frame:

- The NF bit is set at the rising edge of the RXNE bit.
- The invalid data is transferred from the Shift register to the USART\_RDR register.
- No interrupt is generated in case of single byte communication. However this bit rises at the same time as the RXNE bit which itself generates an interrupt. In case of multibuffer communication an interrupt is issued if the EIE bit is set in the USART\_CR3 register.

The NF bit is reset by setting NFCF bit in ICR register.

**Note:** *Oversampling by 8 is not available in LIN, Smartcard and IrDA modes. In those modes, the OVER8 bit is forced to '0' by hardware.*

**Figure 253. Data sampling when oversampling by 16****Figure 254. Data sampling when oversampling by 8****Table 105. Noise detection from sampled data**

Sampled value	NE status	Received bit value
000	0	0
001	1	0
010	1	0
011	1	1
100	1	0
101	1	1
110	1	1
111	0	1

## Framing error

A framing error is detected when the stop bit is not recognized on reception at the expected time, following either a de-synchronization or excessive noise.

When the framing error is detected:

- The FE bit is set by hardware
- The invalid data is transferred from the Shift register to the USART\_RDR register.
- No interrupt is generated in case of single byte communication. However this bit rises at the same time as the RXNE bit which itself generates an interrupt. In case of multibuffer communication an interrupt is issued if the EIE bit is set in the USART\_CR3 register.

The FE bit is reset by writing 1 to the FECF in the USART\_ICR register.

## Configurable stop bits during reception

The number of stop bits to be received can be configured through the control bits of Control Register 2 - it can be either 1 or 2 in normal mode and 0.5 or 1.5 in Smartcard mode.

- **0.5 stop bit (reception in Smartcard mode):** *No sampling is done for 0.5 stop bit. As a consequence, no framing error and no break frame can be detected when 0.5 stop bit is selected.*
- **1 stop bit:** Sampling for 1 stop Bit is done on the 8th, 9th and 10th samples.
- **1.5 stop bits (Smartcard mode):** When transmitting in Smartcard mode, the device must check that the data is correctly sent. Thus the receiver block must be enabled (RE =1 in the USART\_CR1 register) and the stop bit is checked to test if the smartcard has detected a parity error. In the event of a parity error, the smartcard forces the data signal low during the sampling - NACK signal-, which is flagged as a framing error. Then, the FE flag is set with the RXNE at the end of the 1.5 stop bits. Sampling for 1.5 stop bits is done on the 16th, 17th and 18th samples (1 baud clock period after the beginning of the stop bit). The 1.5 stop bits can be decomposed into 2 parts: one 0.5 baud clock period during which nothing happens, followed by 1 normal stop bit period during which sampling occurs halfway through. Refer to [Section 27.5.13: USART Smartcard mode on page 729](#) for more details.
- **2 stop bits:** Sampling for 2 stop bits is done on the 8th, 9th and 10th samples of the first stop bit. If a framing error is detected during the first stop bit the framing error flag is set. The second stop bit is not checked for framing error. The RXNE flag is set at the end of the first stop bit.

### 27.5.4 USART baud rate generation

The baud rate for the receiver and transmitter (Rx and Tx) are both set to the same value as programmed in the USART\_BRR register.

#### Equation 1: Baud rate for standard USART (SPI mode included) (OVER8 = 0 or 1)

In case of oversampling by 16, the equation is:

$$\text{Tx/Rx baud} = \frac{f_{CK}}{\text{USARTDIV}}$$

In case of oversampling by 8, the equation is:

$$\text{Tx/Rx baud} = \frac{2 \times f_{CK}}{\text{USARTDIV}}$$

#### Equation 2: Baud rate in Smartcard, LIN and IrDA modes (OVER8 = 0)

In Smartcard, LIN and IrDA modes, only Oversampling by 16 is supported:

$$\text{Tx/Rx baud} = \frac{f_{CK}}{\text{USARTDIV}}$$

USARTDIV is an unsigned fixed point number that is coded on the USART\_BRR register.

- When OVER8 = 0, BRR = USARTDIV.
- When OVER8 = 1
  - BRR[2:0] = USARTDIV[3:0] shifted 1 bit to the right.
  - BRR[3] must be kept cleared.
  - BRR[15:4] = USARTDIV[15:4]

*Note:* The baud counters are updated to the new value in the baud registers after a write operation to USART\_BRR. Hence the baud rate register value should not be changed during communication.

*In case of oversampling by 16 or 8, USARTDIV must be greater than or equal to 16d.*

#### How to derive USARTDIV from USART\_BRR register values

##### Example 1

To obtain 9600 baud with  $f_{CK}$  = 8 MHz.

- In case of oversampling by 16:  
 $\text{USARTDIV} = 8\ 000\ 000/9600$   
 $\text{BRR} = \text{USARTDIV} = 833d = 0341h$
- In case of oversampling by 8:  
 $\text{USARTDIV} = 2 * 8\ 000\ 000/9600$   
 $\text{USARTDIV} = 1666,66$  ( $1667d = 683h$ )  
 $\text{BRR}[3:0] = 3h\ 1 = 1h$   
 $\text{BRR} = 0x681$

**Example 2**

To obtain 921.6 kbaud with  $f_{CK} = 48$  MHz.

- In case of oversampling by 16:

$$\text{USARTDIV} = 48\ 000\ 000 / 921\ 600$$

$$\text{BRR} = \text{USARTDIV} = 52d = 34h$$

- In case of oversampling by 8:

$$\text{USARTDIV} = 2 * 48\ 000\ 000 / 921\ 600$$

$$\text{USARTDIV} = 104 \ (104d = 68h)$$

$$\text{BRR}[3:0] = \text{USARTDIV}[3:0] >> 1 = 8h >> 1 = 4h$$

$$\text{BRR} = 0x64$$

**Table 106. Error calculation for programmed baud rates at  $f_{CK} = 48$  MHz in both cases of oversampling by 16 or by 8<sup>(1)</sup>**

Baud rate		Oversampling by 16 (OVER8 = 0)			Oversampling by 8 (OVER8 = 1)		
S.No	Desired	Actual	BRR	% Error = (Calculated - Desired)B.Rate / Desired B.Rate	Actual	BRR	% Error
2	2.4 KBps	2.4 KBps	0x4E20	0	2.4 KBps	0x9C40	0
3	9.6 KBps	9.6 KBps	0x1388	0	9.6 KBps	0x2710	0
4	19.2 KBps	19.2 KBps	0x9C4	0	19.2 KBps	0x1384	0
5	38.4 KBps	38.4 KBps	0x4E2	0	38.4 KBps	0x9C2	0
6	57.6 KBps	57.62 KBps	0x341	0.03	57.59 KBps	0x681	0.02
7	115.2 KBps	115.11 KBps	0x1A1	0.08	115.25 KBps	0x340	0.04
8	230.4 KBps	230.76KBps	0xD0	0.16	230.21 KBps	0x1A0	0.08
9	460.8 KBps	461.54KBps	0x68	0.16	461.54KBps	0xD0	0.16
10	921.6KBps	923.07KBps	0x34	0.16	923.07KBps	0x64	0.16
11	2 MBps	2 MBps	0x18	0	2 MBps	0x30	0
12	3 MBps	3 MBps	0x10	0	3 MBps	0x20	0
13	4MBps	N.A	N.A	N.A	4MBps	0x14	0
14	5MBps	N.A	N.A	N.A	5052.63KBps	0x11	1.05
15	6MBps	N.A	N.A	N.A	6MBps	0x10	0

1. The lower the CPU clock the lower the accuracy for a particular baud rate. The upper limit of the achievable baud rate can be fixed with these data.

### 27.5.5 Tolerance of the USART receiver to clock deviation

The asynchronous receiver of the USART works correctly only if the total clock system deviation is less than the tolerance of the USART receiver. The causes which contribute to the total deviation are:

- DTRA: Deviation due to the transmitter error (which also includes the deviation of the transmitter's local oscillator)
- DQUANT: Error due to the baud rate quantization of the receiver
- DREC: Deviation of the receiver's local oscillator
- DTCL: Deviation due to the transmission line (generally due to the transceivers which can introduce an asymmetry between the low-to-high transition timing and the high-to-low transition timing)

$$\text{DTRA} + \text{DQUANT} + \text{DREC} + \text{DTCL} + \text{DWU} < \text{USART receiver's tolerance}$$

where

DWU is the error due to sampling point deviation when the wake-up from Stop mode is used.

when M[1:0] = 01:

$$\text{DWU} = \frac{t_{\text{WUUSART}}}{11 \times \text{Tbit}}$$

when M[1:0] = 00:

$$\text{DWU} = \frac{t_{\text{WUUSART}}}{10 \times \text{Tbit}}$$

when M[1:0] = 10:

$$\text{DWU} = \frac{t_{\text{WUUSART}}}{9 \times \text{Tbit}}$$

$t_{\text{WUUSART}}$  is the time between:

- The detection of start bit falling edge
- The instant when clock (requested by the peripheral) is ready and reaching the peripheral and regulator is ready.

In STM32F0xx,  $t_{\text{WUUSART}}$  corresponds to  $t_{\text{WUSTOP}}$  value provided in the datasheet.

The USART receiver can receive data correctly at up to the maximum tolerated deviation specified in [Table 107](#) and [Table 108](#) depending on the following choices:

- 9-, 10- or 11-bit character length defined by the M bits in the USART\_CR1 register
- Oversampling by 8 or 16 defined by the OVER8 bit in the USART\_CR1 register
- Bits BRR[3:0] of USART\_BRR register are equal to or different from 0000.
- Use of 1 bit or 3 bits to sample the data, depending on the value of the ONEBIT bit in the USART\_CR3 register.

**Table 107. Tolerance of the USART receiver when BRR [3:0] = 0000**

M bits	OVER8 bit = 0		OVER8 bit = 1	
	ONEBIT=0	ONEBIT=1	ONEBIT=0	ONEBIT=1
00	3.75%	4.375%	2.50%	3.75%
01	3.41%	3.97%	2.27%	3.41%
10	4.16%	4.86%	2.77%	4.16%

**Table 108. Tolerance of the USART receiver when BRR [3:0] is different from 0000**

M bits	OVER8 bit = 0		OVER8 bit = 1	
	ONEBIT=0	ONEBIT=1	ONEBIT=0	ONEBIT=1
00	3.33%	3.88%	2%	3%
01	3.03%	3.53%	1.82%	2.73%
10	3.7%	4.31%	2.22%	3.33%

**Note:** The data specified in [Table 107](#) and [Table 108](#) may slightly differ in the special case when the received frames contain some Idle frames of exactly 10-bit durations when M bits = 00 (11-bit durations when M bits =01 or 9- bit durations when M bits = 10).

### 27.5.6 USART auto baud rate detection

The USART is able to detect and automatically set the USART\_BRR register value based on the reception of one character. Automatic baud rate detection is useful under two circumstances:

- The communication speed of the system is not known in advance
- The system is using a relatively low accuracy clock source and this mechanism allows the correct baud rate to be obtained without measuring the clock deviation.

The clock source frequency must be compatible with the expected communication speed (in case only modes 0 and 1 are supported, oversampling by 16 must be selected and baud rate between  $f_{CK}/65535$  and  $f_{CK}/16$ . Otherwise, in case of 4 modes are supported: when oversampling by 16, the baud rate is between  $f_{CK}/65535$  and  $f_{CK}/16$ ; when oversampling by 8, the baud rate is between  $f_{CK}/65535$  and  $f_{CK}/8$ ).

Before activating the auto baud rate detection, the auto baud rate detection mode must be chosen. There are various modes based on different character patterns.

They can be chosen through the ABRMOD[1:0] field in the USART\_CR2 register. In these auto baud rate modes, the baud rate is measured several times during the synchronization data reception and each measurement is compared to the previous one.

These modes are:

- **Mode 0:** Any character starting with a bit at 1. In this case the USART measures the duration of the Start bit (falling edge to rising edge).
- **Mode 1:** Any character starting with a 10xx bit pattern. In this case, the USART measures the duration of the Start and of the 1st data bit. The measurement is done falling edge to falling edge, ensuring better accuracy in the case of slow signal slopes.
- **Mode 2:** A 0x7F character frame (it may be a 0x7F character in LSB first mode or a 0xFE in MSB first mode). In this case, the baud rate is updated first at the end of the start bit (BRs), then at the end of bit 6 (based on the measurement done from falling edge to falling edge: BR6). Bit 0 to bit 6 are sampled at BRs while further bits of the character are sampled at BR6.
- **Mode 3:** A 0x55 character frame. In this case, the baud rate is updated first at the end of the start bit (BRs), then at the end of bit 0 (based on the measurement done from falling edge to falling edge: BR0), and finally at the end of bit 6 (BR6). Bit 0 is sampled at BRs, bit 1 to bit 6 are sampled at BR0, and further bits of the character are sampled at BR6.

In parallel, another check is performed for each intermediate transition of RX line. An error is generated if the transitions on RX are not sufficiently synchronized with the receiver (the receiver being based on the baud rate calculated on bit 0).

Prior to activating auto baud rate detection, the USART\_BRR register must be initialized by writing a non-zero baud rate value.

The automatic baud rate detection is activated by setting the ABREN bit in the USART\_CR2 register. The USART will then wait for the first character on the RX line. The auto baud rate operation completion is indicated by the setting of the ABRF flag in the USART\_ISR register. If the line is noisy, the correct baud rate detection cannot be guaranteed. In this case the BRR value may be corrupted and the ABRE error flag is set. This also happens if the communication speed is not compatible with the automatic baud rate detection range (bit duration not between 16 and 65536 clock periods (oversampling by 16) and not between 8 and 65536 clock periods (oversampling by 8)).

The RXNE interrupt will signal the end of the operation.

At any later time, the auto baud rate detection may be relaunched by resetting the ABRF flag (by writing a 0).

*Note: If the USART is disabled (UE=0) during an auto baud rate operation, the BRR value may be corrupted.*

### 27.5.7 Multiprocessor communication using USART

In multiprocessor communication, the following bits are to be kept cleared:

- LINEN bit in the USART\_CR2 register,
- HDSEL, IREN and SCEN bits in the USART\_CR3 register.

It is possible to perform multiprocessor communication with the USART (with several USARTs connected in a network). For instance one of the USARTs can be the master, its TX output connected to the RX inputs of the other USARTs. The others are slaves, their respective TX outputs are logically ANDed together and connected to the RX input of the master.

In multiprocessor configurations it is often desirable that only the intended message recipient should actively receive the full message contents, thus reducing redundant USART service overhead for all non addressed receivers.

The non addressed devices may be placed in mute mode by means of the muting function. In order to use the mute mode feature, the MME bit must be set in the USART\_CR1 register.

In mute mode:

- None of the reception status bits can be set.
- All the receive interrupts are inhibited.
- The RWU bit in USART\_ISR register is set to 1. RWU can be controlled automatically by hardware or by software, through the MMRQ bit in the USART\_RQR register, under certain conditions.

The USART can enter or exit from mute mode using one of two methods, depending on the WAKE bit in the USART\_CR1 register:

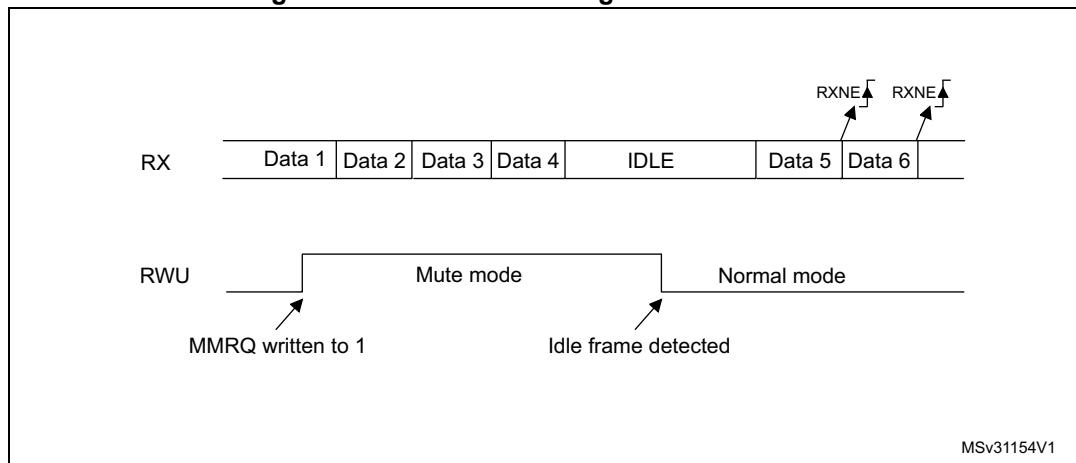
- Idle Line detection if the WAKE bit is reset,
- Address Mark detection if the WAKE bit is set.

### Idle line detection (WAKE=0)

The USART enters mute mode when the MMRQ bit is written to 1 and the RWU is automatically set.

It wakes up when an Idle frame is detected. Then the RWU bit is cleared by hardware but the IDLE bit is not set in the USART\_ISR register. An example of mute mode behavior using Idle line detection is given in [Figure 255](#).

**Figure 255. Mute mode using Idle line detection**



Note:

If the MMRQ is set while the IDLE character has already elapsed, mute mode will not be entered (RWU is not set).

If the USART is activated while the line is IDLE, the idle state is detected after the duration of one IDLE frame (not only after the reception of one character frame).

### 4-bit/7-bit address mark detection (WAKE=1)

In this mode, bytes are recognized as addresses if their MSB is a '1' otherwise they are considered as data. In an address byte, the address of the targeted receiver is put in the 4

or 7 LSBs. The choice of 7 or 4-bit address detection is done using the ADDM7 bit. This 4-bit/7-bit word is compared by the receiver with its own address which is programmed in the ADD bits in the USART\_CR2 register.

**Note:** *In 7-bit and 9-bit data modes, address detection is done on 6-bit and 8-bit addresses (ADD[5:0] and ADD[7:0]) respectively.*

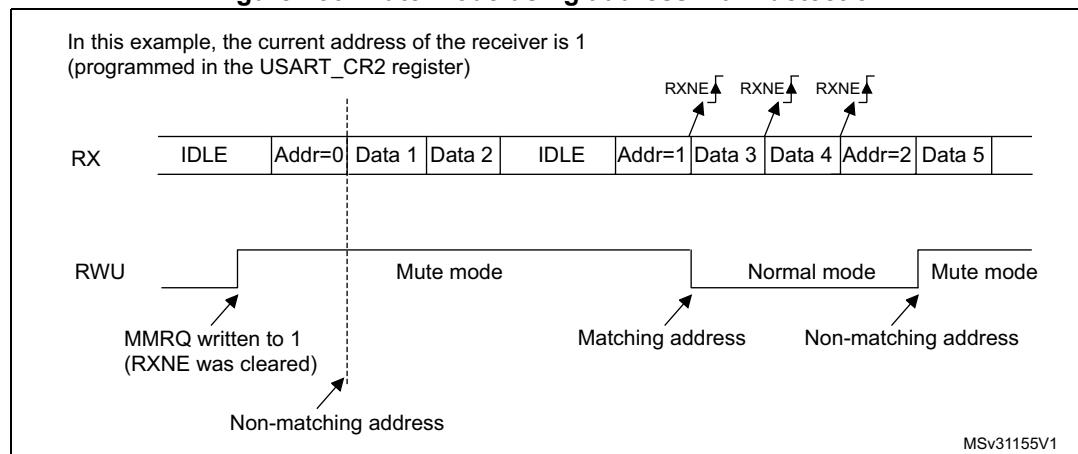
The USART enters mute mode when an address character is received which does not match its programmed address. In this case, the RWU bit is set by hardware. The RXNE flag is not set for this address byte and no interrupt or DMA request is issued when the USART enters mute mode.

The USART also enters mute mode when the MMRQ bit is written to 1. The RWU bit is also automatically set in this case.

The USART exits from mute mode when an address character is received which matches the programmed address. Then the RWU bit is cleared and subsequent bytes are received normally. The RXNE bit is set for the address character since the RWU bit has been cleared.

An example of mute mode behavior using address mark detection is given in [Figure 256](#).

**Figure 256. Mute mode using address mark detection**



## 27.5.8 Modbus communication using USART

The USART offers basic support for the implementation of Modbus/RTU and Modbus/ASCII protocols. Modbus/RTU is a half duplex, block transfer protocol. The control part of the protocol (address recognition, block integrity control and command interpretation) must be implemented in software.

The USART offers basic support for the end of the block detection, without software overhead or other resources.

### Modbus/RTU

In this mode, the end of one block is recognized by a “silence” (idle line) for more than 2 character times. This function is implemented through the programmable timeout function.

The timeout function and interrupt must be activated, through the RTOEN bit in the USART\_CR2 register and the RTOIE in the USART\_CR1 register. The value corresponding to a timeout of 2 character times (for example 22 x bit duration) must be programmed in the

RTO register. when the receive line is idle for this duration, after the last stop bit is received, an interrupt is generated, informing the software that the current block reception is completed.

### Modbus/ASCII

In this mode, the end of a block is recognized by a specific (CR/LF) character sequence. The USART manages this mechanism using the character match function.

By programming the LF ASCII code in the ADD[7:0] field and by activating the character match interrupt (CMIE=1), the software is informed when a LF has been received and can check the CR/LF in the DMA buffer.

## 27.5.9 USART parity control

Parity control (generation of parity bit in transmission and parity checking in reception) can be enabled by setting the PCE bit in the USART\_CR1 register. Depending on the frame length defined by the M bits, the possible USART frame formats are as listed in [Table 109](#).

**Table 109. Frame formats**

M bits	PCE bit	USART frame <sup>(1)</sup>
00	0	SB   8-bit data   STB
00	1	SB   7-bit data   PB   STB
01	0	SB   9-bit data   STB
01	1	SB   8-bit data   PB   STB
10	0	SB   7-bit data   STB
10	1	SB   6-bit data   PB   STB

- Legends: SB: start bit, STB: stop bit, PB: parity bit. In the data register, the PB is always taking the MSB position (9th, 8th or 7th, depending on the M bits value).

### Even parity

The parity bit is calculated to obtain an even number of “1s” inside the frame of the 6, 7 or 8 LSB bits (depending on M bits values) and the parity bit.

As an example, if data=00110101, and 4 bits are set, then the parity bit is 0 if even parity is selected (PS bit in USART\_CR1 = 0).

### Odd parity

The parity bit is calculated to obtain an odd number of “1s” inside the frame made of the 6, 7 or 8 LSB bits (depending on M bits values) and the parity bit.

As an example, if data=00110101 and 4 bits set, then the parity bit is 1 if odd parity is selected (PS bit in USART\_CR1 = 1).

### Parity checking in reception

If the parity check fails, the PE flag is set in the USART\_ISR register and an interrupt is generated if PEIE is set in the USART\_CR1 register. The PE flag is cleared by software writing 1 to the PECF in the USART\_ICR register.

### Parity generation in transmission

If the PCE bit is set in USART\_CR1, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit (even number of "1s" if even parity is selected (PS=0) or an odd number of "1s" if odd parity is selected (PS=1)).

#### 27.5.10 USART LIN (local interconnection network) mode

This section is relevant only when LIN mode is supported. Please refer to [Section 27.4: USART implementation on page 702](#).

The LIN mode is selected by setting the LINEN bit in the USART\_CR2 register. In LIN mode, the following bits must be kept cleared:

- STOP[1:0] and CLKEN in the USART\_CR2 register,
- SCEN, HDSEL and IREN in the USART\_CR3 register.

For code example, refer to [A.19.6: USART LIN mode code example](#).

#### LIN transmission

The procedure explained in [Section 27.5.2: USART transmitter](#) has to be applied for LIN Master transmission. It must be the same as for normal USART transmission with the following differences:

- Clear the M bits to configure 8-bit word length.
- Set the LINEN bit to enter LIN mode. In this case, setting the SBKRQ bit sends 13 '0' bits as a break character. Then 2 bits of value '1' are sent to allow the next start detection.

#### LIN reception

When LIN mode is enabled, the break detection circuit is activated. The detection is totally independent from the normal USART receiver. A break can be detected whenever it occurs, during Idle state or during a frame.

When the receiver is enabled (RE=1 in USART\_CR1), the circuit looks at the RX input for a start signal. The method for detecting start bits is the same when searching break characters or data. After a start bit has been detected, the circuit samples the next bits exactly like for the data (on the 8th, 9th and 10th samples). If 10 (when the LBDL = 0 in USART\_CR2) or 11 (when LBDL=1 in USART\_CR2) consecutive bits are detected as '0', and are followed by a delimiter character, the LBDF flag is set in USART\_ISR. If the LBDIE bit=1, an interrupt is generated. Before validating the break, the delimiter is checked for as it signifies that the RX line has returned to a high level.

If a '1' is sampled before the 10 or 11 have occurred, the break detection circuit cancels the current detection and searches for a start bit again.

If the LIN mode is disabled (LINEN=0), the receiver continues working as normal USART, without taking into account the break detection.

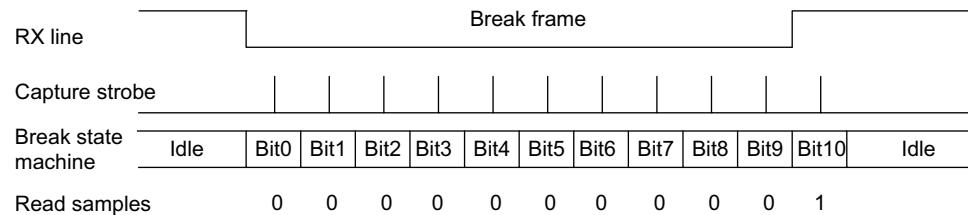
If the LIN mode is enabled (LINEN=1), as soon as a framing error occurs (i.e. stop bit detected at '0', which is the case for any break frame), the receiver stops until the break detection circuit receives either a '1', if the break word was not complete, or a delimiter character if a break has been detected.

The behavior of the break detector state machine and the break flag is shown on the [Figure 257: Break detection in LIN mode \(11-bit break length - LBDL bit is set\) on page 725](#).

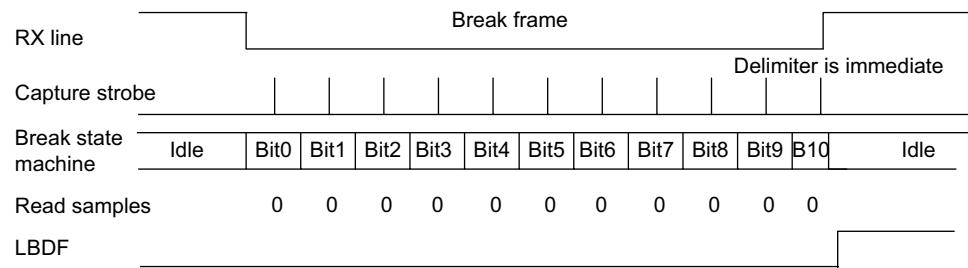
Examples of break frames are given on [Figure 258: Break detection in LIN mode vs. Framing error detection on page 726](#).

**Figure 257. Break detection in LIN mode (11-bit break length - LBDL bit is set)**

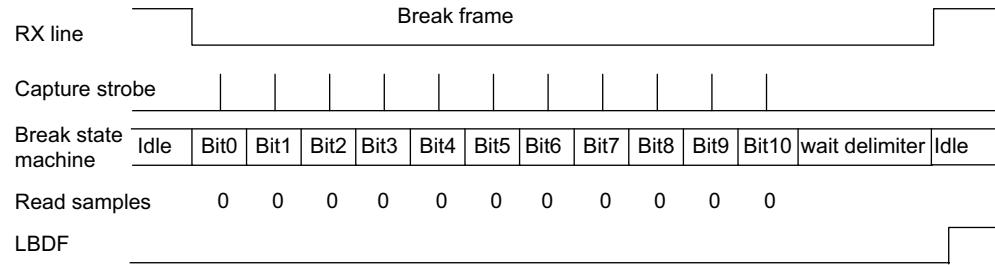
**Case 1: break signal not long enough => break discarded, LBDF is not set**



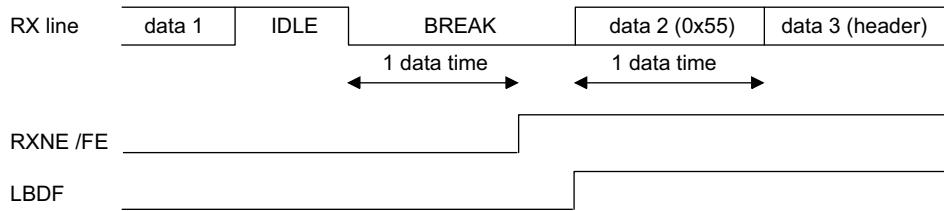
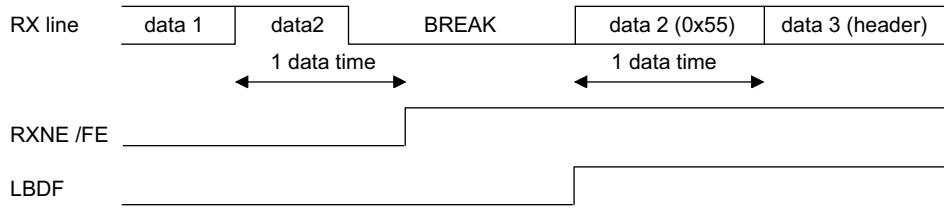
**Case 2: break signal just long enough => break detected, LBDF is set**



**Case 3: break signal long enough => break detected, LBDF is set**



MSv31156V1

**Figure 258. Break detection in LIN mode vs. Framing error detection****Case 1: break occurring after an Idle****Case 2: break occurring while data is being received**

MSv31157V1

### 27.5.11 USART synchronous mode

The synchronous mode is selected by writing the CLKEN bit in the USART\_CR2 register to 1. In synchronous mode, the following bits must be kept cleared:

- LINEN bit in the USART\_CR2 register,
- SCEN, HDSEL and IREN bits in the USART\_CR3 register.

In this mode, the USART can be used to control bidirectional synchronous serial communications in master mode. The CK pin is the output of the USART transmitter clock. No clock pulses are sent to the CK pin during start bit and stop bit. Depending on the state of the LBCL bit in the USART\_CR2 register, clock pulses are, or are not, generated during the last valid data bit (address mark). The CPOL bit in the USART\_CR2 register is used to select the clock polarity, and the CPHA bit in the USART\_CR2 register is used to select the phase of the external clock (see [Figure 259](#), [Figure 260](#) and [Figure 261](#)).

During the Idle state, preamble and send break, the external CK clock is not activated.

In synchronous mode the USART transmitter works exactly like in asynchronous mode. But as CK is synchronized with TX (according to CPOL and CPHA), the data on TX is synchronous.

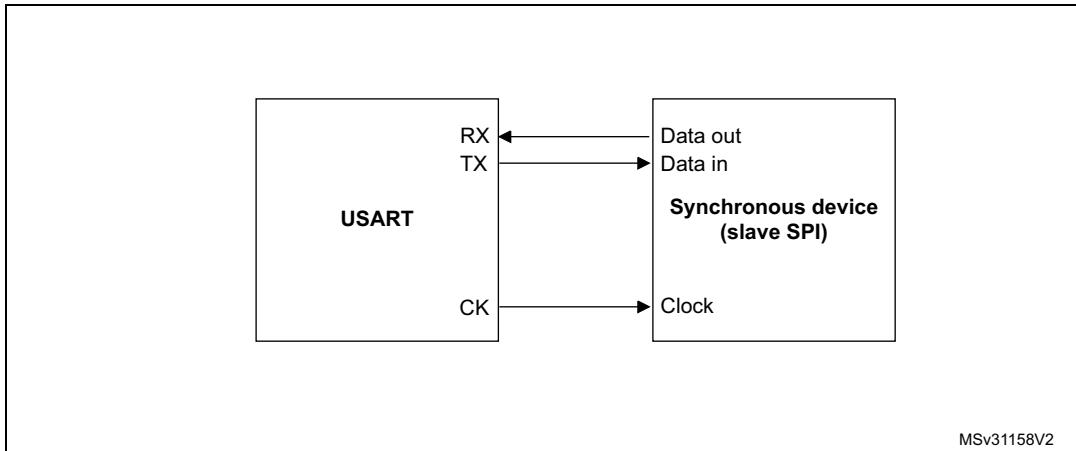
In this mode the USART receiver works in a different manner compared to the asynchronous mode. If RE=1, the data is sampled on CK (rising or falling edge, depending on CPOL and CPHA), without any oversampling. A setup and a hold time must be respected (which depends on the baud rate: 1/16 bit duration).

**Note:** The CK pin works in conjunction with the TX pin. Thus, the clock is provided only if the transmitter is enabled ( $TE=1$ ) and data is being transmitted (the data register USART\_TDR written). This means that it is not possible to receive synchronous data without transmitting data.

The LBCL, CPOL and CPHA bits have to be selected when the USART is disabled ( $UE=0$ ) to ensure that the clock pulses function correctly.

For code example, refer to [A.19.7: USART synchronous mode code example](#).

**Figure 259. USART example of synchronous transmission**



**Figure 260. USART data clock timing diagram (M bits = 00)**

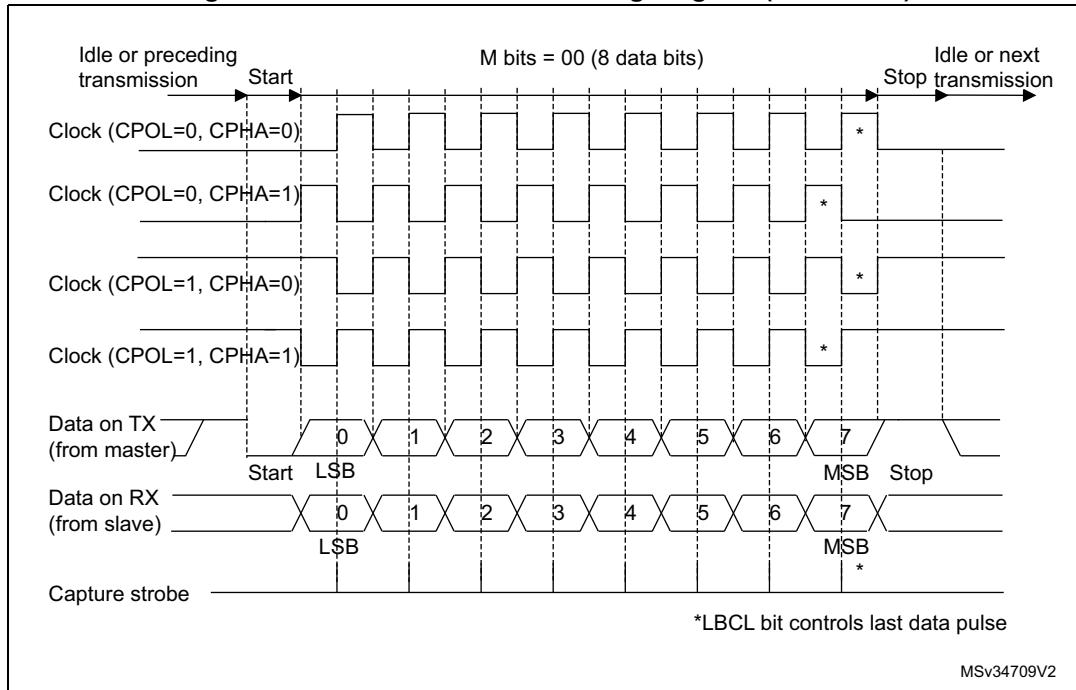


Figure 261. USART data clock timing diagram (M bits = 01)

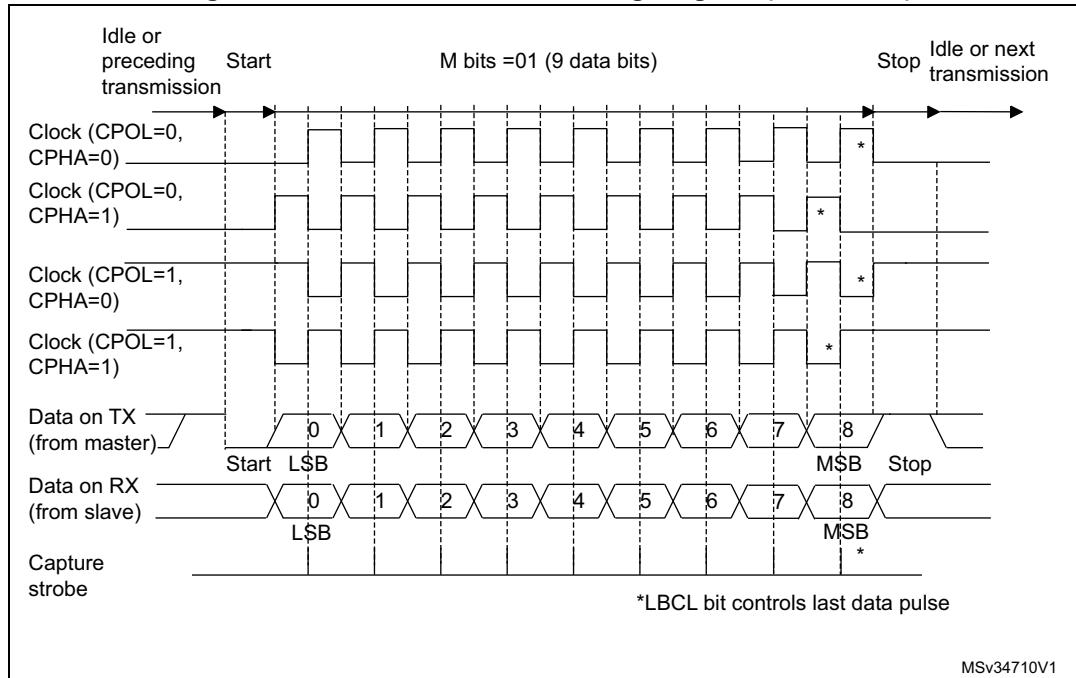
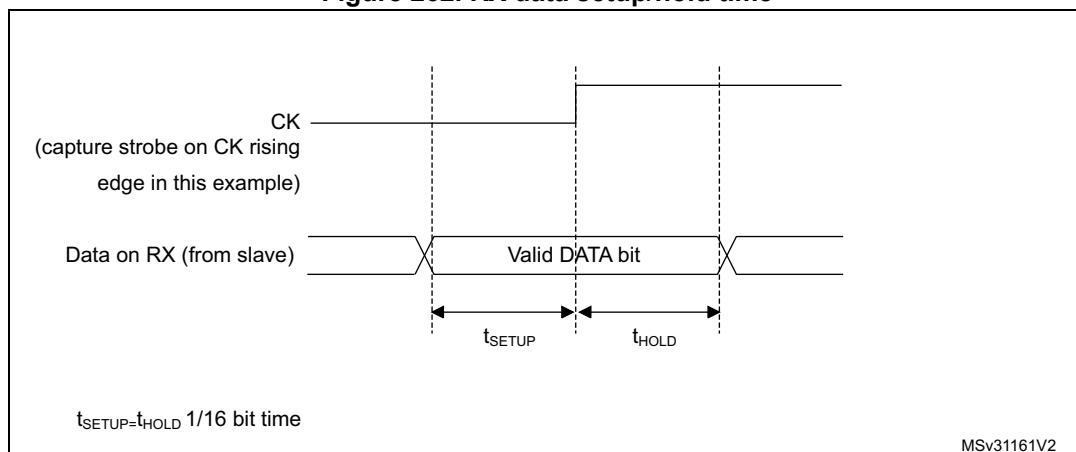


Figure 262. RX data setup/hold time



Note:

The function of CK is different in Smartcard mode. Refer to [Section 27.5.13: USART Smartcard mode](#) for more details.

### 27.5.12 USART Single-wire Half-duplex communication

Single-wire Half-duplex mode is selected by setting the HDSEL bit in the USART\_CR3 register. In this mode, the following bits must be kept cleared:

- LINEN and CLKEN bits in the USART\_CR2 register,
- SCEN and IREN bits in the USART\_CR3 register.

The USART can be configured to follow a Single-wire Half-duplex protocol where the TX and RX lines are internally connected. The selection between half- and Full-duplex communication is made with a control bit HDSEL in USART\_CR3.

As soon as HDSEL is written to 1:

- The TX and RX lines are internally connected
- The RX pin is no longer used
- The TX pin is always released when no data is transmitted. Thus, it acts as a standard I/O in idle or in reception. It means that the I/O must be configured so that TX is configured as alternate function open-drain with an external pull-up.

Apart from this, the communication protocol is similar to normal USART mode. Any conflicts on the line must be managed by software (by the use of a centralized arbiter, for instance). In particular, the transmission is never blocked by hardware and continues as soon as data is written in the data register while the TE bit is set.

For code example, refer to [A.19.11: USART DMA code example](#).

### 27.5.13 USART Smartcard mode

This section is relevant only when Smartcard mode is supported. Please refer to [Section 27.4: USART implementation on page 702](#).

Smartcard mode is selected by setting the SCEN bit in the USART\_CR3 register. In Smartcard mode, the following bits must be kept cleared:

- LINEN bit in the USART\_CR2 register,
- HDSEL and IREN bits in the USART\_CR3 register.

Moreover, the CLKEN bit may be set in order to provide a clock to the smartcard.

The smartcard interface is designed to support asynchronous protocol for smartcards as defined in the ISO 7816-3 standard. Both T=0 (character mode) and T=1 (block mode) are supported.

The USART should be configured as:

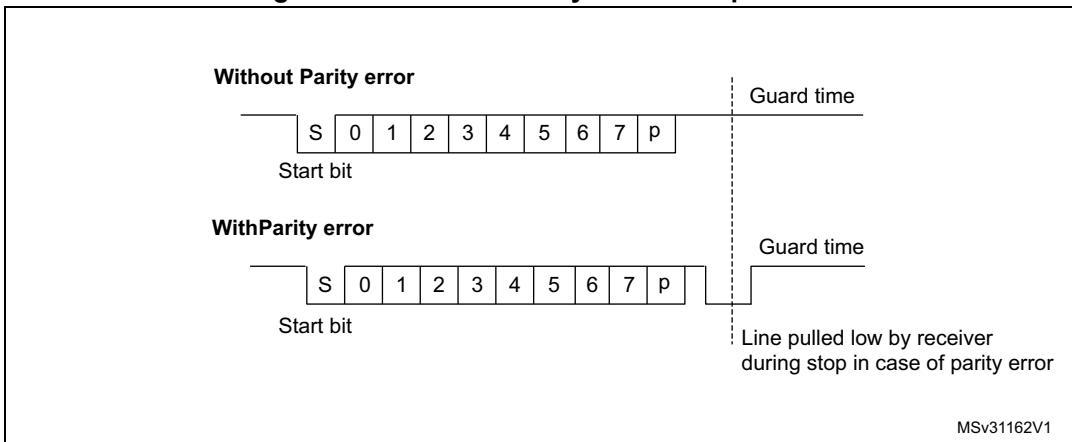
- 8 bits plus parity: where word length is set to 8 bits and PCE=1 in the USART\_CR1 register
- 1.5 stop bits: where STOP=11 in the USART\_CR2 register. It is also possible to choose 0.5 stop bit for receiving.

For code example, refer to [A.19.9: USART smartcard mode code example](#).

In T=0 (character) mode, the parity error is indicated at the end of each character during the guard time period.

[Figure 263](#) shows examples of what can be seen on the data line with and without parity error.

Figure 263. ISO 7816-3 asynchronous protocol



When connected to a smartcard, the TX output of the USART drives a bidirectional line that is also driven by the smartcard. The TX pin must be configured as open drain.

Smartcard mode implements 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 starts shifting on the next baud clock edge. In Smartcard mode this transmission is further delayed by a guaranteed 1/2 baud clock.
- In transmission, if the smartcard detects a parity error, it signals this condition to the USART by driving the line low (NACK). This NACK signal (pulling transmit line low for 1 baud clock) causes a framing error on the transmitter side (configured with 1.5 stop bits). The USART can handle automatic re-sending of data according to the protocol. The number of retries is programmed in the SCARCNT bit field. If the USART continues receiving the NACK after the programmed number of retries, it stops transmitting and signals the error as a framing error. The TXE bit can be set using the TXFRQ bit in the USART\_RQR register.
- Smartcard auto-retry in transmission: a delay of 2.5 baud periods is inserted between the NACK detection by the USART and the start bit of the repeated character. The TC bit is set immediately at the end of reception of the last repeated character (no guard-time). If the software wants to repeat it again, it must insure the minimum 2 baud periods required by the standard.
- If a parity error is detected during reception of a frame programmed with a 1.5 stop bits period, the transmit line is pulled low for a baud clock period after the completion of the receive frame. This is to indicate to the smartcard that the data transmitted to the USART has not been correctly received. A parity error is NACKed by the receiver if the NACK control bit is set, otherwise a NACK is not transmitted (to be used in T=1 mode). If the received character is erroneous, the RXNE/receive DMA request is not activated. According to the protocol specification, the smartcard must resend the same character. If the received character is still erroneous after the maximum number of retries specified in the SCARCNT bit field, the USART stops transmitting the NACK and signals the error as a parity error.
- Smartcard auto-retry in reception: the BUSY flag remains set if the USART NACKs the card but the card doesn't repeat the character.

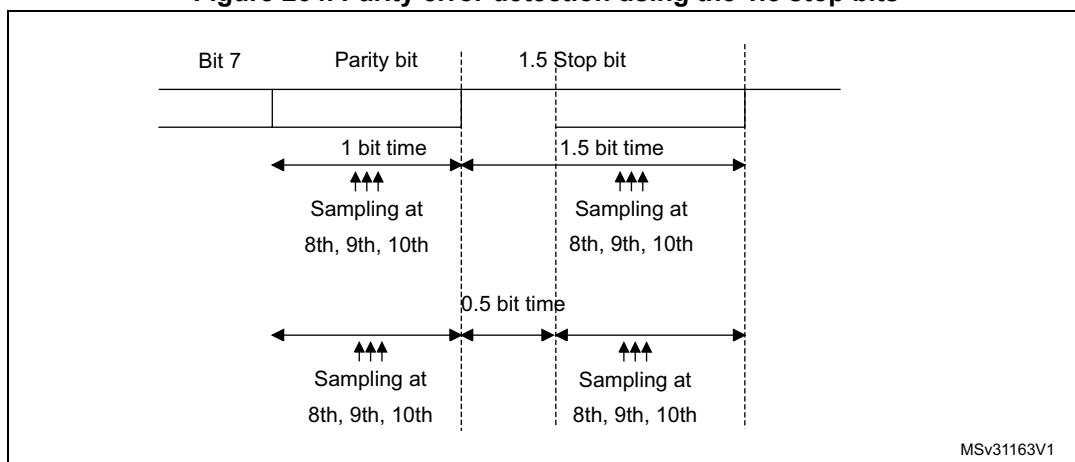
- In transmission, the USART inserts the Guard Time (as programmed in the Guard Time register) between two successive characters. As the Guard Time is measured after the stop bit of the previous character, the GT[7:0] register must be programmed to the desired CGT (Character Guard Time, as defined by the 7816-3 specification) minus 12 (the duration of one character).
- 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 is not 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 does not detect the NACK as a start bit.

*Note:* A break character is not significant in Smartcard mode. A 0x00 data with a framing error is treated as data and not as a break.

No Idle frame is transmitted when toggling the TE bit. The Idle frame (as defined for the other configurations) is not defined by the ISO protocol.

*Figure 264* details how the NACK signal is sampled by the USART. In this example the USART is transmitting data and is configured with 1.5 stop bits. The receiver part of the USART is enabled in order to check the integrity of the data and the NACK signal.

**Figure 264. Parity error detection using the 1.5 stop bits**



The USART can provide a clock to the smartcard through the CK output. In Smartcard mode, 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 USART\_GTPR. CK frequency can be programmed from  $f_{CK}/2$  to  $f_{CK}/62$ , where  $f_{CK}$  is the peripheral input clock.

## Block mode (T=1)

In T=1 (block) mode, the parity error transmission is deactivated, by clearing the NACK bit in the USART\_CR3 register.

When requesting a read from the smartcard, in block mode, the software must enable the receiver Timeout feature by setting the RTOEN bit in the USART\_CR2 register and program the RTO bits field in the RTOR register to the BWT (block wait time) - 11 value. If no answer is received from the card before the expiration of this period, the RTOF flag is set and a timeout interrupt is generated (if RTOIE bit in the USART\_CR1 register is set). If the first character is received before the expiration of the period, it is signaled by the RXNE interrupt.

**Note:** *The RXNE interrupt must be enabled even when using the USART in DMA mode to read from the smartcard in block mode. In parallel, the DMA must be enabled only after the first received byte.*

After the reception of the first character (RXNE interrupt), the RTO bit fields in the RTOR register must be programmed to the CWT (character wait time) - 11 value, in order to allow the automatic check of the maximum wait time between two consecutive characters. This time is expressed in baudtime units. If the smartcard does not send a new character in less than the CWT period after the end of the previous character, the USART signals this to the software through the RTOF flag and interrupt (when RTOIE bit is set).

**Note:** *The RTO counter starts counting:*

- From the end of the stop bit in case STOP = 00.
- From the end of the second stop bit in case of STOP = 10.
- 1 bit duration after the beginning of the STOP bit in case STOP = 11.
- From the beginning of the STOP bit in case STOP = 01.

*As in the Smartcard protocol definition, the BWT/CWT values are defined from the beginning (start bit) of the last character. The RTO register must be programmed to BWT - 11 or CWT -11, respectively, taking into account the length of the last character itself.*

A block length counter is used to count all the characters received by the USART. This counter is reset when the USART is transmitting (TXE=0). The length of the block is communicated by the smartcard in the third byte of the block (prologue field). This value must be programmed to the BLEN field in the USART\_RTOR register. When using DMA mode, before the start of the block, this register field must be programmed to the minimum value (0x0). With this value, an interrupt is generated after the 4th received character. The software must read the LEN field (third byte), its value must be read from the receive buffer.

In interrupt driven receive mode, the length of the block may be checked by software or by programming the BLEN value. However, before the start of the block, the maximum value of BLEN (0xFF) may be programmed. The real value is programmed after the reception of the third character.

If the block is using the LRC longitudinal redundancy check (1 epilogue byte), the BLEN=LEN. If the block is using the CRC mechanism (2 epilogue bytes), BLEN=LEN+1 must be programmed. The total block length (including prologue, epilogue and information fields) equals BLEN+4. The end of the block is signaled to the software through the EOBF flag and interrupt (when EOBIIE bit is set).

In case of an error in the block length, the end of the block is signaled by the RTO interrupt (Character wait Time overflow).

**Note:** *The error checking code (LRC/CRC) must be computed/verified by software.*

## Direct and inverse convention

The Smartcard protocol defines two conventions: direct and inverse.

The direct convention is defined as: LSB first, logical bit value of 1 corresponds to a H state of the line and parity is even. In order to use this convention, the following control bits must be programmed: MSBFIRST=0, DATAINV=0 (default values).

The inverse convention is defined as: MSB first, logical bit value 1 corresponds to an L state on the signal line and parity is even. In order to use this convention, the following control bits must be programmed: MSBFIRST=1, DATAINV=1.

**Note:** *When logical data values are inverted (0=H, 1=L), the parity bit is also inverted in the same way.*

In order to recognize the card convention, the card sends the initial character, TS, as the first character of the ATR (Answer To Reset) frame. The two possible patterns for the TS are: LHHL LLL LLH and LHHL HHH LLH.

- (H) LHHL LLL LLH sets up the inverse convention: state L encodes value 1 and moment 2 conveys the most significant bit (MSB first). when decoded by inverse convention, the conveyed byte is equal to '3F'.
- (H) LHHL HHH LLH sets up the direct convention: state H encodes value 1 and moment 2 conveys the least significant bit (LSB first). when decoded by direct convention, the conveyed byte is equal to '3B'.

Character parity is correct when there is an even number of bits set to 1 in the nine moments 2 to 10.

As the USART does not know which convention is used by the card, it needs to be able to recognize either pattern and act accordingly. The pattern recognition is not done in hardware, but through a software sequence. Moreover, supposing that the USART is configured in direct convention (default) and the card answers with the inverse convention, TS = LHHL LLL LLH => the USART received character is '03' and the parity is odd.

Therefore, two methods are available for TS pattern recognition:

### Method 1

The USART is programmed in standard Smartcard mode/direct convention. In this case, the TS pattern reception generates a parity error interrupt and error signal to the card.

- The parity error interrupt informs the software that the card didn't answer correctly in direct convention. Software then reprograms the USART for inverse convention
- In response to the error signal, the card retries the same TS character, and it is correctly received this time, by the reprogrammed USART

Alternatively, in answer to the parity error interrupt, the software may decide to reprogram the USART and to also generate a new reset command to the card, then wait again for the TS.

### Method 2

The USART is programmed in 9-bit/no-parity mode, no bit inversion. In this mode it receives any of the two TS patterns as:

- (H) LHHL LLL LLH = 0x103 -> inverse convention to be chosen
- (H) LHHL HHH LLH = 0x13B -> direct convention to be chosen

The software checks the received character against these two patterns and, if any of them match, then programs the USART accordingly for the next character reception.

If none of the two is recognized, a card reset may be generated in order to restart the negotiation.

### 27.5.14 USART IrDA SIR ENDEC block

This section is relevant only when IrDA mode is supported. Please refer to [Section 27.4: USART implementation on page 702](#).

IrDA mode is selected by setting the IREN bit in the USART\_CR3 register. In IrDA mode, the following bits must be kept cleared:

- LINEN, STOP and CLKEN bits in the USART\_CR2 register,
- SCEN and HDSEL bits in the USART\_CR3 register.

The IrDA SIR physical layer specifies use of a Return to Zero, Inverted (RZI) modulation scheme that represents logic 0 as an infrared light pulse (see [Figure 265](#)).

The SIR Transmit encoder modulates the Non Return to Zero (NRZ) transmit bit stream output from USART. The output pulse stream is transmitted to an external output driver and infrared LED. USART supports only bit rates up to 115.2 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 the USART. The decoder input is normally high (marking state) in the Idle state. The transmit encoder output has the opposite polarity to the decoder input. A start bit is detected when the decoder input is low.

- IrDA is a half duplex communication protocol. If the Transmitter is busy (when the USART is sending data to the IrDA encoder), any data on the IrDA receive line is ignored by the IrDA decoder and if the Receiver is busy (when the USART is receiving decoded data from the IrDA decoder), data on the TX from the USART to IrDA is not encoded. 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 266](#)).
- The SIR decoder converts the IrDA compliant receive signal into a bit stream for USART.
- The SIR receive logic interprets a high state as a logic one and low pulses as logic zeros.
- The transmit encoder output has the opposite polarity to the decoder input. The SIR output is in low state when Idle.
- The IrDA specification requires the acceptance of pulses greater than 1.41  $\mu$ s. The acceptable pulse width is programmable. Glitch detection logic on the receiver end filters out pulses of width less than 2 PSC periods (PSC is the prescaler value programmed in the USART\_GTPR). Pulses of width less than 1 PSC period are always rejected, but those of width greater than one and less than two periods may be accepted or rejected, those greater than 2 periods are accepted as a pulse. The IrDA encoder/decoder doesn't work when PSC=0.
- The receiver can communicate with a low-power transmitter.
- In IrDA mode, the STOP bits in the USART\_CR2 register must be configured to "1 stop bit".

## IrDA low-power mode

### Transmitter

In low-power mode the pulse width is not maintained at 3/16 of the bit period. Instead, the width of the pulse is 3 times the low-power baud rate which can be a minimum of 1.42 MHz.

Generally, this value is 1.8432 MHz (1.42 MHz < PSC < 2.12 MHz). A low-power mode programmable divisor divides the system clock to achieve this value.

### Receiver

Receiving in low-power mode is similar to receiving in normal mode. For glitch detection the USART should discard pulses of duration shorter than 1 PSC period. A valid low is accepted only if its duration is greater than 2 periods of the IrDA low-power Baud clock (PSC value in the USART\_GTPR).

**Note:** *A pulse of width less than two and greater than one PSC period(s) may or may not be rejected.*

*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 265. IrDA SIR ENDEC- block diagram

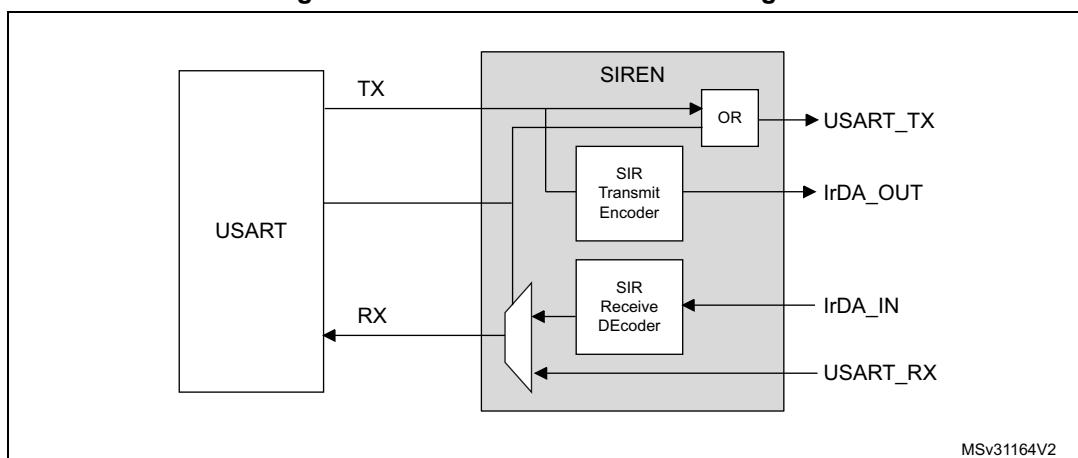
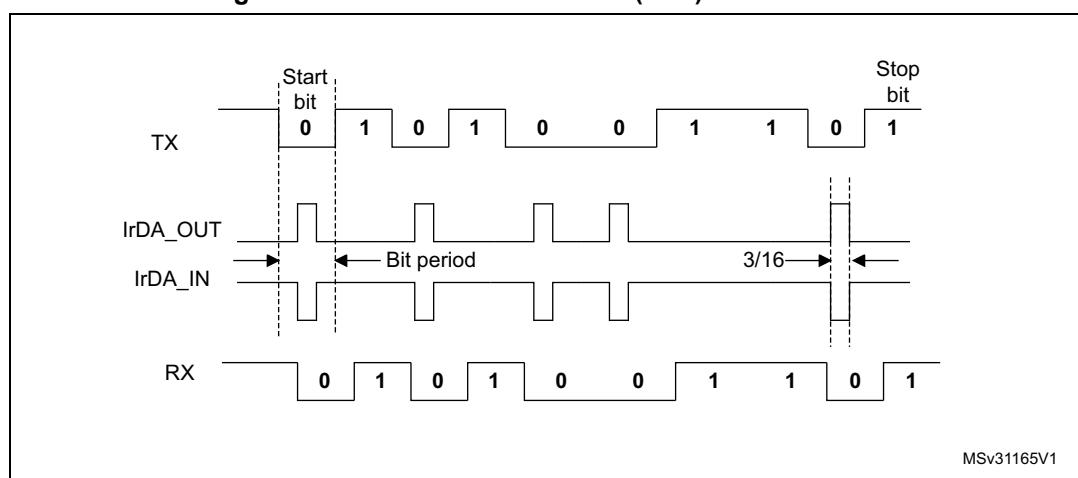


Figure 266. IrDA data modulation (3/16) -Normal Mode



### 27.5.15 USART continuous communication in DMA mode

The USART is capable of performing continuous communication using the DMA. The DMA requests for Rx buffer and Tx buffer are generated independently.

**Note:** Please refer to [Section 27.4: USART implementation on page 702](#) to determine if the DMA mode is supported. If DMA is not supported, use the USART as explained in [Section 27.5.2: USART transmitter](#) or [Section 27.5.3: USART receiver](#). To perform continuous communication, the user can clear the TXE/ RXNE flags in the USART\_ISR register.

For code example, refer to [A.19.11: USART DMA code example](#).

#### Transmission using DMA

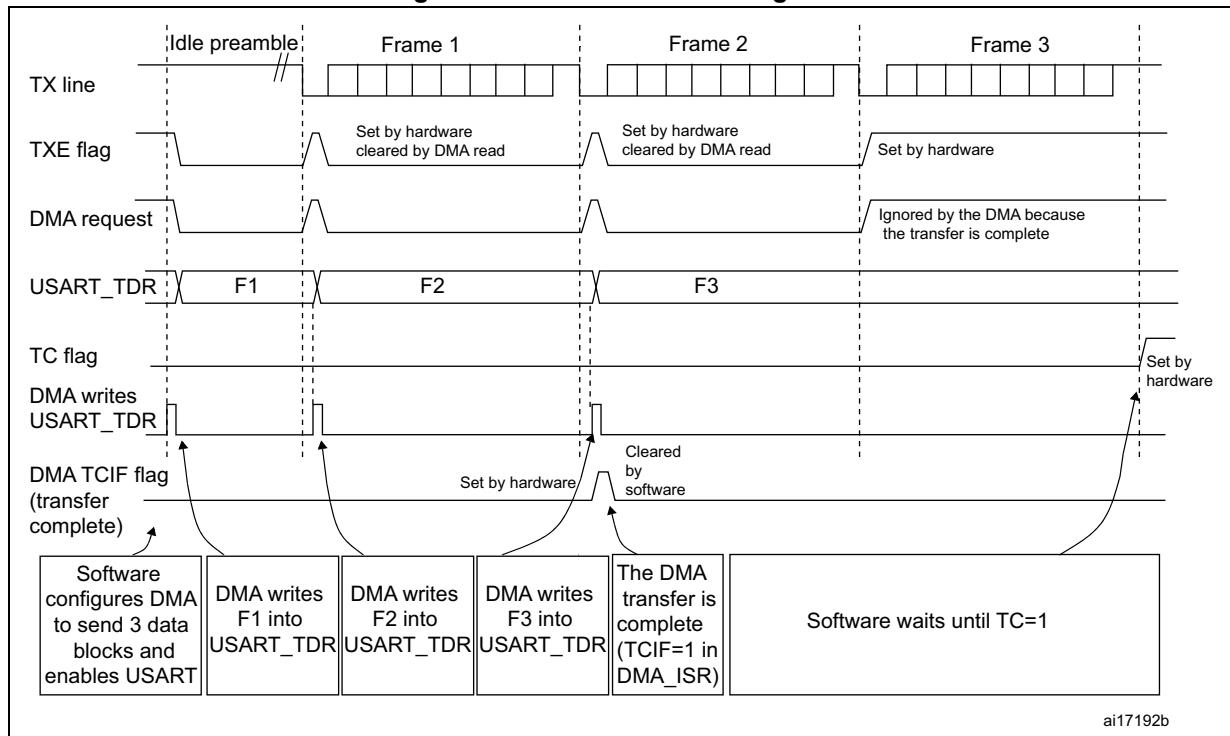
DMA mode can be enabled for transmission by setting DMAT bit in the USART\_CR3 register. Data is loaded from a SRAM area configured using the DMA peripheral (refer to [Section 10: Direct memory access controller \(DMA\) on page 188](#)) to the USART\_TDR register whenever the TXE bit is set. To map a DMA channel for USART transmission, use the following procedure (x denotes the channel number):

1. Write the USART\_TDR register address in the DMA control register to configure it as the destination of the transfer. The data is moved to this address from memory after each TXE event.
2. Write the memory address in the DMA control register to configure it as the source of the transfer. The data is loaded into the USART\_TDR register from this memory area after each TXE event.
3. Configure the total number of bytes to be transferred to the DMA control register.
4. Configure the channel priority in the DMA register
5. Configure DMA interrupt generation after half/ full transfer as required by the application.
6. Clear the TC flag in the USART\_ISR register by setting the TCCF bit in the USART\_ICR register.
7. Activate the channel in the DMA register.

When the number of data transfers programmed in the DMA Controller is reached, the DMA controller generates an interrupt on the DMA channel interrupt vector.

In transmission mode, once the DMA has written all the data to be transmitted (the TCIF flag is set in the DMA\_ISR register), the TC flag can be monitored to make sure that the USART communication is complete. This is required to avoid corrupting the last transmission before disabling the USART or entering Stop mode. Software must wait until TC=1. The TC flag remains cleared during all data transfers and it is set by hardware at the end of transmission of the last frame.

Figure 267. Transmission using DMA



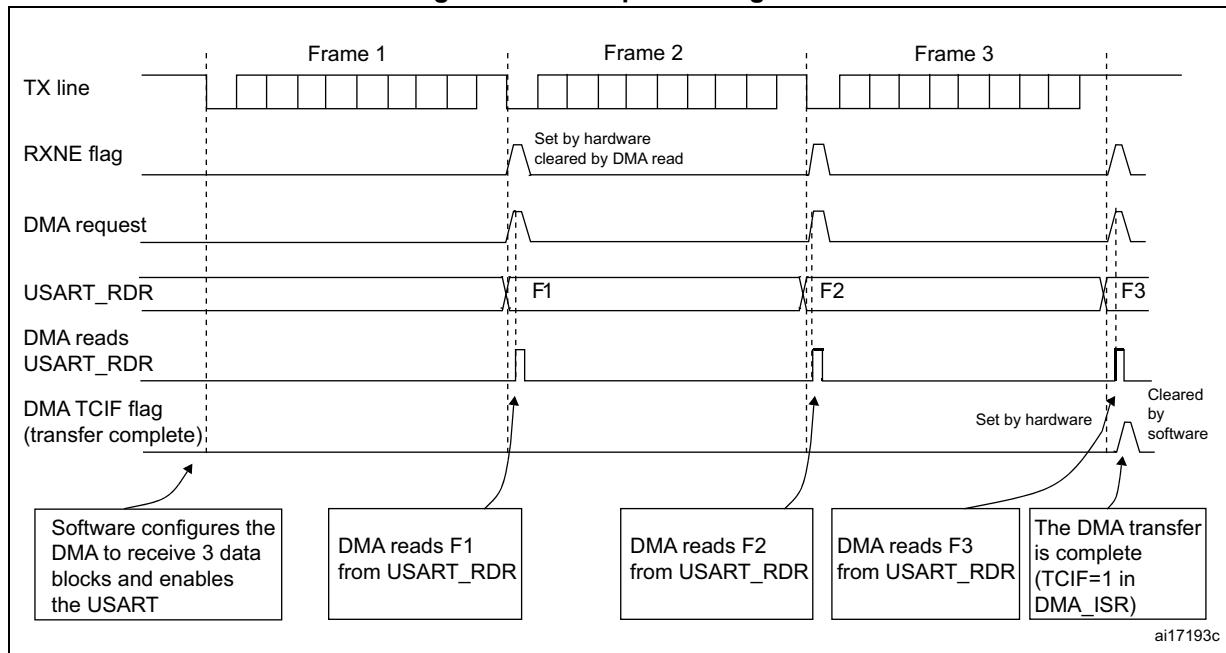
### Reception using DMA

DMA mode can be enabled for reception by setting the DMAR bit in USART\_CR3 register. Data is loaded from the USART\_RDR register to a SRAM area configured using the DMA peripheral (refer to [Section 10: Direct memory access controller \(DMA\) on page 188](#)) whenever a data byte is received. To map a DMA channel for USART reception, use the following procedure:

1. Write the USART\_RDR register address in the DMA control register to configure it as the source of the transfer. The data is moved from this address to the memory after each RXNE event.
2. Write the memory address in the DMA control register to configure it as the destination of the transfer. The data is loaded from USART\_RDR to this memory area after each RXNE event.
3. Configure the total number of bytes to be transferred to the DMA control register.
4. Configure the channel priority in the DMA control register
5. Configure interrupt generation after half/ full transfer as required by the application.
6. Activate the channel in the DMA control register.

When the number of data transfers programmed in the DMA Controller is reached, the DMA controller generates an interrupt on the DMA channel interrupt vector.

Figure 268. Reception using DMA



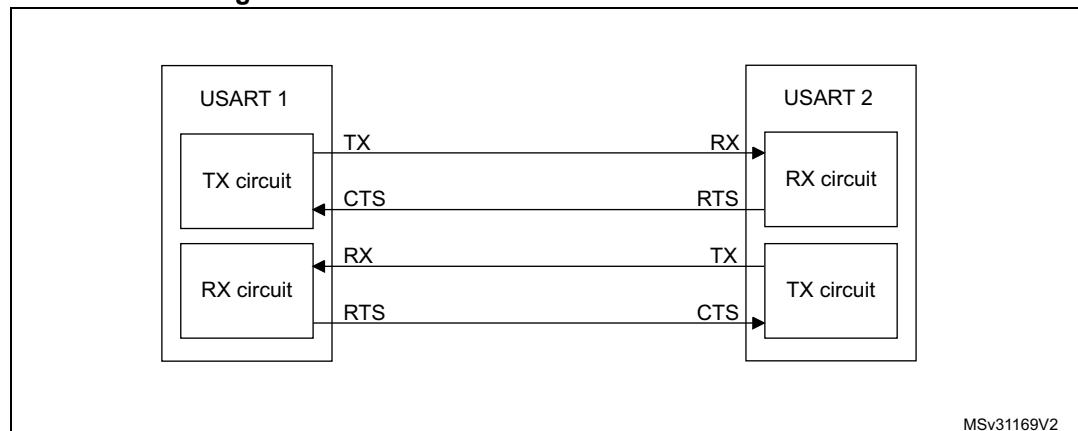
### Error flagging and interrupt generation in multibuffer communication

In multibuffer communication if any error occurs during the transaction the error flag is asserted after the current byte. An interrupt is generated if the interrupt enable flag is set. For framing error, overrun error and noise flag which are asserted with RXNE in single byte reception, there is a separate error flag interrupt enable bit (EIE bit in the USART\_CR3 register), which, if set, enables an interrupt after the current byte if any of these errors occur.

### 27.5.16 RS232 hardware flow control and RS485 driver enable using USART

It is possible to control the serial data flow between 2 devices by using the CTS input and the RTS output. The [Figure 269](#) shows how to connect 2 devices in this mode:

Figure 269. Hardware flow control between 2 USARTs

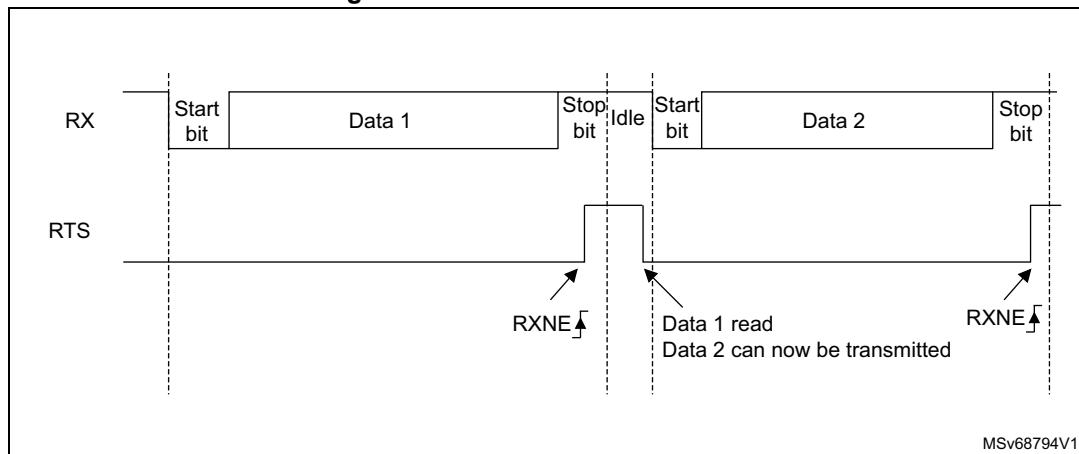


RS232 RTS and CTS flow control can be enabled independently by writing the RTSE and CTSE bits respectively to 1 (in the USART\_CR3 register).

### RS232 RTS flow control

If the RTS flow control is enabled (RTSE=1), then RTS is deasserted (tied low) as long as the USART receiver is ready to receive a new data. When the receive register is full, RTS is asserted, indicating that the transmission is expected to stop at the end of the current frame. [Figure 270](#) shows an example of communication with RTS flow control enabled.

**Figure 270. RS232 RTS flow control**

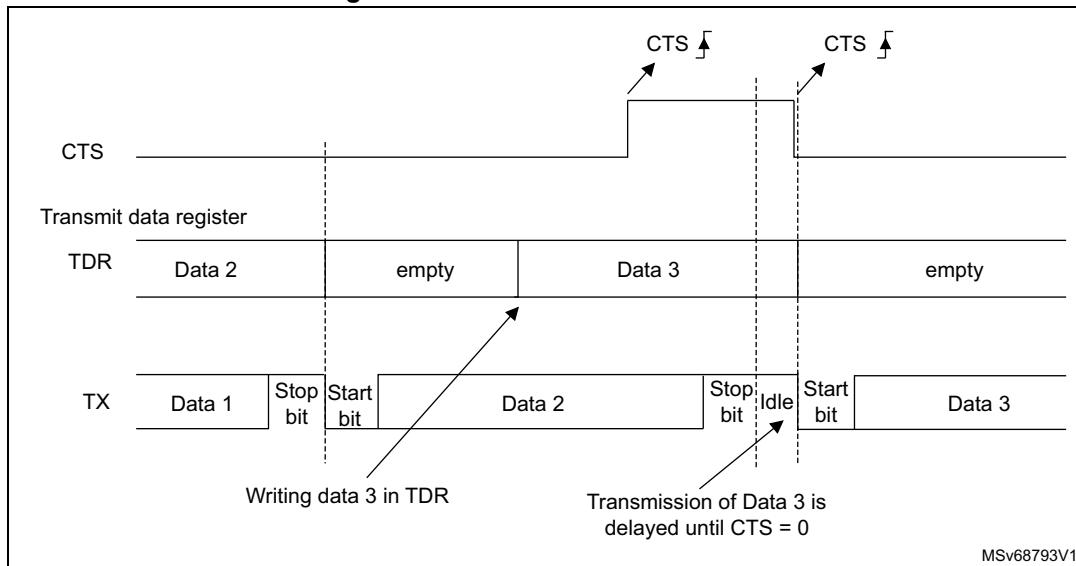


### RS232 CTS flow control

If the CTS flow control is enabled (CTSE=1), then the transmitter checks the CTS input before transmitting the next frame. If CTS is deasserted (tied low), then the next data is transmitted (assuming that data is to be transmitted, in other words, if TXE=0), else the transmission does not occur. When CTS is asserted during a transmission, the current transmission is completed before the transmitter stops.

When CTSE=1, the CTSIF status bit is automatically set by hardware as soon as the CTS input toggles. It indicates when the receiver becomes ready or not ready for communication. An interrupt is generated if the CTSIE bit in the USART\_CR3 register is set. [Figure 271](#) shows an example of communication with CTS flow control enabled.

Figure 271. RS232 CTS flow control



**Note:** For correct behavior, CTS must be deasserted at least 3 USART clock source periods before the end of the current character. In addition it should be noted that the CTSCF flag may not be set for pulses shorter than 2 x PCLK periods.

For code example, refer to [A.19.12: USART hardware flow control code example](#).

### RS485 Driver Enable

The driver enable feature is enabled by setting bit DEM in the USART\_CR3 control register. This allows the user to activate the external transceiver control, through the DE (Driver Enable) signal. The assertion time is the time between the activation of the DE signal and the beginning of the START bit. It is programmed using the DEAT [4:0] bit fields in the USART\_CR1 control register. The de-assertion time is the time between the end of the last stop bit, in a transmitted message, and the de-activation of the DE signal. It is programmed using the DEDT [4:0] bit fields in the USART\_CR1 control register. The polarity of the DE signal can be configured using the DEP bit in the USART\_CR3 control register.

In USART, the DEAT and DEDT are expressed in sample time units (1/8 or 1/16 bit duration, depending on the oversampling rate).

#### 27.5.17 Wake-up from Stop mode using USART

The USART is able to wake up the MCU from Stopmode when the UESM bit is set and the USART clock is set to HSI or LSE (refer to Section Reset and clock control (RCC)).

- USART source clock is HSI

If during Stop mode the HSI clock is switched OFF, when a falling edge on the USART receive line is detected, the USART interface requests the HSI clock to be switched ON. The HSI clock is then used for the frame reception.

- If the wake-up event is verified, the MCU wakes up from low-power mode and data reception goes on normally.
- If the wake-up event is not verified, the HSI clock is switched OFF again, the MCU is not waken up and stays in low-power mode and the clock request is released.

**Note:** If the USART kernel clock is kept ON during Stop mode, there is no constraint on the maximum baud rate that allows waking up from Stop mode. It is the same as in Run mode.

- USART source clock is LSE

Same principle as described in case of USART source clock is HSI with the difference that the LSE is ON in Stop mode, but the LSE clock is not propagated to USART if the USART is not requesting it. The LSE clock is not OFF but there is a clock gating to avoid useless consumption.

The MCU wake-up from Stop mode can be done using the standard RXNE interrupt. In this case, the RXNEIE bit must be set before entering Stop mode.

Alternatively, a specific interrupt may be selected through the WUS bit fields.

In order to be able to wake up the MCU from Stop mode, the UESM bit in the USART\_CR1 control register must be set prior to entering Stop mode.

When the wake-up event is detected, the WUF flag is set by hardware and a wake-up interrupt is generated if the WUFIE bit is set.

**Note:** Before entering Stop mode, the user must ensure that the USART is not performing a transfer. BUSY flag cannot ensure that Stop mode is never entered during a running reception.

The WUF flag is set when a wake-up event is detected, independently of whether the MCU is in Stop or in an active mode.

When entering Stop mode just after having initialized and enabled the receiver, the REACK bit must be checked to ensure the USART is actually enabled.

When DMA is used for reception, it must be disabled before entering Stop mode and re-enabled upon exit from Stop mode.

The wake-up from Stop mode feature is not available for all modes. For example it doesn't work in SPI mode because the SPI operates in master mode only.

### Using Mute mode with Stop mode

If the USART is put into Mute mode before entering Stop mode:

- Wake-up from Mute mode on idle detection must not be used, because idle detection cannot work in Stop mode.
- If the wake-up from Mute mode on address match is used, then the source of wake-up from Stop mode must also be the address match. If the RXNE flag is set when entering the Stop mode, the interface will remain in mute mode upon address match and wake up from Stop.
- If the USART is configured to wake up the MCU from Stop mode on START bit detection, the WUF flag is set, but the RXNE flag is not set.

### Determining the maximum USART baud rate allowing to wake-up correctly from Stop mode when the USART clock source is the HSI clock

The maximum baud rate allowing to wake-up correctly from Stop mode depends on:

- the parameter  $t_{WUUSART}$  provided in the device datasheet
- the USART receiver tolerance provided in the [Section 27.5.5: Tolerance of the USART receiver to clock deviation](#).

Let us take this example: OVER8 = 0, M bits = 01, ONEBIT = 0, BRR [3:0] = 0000.

In these conditions, according to [Table 107: Tolerance of the USART receiver when BRR \[3:0\] = 0000](#), the USART receiver tolerance is 3.41 %.

$DTRA + DQUANT + DREC + DTCL + DWU < \text{USART receiver's tolerance}$

$$DWU_{\text{max}} = t_{\text{WUUSART}} / (9 \times \text{Tbit Min})$$

$$\text{Tbit Min} = t_{\text{WUUSART}} / (9 \times DWU_{\text{max}})$$

If we consider an ideal case where the parameters DTRA, DQUANT, DREC and DTCL are at 0%, the DWU max is 3.41 %. In reality, we need to consider at least the HSI inaccuracy.

Let us consider HSI inaccuracy = 1 %,  $t_{\text{WUUSART}} = 3 \mu\text{s}$  (values provided as example, for correct values, refer to the device datasheet):

$$DWU_{\text{max}} = 3.41\% - 1\% = 2.41\%$$

$$\text{Tbit min} = 3 \mu\text{s} / (11 \times 2.41\%) = 11.32 \mu\text{s}$$

In these conditions, the maximum baud rate allowing to wake-up correctly from Stop mode is  $1/11.32 \mu\text{s} = 88.36 \text{ kbaud}$ .

## 27.6 USART in low-power modes

**Table 110. Effect of low-power modes on the USART**

Mode	Description
Sleep	No effect. USART interrupt causes the device to exit Sleep mode.
Stop	The USART is able to wake up the MCU from Stop mode when the UESM bit is set and the USART clock is set to HSI or LSE. The MCU wake-up from Stop mode can be done using either a standard RXNE or a WUF interrupt.
Standby	The USART is powered down and must be reinitialized when the device has exited from Standby mode.

## 27.7 USART interrupts

**Table 111. USART interrupt requests**

Interrupt event	Event flag	Enable Control bit
Transmit data register empty	TXE	TXEIE
CTS interrupt	CTSIF	CTSIE
Transmission Complete	TC	TCIE
Receive data register not empty (data ready to be read)	RXNE	RXNEIE
Overrun error detected	ORE	
Idle line detected	IDLE	IDLEIE
Parity error	PE	PEIE
LIN break	LBDIF	LBDIE

Table 111. USART interrupt requests (continued)

Interrupt event	Event flag	Enable Control bit
Noise Flag, Overrun error and Framing Error in multibuffer communication.	NF or ORE or FE	EIE
Character match	CMF	CMIE
Receiver timeout	RTOF	RTOIE
End of Block	EOBF	EOBIE
Wake-up from Stop mode	WUF <sup>(1)</sup>	WUFIE

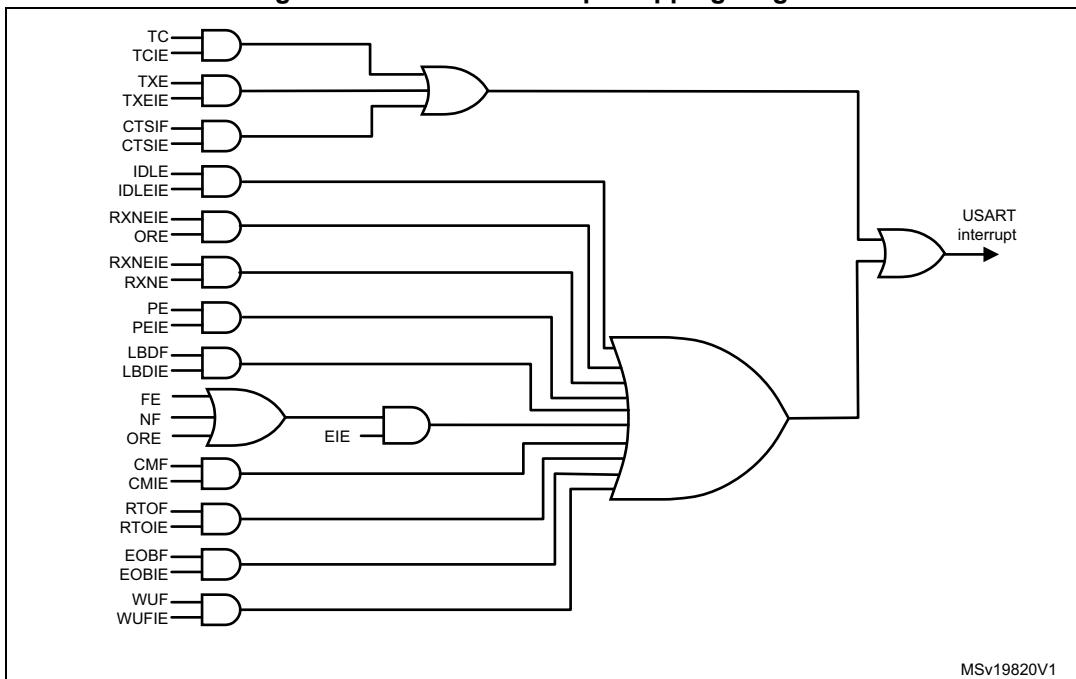
1. The WUF interrupt is active only in Stop mode.

The USART interrupt events are connected to the same interrupt vector (see [Figure 272](#)).

- During transmission: Transmission Complete, Clear to Send, Transmit data Register empty or Framing error (in Smartcard mode) interrupt.
- During reception: Idle Line detection, Overrun error, Receive data register not empty, Parity error, LIN break detection, Noise Flag, Framing Error, Character match, etc.

These events generate an interrupt if the corresponding Enable Control Bit is set.

Figure 272. USART interrupt mapping diagram



MSv19820V1

## 27.8 USART registers

Refer to [Section 1.2 on page 42](#) for a list of abbreviations used in register descriptions.

The peripheral registers have to be accessed by words (32 bits).

### 27.8.1 USART control register 1 (USART\_CR1)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	M1	EOBIE	RTOIE	DEAT[4:0]					DEDT[4:0]				
			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OVER8	CMIE	MME	M0	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	UESM	UE
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

#### Bit 28 M1: Word length

This bit, with bit 12 (M0), determines the word length. It is set or cleared by software.

M[1:0] = 00: 1 Start bit, 8 data bits, n stop bits

M[1:0] = 01: 1 Start bit, 9 data bits, n stop bits

M[1:0] = 10: 1 Start bit, 7 data bits, n stop bits

This bit can only be written when the USART is disabled (UE=0).

*Note:* Not all modes are supported In 7-bit data length mode. Refer to [Section 27.4: USART implementation](#) for details.

#### Bit 27 EOBIE: End of Block interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A USART interrupt is generated when the EOBF flag is set in the USART\_ISR register.

*Note:* If the USART does not support Smartcard mode, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).

#### Bit 26 RTOIE: Receiver timeout interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated when the RTOF bit is set in the USART\_ISR register.

*Note:* If the USART does not support the Receiver timeout feature, this bit is reserved and must be kept at reset value. [Section 27.4: USART implementation on page 702](#).

#### Bits 25:21 DEAT[4:0]: Driver Enable assertion time

This 5-bit value defines the time between the activation of the DE (Driver Enable) signal and the beginning of the start bit. It is expressed in sample time units (1/8 or 1/16 bit duration, depending on the oversampling rate).

This bit field can only be written when the USART is disabled (UE=0).

*Note:* If the Driver Enable feature is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).

Bits 20:16 **DEDT[4:0]**: Driver Enable de-assertion time

This 5-bit value defines the time between the end of the last stop bit, in a transmitted message, and the de-activation of the DE (Driver Enable) signal. It is expressed in sample time units (1/8 or 1/16 bit duration, depending on the oversampling rate).

If the USART\_TDR register is written during the DEDT time, the new data is transmitted only when the DEDT and DEAT times have both elapsed.

This bit field can only be written when the USART is disabled (UE=0).

*Note:* If the Driver Enable feature is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).

Bit 15 **OVER8**: Oversampling mode

- 0: Oversampling by 16
- 1: Oversampling by 8

This bit can only be written when the USART is disabled (UE=0).

*Note:* In LIN, IrDA and modes, this bit must be kept at reset value.

Bit 14 **CMIE**: Character match interrupt enable

This bit is set and cleared by software.

- 0: Interrupt is inhibited
- 1: A USART interrupt is generated when the CMF bit is set in the USART\_ISR register.

Bit 13 **MME**: Mute mode enable

This bit activates the mute mode function of the USART. When set, the USART can switch between the active and mute modes, as defined by the WAKE bit. It is set and cleared by software.

- 0: Receiver in active mode permanently
- 1: Receiver can switch between mute mode and active mode.

Bit 12 **M0**: Word length

This bit, with bit 28 (M1), determines the word length. It is set or cleared by software. See Bit 28 (M1) description.  
This bit can only be written when the USART is disabled (UE=0).

Bit 11 **WAKE**: Receiver wake-up method

This bit determines the USART wake-up method from Mute mode. It is set or cleared by software.

- 0: Idle line
- 1: Address mark

This bit field can only be written when the USART is disabled (UE=0).

Bit 10 **PCE**: Parity control enable

This bit selects the hardware parity control (generation and detection). When the parity control is enabled, the computed parity is inserted at the MSB position (9th bit if M=1; 8th bit if M=0) and parity is checked on the received data. This bit is set and cleared by software. Once it is set, PCE is active after the current byte (in reception and in transmission).

- 0: Parity control disabled
- 1: Parity control enabled

This bit field can only be written when the USART is disabled (UE=0).

Bit 9 **PS**: Parity selection

This bit selects the odd or even parity when the parity generation/detection is enabled (PCE bit set). It is set and cleared by software. The parity is selected after the current byte.

- 0: Even parity
- 1: Odd parity

This bit field can only be written when the USART is disabled (UE=0).

**Bit 8 PEIE:** PE interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A USART interrupt is generated whenever PE=1 in the USART\_ISR register

**Bit 7 TXEIE:** interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A USART interrupt is generated whenever TXE=1 in the USART\_ISR register

**Bit 6 TCIE:** Transmission complete interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A USART interrupt is generated whenever TC=1 in the USART\_ISR register

**Bit 5 RXNEIE:** RXNE interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A USART interrupt is generated whenever ORE=1 or RXNE=1 in the USART\_ISR register

**Bit 4 IDLEIE:** IDLE interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A USART interrupt is generated whenever IDLE=1 in the USART\_ISR register

**Bit 3 TE:** Transmitter enable

This bit enables the transmitter. It is set and cleared by software.

0: Transmitter is disabled

1: Transmitter is enabled

*Note: During transmission, a “0” pulse on the TE bit (“0” followed by “1”) sends a preamble (idle line) after the current word, except in Smartcard mode. In order to generate an idle character, the TE must not be immediately written to 1. In order to ensure the required duration, the software can poll the TEACK bit in the USART\_ISR register.*

*In Smartcard mode, when TE is set there is a 1 bit-time delay before the transmission starts.*

**Bit 2 RE:** Receiver enable

This bit enables the receiver. It is set and cleared by software.

0: Receiver is disabled

1: Receiver is enabled and begins searching for a start bit

**Bit 1 UESM:** USART enable in Stop mode

When this bit is cleared, the USART is not able to wake up the MCU from Stop mode.

When this bit is set, the USART is able to wake up the MCU from Stop mode, provided that the USART clock selection is HSI or LSE in the RCC.

This bit is set and cleared by software.

0: USART not able to wake up the MCU from Stop mode.

1: USART able to wake up the MCU from Stop mode. When this function is active, the clock source for the USART must be HSI or LSE (see Section Reset and clock control (RCC)).

*Note: It is recommended to set the UESM bit just before entering Stop mode and clear it on exit from Stop mode.*

*If the USART does not support the wake-up from Stop feature, this bit is reserved and must be kept at reset value. Refer to Section 27.4: USART implementation on page 702.*

**Bit 0 UE:** USART enable

When this bit is cleared, the USART prescalers and outputs are stopped immediately, and current operations are discarded. The configuration of the USART is kept, but all the status flags, in the USART\_ISR are set to their default values. This bit is set and cleared by software.

0: USART prescaler and outputs disabled, low-power mode

1: USART enabled

*Note: In order to go into low-power mode without generating errors on the line, the TE bit must be reset before and the software must wait for the TC bit in the USART\_ISR to be set before resetting the UE bit.*

*The DMA requests are also reset when UE = 0 so the DMA channel must be disabled before resetting the UE bit.*

### 27.8.2 USART control register 2 (USART\_CR2)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWAP	LINEN		STOP[1:0]	CLKEN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	ADDM7	Res.	Res.	Res.	Res.
rw	rw		rw	rw	rw	rw	rw		rw	rw	rw				

**Bits 31:28 ADD[7:4]: Address of the USART node**

This bit-field gives the address of the USART node or a character code to be recognized.

This is used in multiprocessor communication during Mute mode or Stop mode, for wake-up with 7-bit address mark detection. The MSB of the character sent by the transmitter should be equal to 1. It may also be used for character detection during normal reception, Mute mode inactive (for example, end of block detection in ModBus protocol). In this case, the whole received character (8-bit) is compared to the ADD[7:0] value and CMF flag is set on match.

This bit field can only be written when reception is disabled (RE = 0) or the USART is disabled (UE=0)

**Bits 27:24 ADD[3:0]: Address of the USART node**

This bit-field gives the address of the USART node or a character code to be recognized.

This is used in multiprocessor communication during Mute mode or Stop mode, for wake-up with address mark detection.

This bit field can only be written when reception is disabled (RE = 0) or the USART is disabled (UE=0)

**Bit 23 RTOEN: Receiver timeout enable**

This bit is set and cleared by software.

0: Receiver timeout feature disabled.

1: Receiver timeout feature enabled.

When this feature is enabled, the RTOF flag in the USART\_ISR register is set if the RX line is idle (no reception) for the duration programmed in the RTOR (receiver timeout register).

*Note: If the USART does not support the Receiver timeout feature, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).*

**Bits 22:21 ABRMOD[1:0]: Auto baud rate mode**

These bits are set and cleared by software.

00: Measurement of the start bit is used to detect the baud rate.

01: Falling edge to falling edge measurement. (the received frame must start with a single bit = 1 -> Frame = Start10xxxxxx)

10: 0x7F frame detection.

11: 0x55 frame detection

This bit field can only be written when ABREN = 0 or the USART is disabled (UE=0).

*Note: If DATAINV=1 and/or MSBFIRST=1 the patterns must be the same on the line, for example 0xAA for MSBFIRST)*

*If the USART does not support the auto baud rate feature, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).*

**Bit 20 ABREN: Auto baud rate enable**

This bit is set and cleared by software.

0: Auto baud rate detection is disabled.

1: Auto baud rate detection is enabled.

*Note: If the USART does not support the auto baud rate feature, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).*

**Bit 19 MSBFIRST: Most significant bit first**

This bit is set and cleared by software.

0: data is transmitted/received with data bit 0 first, following the start bit.

1: data is transmitted/received with the MSB (bit 7/8/9) first, following the start bit.

This bit field can only be written when the USART is disabled (UE=0).

**Bit 18 DATAINV:** Binary data inversion

This bit is set and cleared by software.

- 0: Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)
- 1: Logical data from the data register are send/received in negative/inverse logic. (1=L, 0=H). The parity bit is also inverted.

This bit field can only be written when the USART is disabled (UE=0).

**Bit 17 TXINV:** TX pin active level inversion

This bit is set and cleared by software.

- 0: TX pin signal works using the standard logic levels ( $V_{DD} = 1/\text{idle}$ , Gnd=0/mark)
- 1: TX pin signal values are inverted. ( $V_{DD} = 0/\text{mark}$ , Gnd=1/idle).

This allows the use of an external inverter on the TX line.

This bit field can only be written when the USART is disabled (UE=0).

**Bit 16 RXINV:** RX pin active level inversion

This bit is set and cleared by software.

- 0: RX pin signal works using the standard logic levels ( $V_{DD} = 1/\text{idle}$ , Gnd=0/mark)
- 1: RX pin signal values are inverted. ( $V_{DD} = 0/\text{mark}$ , Gnd=1/idle).

This allows the use of an external inverter on the RX line.

This bit field can only be written when the USART is disabled (UE=0).

**Bit 15 SWAP:** Swap TX/RX pins

This bit is set and cleared by software.

- 0: TX/RX pins are used as defined in standard pinout
- 1: The TX and RX pins functions are swapped. This allows to work in the case of a cross-wired connection to another USART.

This bit field can only be written when the USART is disabled (UE=0).

**Bit 14 LINEN:** LIN mode enable

This bit is set and cleared by software.

- 0: LIN mode disabled
- 1: LIN mode enabled

The LIN mode enables the capability to send LIN synchronous breaks (13 low bits) using the SBKRQ bit in the USART\_RQR register, and to detect LIN Sync breaks.

This bit field can only be written when the USART is disabled (UE=0).

*Note: If the USART does not support LIN mode, this bit is reserved and must be kept at reset value.  
Please refer to [Section 27.4: USART implementation on page 702](#).*

**Bits 13:12 STOP[1:0]:** STOP bits

These bits are used for programming the stop bits.

- 00: 1 stop bit
- 01: 0.5 stop bit
- 10: 2 stop bits
- 11: 1.5 stop bits

This bit field can only be written when the USART is disabled (UE=0).

Bit 11 **CLKEN**: Clock enable

This bit allows the user to enable the CK pin.

0: CK pin disabled

1: CK pin enabled

This bit can only be written when the USART is disabled (UE=0).

*Note: If neither synchronous mode nor Smartcard mode is supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).*

*In order to provide correctly the CK clock to the Smartcard when CK is always available When CLKEN = 1, regardless of the UE bit value, the steps below must be respected:*

- UE = 0

- SCEN = 1

- GTPR configuration (If PSC needs to be configured, it is recommended to configure PSC and GT in a single access to USART\_GTPR register).

- CLKEN= 1

- UE = 1

Bit 10 **CPOL**: Clock polarity

This bit allows the user to select the polarity of the clock output on the CK pin in synchronous mode.

It works in conjunction with the CPHA bit to produce the desired clock/data relationship

0: Steady low value on CK pin outside transmission window

1: Steady high value on CK pin outside transmission window

This bit can only be written when the USART is disabled (UE=0).

*Note: If synchronous mode is not supported, this bit is reserved and must be kept at reset value.*

*Please refer to [Section 27.4: USART implementation on page 702](#).*

Bit 9 **CPHA**: Clock phase

This bit is used to select the phase of the clock output on the CK pin in synchronous mode. It works in conjunction with the CPOL bit to produce the desired clock/data relationship (see [Figure 260](#) and [Figure 261](#))

0: The first clock transition is the first data capture edge

1: The second clock transition is the first data capture edge

This bit can only be written when the USART is disabled (UE=0).

*Note: If synchronous mode is not supported, this bit is reserved and must be kept at reset value.*

*Please refer to [Section 27.4: USART implementation on page 702](#).*

Bit 8 **LBCL**: Last bit clock pulse

This bit is used to select whether the clock pulse associated with the last data bit transmitted (MSB) has to be output on the CK pin in synchronous mode.

0: The clock pulse of the last data bit is not output to the CK pin

1: The clock pulse of the last data bit is output to the CK pin

**Caution:** The last bit is the 7th or 8th or 9th data bit transmitted depending on the 7 or 8 or 9 bit format selected by the M bits in the USART\_CR1 register.

This bit can only be written when the USART is disabled (UE=0).

*Note: If synchronous mode is not supported, this bit is reserved and must be kept at reset value.*

*Please refer to [Section 27.4: USART implementation on page 702](#).*

## Bit 7 Reserved, must be kept at reset value.

Bit 6 **LBDIE**: LIN break detection interrupt enable

Break interrupt mask (break detection using break delimiter).

0: Interrupt is inhibited

1: An interrupt is generated whenever LBDF=1 in the USART\_ISR register

*Note: If LIN mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).*

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

This bit can only be written when the USART is disabled (UE=0).

*Note: If LIN mode is not supported, this bit is reserved and must be kept at reset value. Please refer to Section 27.4: USART implementation on page 702.*

Bit 4 **ADDM7**:7-bit Address Detection/4-bit Address Detection

This bit is for selection between 4-bit address detection or 7-bit address detection.

0: 4-bit address detection

1: 7-bit address detection (in 8-bit data mode)

This bit can only be written when the USART is disabled (UE=0)

*Note: In 7-bit and 9-bit data modes, the address detection is done on 6-bit and 8-bit address (ADD[5:0] and ADD[7:0]) respectively.*

Bits 3:0 Reserved, must be kept at reset value.

*Note: The 3 bits (CPOL, CPHA, LBCL) should not be written while the transmitter is enabled.*

**27.8.3 USART control register 3 (USART\_CR3)**

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	WUFIE	WUS1	WUS0	SCARC NT2	SCARC NT1	SCARC NT0	Res.
									rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DEP	DEM	DDRE	OVRDI S	ONEBI T	CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:25 Reserved, must be kept at reset value.

Bit 24 Reserved, must be kept at reset value.

Bit 23 Reserved, must be kept at reset value.

Bit 22 **WUFIE**: Wake-up from Stop mode interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated whenever WUF=1 in the USART\_ISR register

*Note: WUFIE must be set before entering in Stop mode.*

*The WUF interrupt is active only in Stop mode.*

*If the USART does not support the wake-up from Stop feature, this bit is reserved and must be kept at reset value.*

Bits 21:20 **WUS[1:0]**: Wake-up from Stop mode interrupt flag selection

This bit-field specify the event which activates the WUF (wake-up from Stop mode flag).

00: WUF active on address match (as defined by ADD[7:0] and ADDM7)

01:Reserved.

10: WuF active on Start bit detection

11: WUF active on RXNE.

This bit field can only be written when the USART is disabled (UE=0).

*Note: If the USART does not support the wake-up from Stop feature, this bit is reserved and must be kept at reset value.*

Bits 19:17 **SCARCNT[2:0]**: Smartcard auto-retry count

This bit-field specifies the number of retries in transmit and receive, in Smartcard mode.

In transmission mode, it specifies the number of automatic retransmission retries, before generating a transmission error (FE bit set).

In reception mode, it specifies the number of erroneous reception trials, before generating a reception error (RXNE and PE bits set).

This bit field must be programmed only when the USART is disabled (UE=0).

When the USART is enabled (UE=1), this bit field may only be written to 0x0, in order to stop retransmission.

0x0: retransmission disabled - No automatic retransmission in transmit mode.

0x1 to 0x7: number of automatic retransmission attempts (before signaling error)

*Note: If Smartcard mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).*

Bit 16 Reserved, must be kept at reset value.

Bit 15 **DEP**: Driver enable polarity selection

0: DE signal is active high.

1: DE signal is active low.

This bit can only be written when the USART is disabled (UE=0).

*Note: If the Driver Enable feature is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).*

Bit 14 **DEM**: Driver enable mode

This bit allows the user to activate the external transceiver control, through the DE signal.

0: DE function is disabled.

1: DE function is enabled. The DE signal is output on the RTS pin.

This bit can only be written when the USART is disabled (UE=0).

*Note: If the Driver Enable feature is not supported, this bit is reserved and must be kept at reset value. [Section 27.4: USART implementation on page 702](#).*

Bit 13 **DDRE**: DMA Disable on Reception Error

0: DMA is not disabled in case of reception error. The corresponding error flag is set but RXNE is kept 0 preventing from overrun. As a consequence, the DMA request is not asserted, so the erroneous data is not transferred (no DMA request), but next correct received data are transferred (used for Smartcard mode).

1: DMA is disabled following a reception error. The corresponding error flag is set, as well as RXNE. The DMA request is masked until the error flag is cleared. This means that the software must first disable the DMA request (DMAR = 0) or clear RXNE before clearing the error flag.

This bit can only be written when the USART is disabled (UE=0).

*Note: The reception errors are: parity error, framing error or noise error.*

Bit 12 **OVRDIS**: Overrun Disable

This bit is used to disable the receive overrun detection.

0: Overrun Error Flag, ORE, is set when received data is not read before receiving new data.  
 1: Overrun functionality is disabled. If new data is received while the RXNE flag is still set the ORE flag is not set and the new received data overwrites the previous content of the USART\_RDR register.

This bit can only be written when the USART is disabled (UE=0).

*Note: This control bit allows checking the communication flow without reading the data.*

Bit 11 **ONEBIT**: One sample bit method enable

This bit allows the user to select the sample method. When the one sample bit method is selected the noise detection flag (NF) is disabled.

0: Three sample bit method

1: One sample bit method

This bit can only be written when the USART is disabled (UE=0).

*Note: ONEBIT feature applies only to data bits, It does not apply to Start bit.*

Bit 10 **CTSIE**: CTS interrupt enable

0: Interrupt is inhibited

1: An interrupt is generated whenever CTSIF=1 in the USART\_ISR register

*Note: If the hardware flow control feature is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).*

Bit 9 **CTSE**: CTS enable

0: CTS hardware flow control disabled

1: CTS mode enabled, data is only transmitted when the CTS input is deasserted (tied to 0). If the CTS input is asserted while data is being transmitted, then the transmission is completed before stopping. If data is written into the data register while CTS is asserted, the transmission is postponed until CTS is deasserted.

This bit can only be written when the USART is disabled (UE=0)

*Note: If the hardware flow control feature is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).*

Bit 8 **RTSE**: RTS enable

0: RTS hardware flow control disabled

1: RTS output enabled, data is only requested when there is space in the receive buffer. The transmission of data is expected to cease after the current character has been transmitted. The RTS output is deasserted (pulled to 0) when data can be received.

This bit can only be written when the USART is disabled (UE=0).

*Note: If the hardware flow control feature is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).*

Bit 7 **DMAT**: DMA enable transmitter

This bit is set/reset by software

1: DMA mode is enabled for transmission

0: DMA mode is disabled for transmission

Bit 6 **DMAR**: DMA enable receiver

This bit is set/reset by software

1: DMA mode is enabled for reception

0: DMA mode is disabled for reception

**Bit 5 SCEN:** Smartcard mode enable

This bit is used for enabling Smartcard mode.

0: Smartcard Mode disabled

1: Smartcard Mode enabled

This bit field can only be written when the USART is disabled (UE=0).

*Note: If the USART does not support Smartcard mode, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).*

**Bit 4 NACK:** Smartcard NACK enable

0: NACK transmission in case of parity error is disabled

1: NACK transmission during parity error is enabled

This bit field can only be written when the USART is disabled (UE=0).

*Note: If the USART does not support Smartcard mode, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).*

**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

This bit can only be written when the USART is disabled (UE=0).

**Bit 2 IRLP:** IrDA low-power

This bit is used for selecting between normal and low-power IrDA modes

0: Normal mode

1: Low-power mode

This bit can only be written when the USART is disabled (UE=0).

*Note: If IrDA mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).*

**Bit 1 IREN:** IrDA mode enable

This bit is set and cleared by software.

0: IrDA disabled

1: IrDA enabled

This bit can only be written when the USART is disabled (UE=0).

*Note: If IrDA mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).*

**Bit 0 EIE:** Error interrupt enable

Error Interrupt Enable Bit is required to enable interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the USART\_ISR register).

0: Interrupt is inhibited

1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USART\_ISR register.

### 27.8.4 USART baud rate register (USART\_BRR)

This register can only be written when the USART is disabled (UE=0). It may be automatically updated by hardware in auto baud rate detection mode.

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
BRR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 15:4 **BRR[15:4]**

$$\text{BRR}[15:4] = \text{USARTDIV}[15:4]$$

Bits 3:0 **BRR[3:0]**

When OVER8 = 0, BRR[3:0] = USARTDIV[3:0].

When OVER8 = 1:

BRR[2:0] = USARTDIV[3:0] shifted 1 bit to the right.

BRR[3] must be kept cleared.

### 27.8.5 USART guard time and prescaler register (USART\_GTPR)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
GT[7:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 15:8 **GT[7:0]**: Guard time value

This bit-field is used to program the Guard time value in terms of number of baud clock periods.

This is used in Smartcard mode. The Transmission Complete flag is set after this guard time value.

This bit field can only be written when the USART is disabled (UE=0).

*Note: If Smartcard mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).*

Bits 7:0 **PSC[7:0]**: Prescaler value

**In IrDA Low-power and normal IrDA mode:**

PSC[7:0] = IrDA Normal and Low-Power Baud Rate

Used for programming the prescaler for dividing the USART source clock to achieve the low-power frequency:

The source clock is divided by the value given in the register (8 significant bits):

00000000: Reserved - do not program this value

00000001: divides the source clock by 1

00000010: divides the source clock by 2

...

**In Smartcard mode:**

PSC[4:0]: Prescaler value

Used for programming the prescaler for dividing the USART source clock to provide the Smartcard clock.

The value given in the register (5 significant bits) is multiplied by 2 to give the division factor of the source clock frequency:

00000: Reserved - do not program this value

00001: divides the source clock by 2

00010: divides the source clock by 4

00011: divides the source clock by 6

...

This bit field can only be written when the USART is disabled (UE=0).

*Note: Bits [7:5] must be kept at reset value if Smartcard mode is used.*

*This bit field is reserved and must be kept at reset value when the Smartcard and IrDA modes are not supported. Please refer to [Section 27.4: USART implementation on page 702](#).*

## 27.8.6 USART receiver timeout register (USART\_RTOR)

Address offset: 0x14

Reset value: 0x0000 0000

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

**Bits 31:24 BLEN[7:0]: Block Length**

This bit-field gives the Block length in Smartcard T=1 Reception. Its value equals the number of information characters + the length of the Epilogue Field (1-LEC/2-CRC) - 1.

Examples:

BLEN = 0 -> 0 information characters + LEC

BLEN = 1 -> 0 information characters + CRC

BLEN = 255 -> 254 information characters + CRC (total 256 characters))

In Smartcard mode, the Block length counter is reset when TXE=0.

This bit-field can be used also in other modes. In this case, the Block length counter is reset when RE=0 (receiver disabled) and/or when the EOBCF bit is written to 1.

*Note: This value can be programmed after the start of the block reception (using the data from the LEN character in the Prologue Field). It must be programmed only once per received block.*

**Bits 23:0 RTO[23:0]: Receiver timeout value**

This bit-field gives the Receiver timeout value in terms of number of bit duration.

In standard mode, the RTOF flag is set if, after the last received character, no new start bit is detected for more than the RTO value.

In Smartcard mode, this value is used to implement the CWT and BWT. See Smartcard section for more details.

In this case, the timeout measurement is done starting from the Start Bit of the last received character.

*Note: This value must only be programmed once per received character.*

**Note:** *RTOR can be written on the fly. If the new value is lower than or equal to the counter, the RTOF flag is set.*

*This register is reserved and forced by hardware to “0x00000000” when the Receiver timeout feature is not supported. Please refer to [Section 27.4: USART implementation on page 702](#).*

### 27.8.7 USART request register (USART\_RQR)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TXFRQ	RXFRQ	MMRQ	SBKRQ	ABRRQ										
											w	w	w	w	w

Bits 31:5 Reserved, must be kept at reset value.

**Bit 4 TXFRQ:** Transmit data flush request

Writing 1 to this bit sets the TXE flag.

This allows to discard the transmit data. This bit must be used only in Smartcard mode, when data has not been sent due to errors (NACK) and the FE flag is active in the USART\_ISR register.

If the USART does not support Smartcard mode, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).

**Bit 3 RXFRQ:** Receive data flush request

Writing 1 to this bit clears the RXNE flag.

This allows to discard the received data without reading it, and avoid an overrun condition.

**Bit 2 MMRQ:** Mute mode request

Writing 1 to this bit puts the USART in mute mode and sets the RWU flag.

**Bit 1 SBKRQ:** Send break request

Writing 1 to this bit sets the SBKF flag and request to send a BREAK on the line, as soon as the transmit machine is available.

*Note: In the case the application needs to send the break character following all previously inserted data, including the ones not yet transmitted, the software should wait for the TXE flag assertion before setting the SBKRQ bit.*

**Bit 0 ABRRQ:** Auto baud rate request

Writing 1 to this bit resets the ABRF and ABRE flags in the USART\_ISR and request an automatic baud rate measurement on the next received data frame.

*Note: If the USART does not support the auto baud rate feature, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).*

## 27.8.8 USART interrupt and status register (USART\_ISR)

Address offset: 0x1C

Reset value: 0x0200 00C0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	REACK	TEACK	WUF	RWU	SBKF	CMF	BUSY						
									r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABRF	ABRE	Res.	EOBF	RTOF	CTS	CTSIF	LBDF	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
r	r		r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:25 Reserved, must be kept at reset value.

Bits 24:23 Reserved, must be kept at reset value.

Bit 22 **REACK**: Receive enable acknowledge flag

This bit is set/reset by hardware, when the Receive Enable value is taken into account by the USART.

When the wake-up from Stop mode is supported, the REACK flag can be used to verify that the USART is ready for reception before entering Stop mode.

Bit 21 **TEACK**: Transmit enable acknowledge flag

This bit is set/reset by hardware, when the Transmit Enable value is taken into account by the USART.

It can be used when an idle frame request is generated by writing TE=0, followed by TE=1 in the USART\_CR1 register, in order to respect the TE=0 minimum period.

Bit 20 **WUF**: Wake-up from Stop mode flag

This bit is set by hardware, when a wake-up event is detected. The event is defined by the WUS bit field. It is cleared by software, writing a 1 to the WUCF in the USART\_ICR register.

An interrupt is generated if WUFIE=1 in the USART\_CR3 register.

*Note: When UESM is cleared, WUF flag is also cleared.*

*The WUF interrupt is active only in Stop mode.*

*If the USART does not support the wake-up from Stop feature, this bit is reserved and kept at reset value.*

Bit 19 **RWU**: Receiver wake-up from Mute mode

This bit indicates if the USART is in mute mode. It is cleared/set by hardware when a wake-up/mute sequence is recognized. The mute mode control sequence (address or IDLE) is selected by the WAKE bit in the USART\_CR1 register.

When wake-up on IDLE mode is selected, this bit can only be set by software, writing 1 to the MMRQ bit in the USART\_RQR register.

0: Receiver in active mode

1: Receiver in mute mode

Bit 18 **SBKF**: Send break flag

This bit indicates that a send break character was requested. It is set by software, by writing 1 to the SBKRQ bit in the USART\_RQR register. It is automatically reset by hardware during the stop bit of break transmission.

0: No break character is transmitted

1: Break character is transmitted

Bit 17 **CMF**: Character match flag

This bit is set by hardware, when the character defined by ADD[7:0] is received. It is cleared by software, writing 1 to the CMCF in the USART\_ICR register.

An interrupt is generated if CMIE=1 in the USART\_CR1 register.

0: No Character match detected

1: Character Match detected

Bit 16 **BUSY**: Busy flag

This bit is set and reset by hardware. It is active when a communication is ongoing on the RX line (successful start bit detected). It is reset at the end of the reception (successful or not).

0: USART is idle (no reception)

1: Reception on going

Bit 15 **ABRF**: Auto baud rate flag

This bit is set by hardware when the automatic baud rate has been set (RXNE will also be set, generating an interrupt if RXNEIE = 1) or when the auto baud rate operation was completed without success (ABRE=1) (ABRE, RXNE and FE are also set in this case). It is cleared by software, in order to request a new auto baud rate detection, by writing 1 to the ABRRQ in the USART\_RQR register.

*Note: If the USART does not support the auto baud rate feature, this bit is reserved and kept at reset value.*

Bit 14 **ABRE**: Auto baud rate error

This bit is set by hardware if the baud rate measurement failed (baud rate out of range or character comparison failed).

It is cleared by software, by writing 1 to the ABRRQ bit in the USART\_RQR register.

*Note: If the USART does not support the auto baud rate feature, this bit is reserved and kept at reset value.*

## Bit 13 Reserved, must be kept at reset value.

Bit 12 **EOBF**: End of block flag

This bit is set by hardware when a complete block has been received (for example T=1 Smartcard mode). The detection is done when the number of received bytes (from the start of the block, including the prologue) is equal or greater than BLEN + 4.

An interrupt is generated if EOBIE = 1 in the USART\_CR1 register.

It is cleared by software, writing 1 to EOBCF in the USART\_ICR register.

0: End of Block not reached

1: End of Block (number of characters) reached

*Note: If Smartcard mode is not supported, this bit is reserved and kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).*

Bit 11 **RTOF**: Receiver timeout

This bit is set by hardware when the timeout value, programmed in the RTOR register has lapsed, without any communication. It is cleared by software, writing 1 to the RTOCF bit in the USART\_ICR register.

An interrupt is generated if RTOIE=1 in the USART\_CR1 register.

In Smartcard mode, the timeout corresponds to the CWT or BWT timings.

0: Timeout value not reached

1: Timeout value reached without any data reception

*Note: If a time equal to the value programmed in RTOR register separates 2 characters, RTOF is not set. If this time exceeds this value + 2 sample times (2/16 or 2/8, depending on the oversampling method), RTOF flag is set.*

*The counter counts even if RE = 0 but RTOF is set only when RE = 1. If the timeout has already elapsed when RE is set, then RTOF is set.*

*If the USART does not support the Receiver timeout feature, this bit is reserved and kept at reset value.*

Bit 10 **CTS**: CTS flag

This bit is set/reset by hardware. It is an inverted copy of the status of the CTS input pin.

0: CTS line set

1: CTS line reset

*Note: If the hardware flow control feature is not supported, this bit is reserved and kept at reset value.*

Bit 9 **CTSIF**: CTS interrupt flag

This bit is set by hardware when the CTS input toggles, if the CTSE bit is set. It is cleared by software, by writing 1 to the CTSCF bit in the USART\_ICR register.

An interrupt is generated if CTSIE=1 in the USART\_CR3 register.

0: No change occurred on the CTS status line

1: A change occurred on the CTS status line

*Note: If the hardware flow control feature is not supported, this bit is reserved and kept at reset value.*

Bit 8 **LBDF**: LIN break detection flag

This bit is set by hardware when the LIN break is detected. It is cleared by software, by writing 1 to the LBDCF in the USART\_ICR.

An interrupt is generated if LBDIE = 1 in the USART\_CR2 register.

0: LIN Break not detected

1: LIN break detected

*Note: If the USART does not support LIN mode, this bit is reserved and kept at reset value.  
Please refer to [Section 27.4: USART implementation on page 702](#).*

Bit 7 **TXE**: Transmit data register empty

This bit is set by hardware when the content of the USART\_TDR register has been transferred into the shift register. It is cleared by a write to the USART\_TDR register.

The TXE flag can also be cleared by writing 1 to the TXFRQ in the USART\_RQR register, in order to discard the data (only in Smartcard T=0 mode, in case of transmission failure).

An interrupt is generated if the TXIE bit =1 in the USART\_CR1 register.

0: data is not transferred to the shift register

1: data is transferred to the shift register)

*Note: This bit is used during single buffer transmission.*

Bit 6 **TC**: Transmission complete

This bit is set by hardware if the transmission of a frame containing data is complete and if TXE is set. An interrupt is generated if TCIE=1 in the USART\_CR1 register. It is cleared by software, writing 1 to the TCCF in the USART\_ICR register or by a write to the USART\_TDR register.

An interrupt is generated if TCIE=1 in the USART\_CR1 register.

0: Transmission is not complete

1: Transmission is complete

*Note: If TE bit is reset and no transmission is on going, the TC bit is set immediately.*

Bit 5 **RXNE**: Read data register not empty

This bit is set by hardware when the content of the RDR shift register has been transferred to the USART\_RDR register. It is cleared by a read to the USART\_RDR register. The RXNE flag can also be cleared by writing 1 to the RXFRQ in the USART\_RQR register.

An interrupt is generated if RXNEIE=1 in the USART\_CR1 register.

0: data is not received

1: Received data is ready to be read.

**Bit 4 IDLE:** Idle line detected

This bit is set by hardware when an Idle Line is detected. An interrupt is generated if IDLEIE=1 in the USART\_CR1 register. It is cleared by software, writing 1 to the IDLECF in the USART\_ICR register.

- 0: No Idle line is detected
- 1: Idle line is detected

*Note: The IDLE bit will not be set again until the RXNE bit has been set (i.e. a new idle line occurs).*

*If mute mode is enabled (MME=1), IDLE is set if the USART is not mute (RWU=0), whatever the mute mode selected by the WAKE bit. If RWU=1, IDLE is not set.*

**Bit 3 ORE:** Overrun error

This bit is set by hardware when the data currently being received in the shift register is ready to be transferred into the RDR register while RXNE=1. It is cleared by a software, writing 1 to the ORECF, in the USART\_ICR register.

An interrupt is generated if RXNEIE=1 in the USART\_CR1 register, or EIE = 1 in the USART\_CR3 register.

- 0: No overrun error
- 1: Overrun error is detected

*Note: When this bit is set, the RDR register content is not lost but the shift register is overwritten. An interrupt is generated if the ORE flag is set during multibuffer communication if the EIE bit is set.*

*This bit is permanently forced to 0 (no overrun detection) when the OVRDIS bit is set in the USART\_CR3 register.*

**Bit 2 NF:** START bit Noise detection flag

This bit is set by hardware when noise is detected on a received frame. It is cleared by software, writing 1 to the NFCF bit in the USART\_ICR register.

- 0: No noise is detected
- 1: Noise is detected

*Note: This bit does not generate an interrupt as it appears at the same time as the RXNE bit which itself generates an interrupt. An interrupt is generated when the NF flag is set during multibuffer communication if the EIE bit is set.*

*Note: When the line is noise-free, the NF flag can be disabled by programming the ONEBIT bit to 1 to increase the USART tolerance to deviations (Refer to [Section 27.5.5: Tolerance of the USART receiver to clock deviation on page 718](#)).*

**Bit 1 FE:** Framing error

This bit is set by hardware when a de-synchronization, excessive noise or a break character is detected. It is cleared by software, writing 1 to the FECF bit in the USART\_ICR register.

In Smartcard mode, in transmission, this bit is set when the maximum number of transmit attempts is reached without success (the card NACKs the data frame).

An interrupt is generated if EIE = 1 in the USART\_CR3 register.

- 0: No Framing error is detected
- 1: Framing error or break character is detected

**Bit 0 PE:** Parity error

This bit is set by hardware when a parity error occurs in receiver mode. It is cleared by software, writing 1 to the PECF in the USART\_ICR register.

An interrupt is generated if PEIE = 1 in the USART\_CR1 register.

- 0: No parity error
- 1: Parity error

### 27.8.9 USART interrupt flag clear register (USART\_ICR)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	WUCF	Res.	Res.	CMCF	Res.
											rc_w1			rc_w1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	EOBCF	RTOCF	Res.	CTSCF	LBDCF	Res.	TCCF	Res.	IDLECF	ORECF	NCF	FECF	PECF
			rc_w1	rc_w1		rc_w1	rc_w1		rc_w1		rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:21 Reserved, must be kept at reset value.

Bit 20 **WUCF**: Wake-up from Stop mode clear flag

Writing 1 to this bit clears the WUF flag in the USART\_ISR register.

*Note: If the USART does not support the wake-up from Stop feature, this bit is reserved and must be kept at reset value.*

Bits 19:18 Reserved, must be kept at reset value.

Bit 17 **CMCF**: Character match clear flag

Writing 1 to this bit clears the CMF flag in the USART\_ISR register.

Bits 16:13 Reserved, must be kept at reset value.

Bit 12 **EOBCF**: End of block clear flag

Writing 1 to this bit clears the EOBF flag in the USART\_ISR register.

*Note: If the USART does not support Smartcard mode, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).*

Bit 11 **RTOCF**: Receiver timeout clear flag

Writing 1 to this bit clears the RTOF flag in the USART\_ISR register.

*Note: If the USART does not support the Receiver timeout feature, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).*

Bit 10 Reserved, must be kept at reset value.

Bit 9 **CTSCF**: CTS clear flag

Writing 1 to this bit clears the CTSIF flag in the USART\_ISR register.

*Note: If the hardware flow control feature is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).*

Bit 8 **LBDCF**: LIN break detection clear flag

Writing 1 to this bit clears the LBDF flag in the USART\_ISR register.

*Note: If LIN mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 702](#).*

Bit 7 Reserved, must be kept at reset value.

Bit 6 **TCCF**: Transmission complete clear flag

Writing 1 to this bit clears the TC flag in the USART\_ISR register.

Bit 5 Reserved, must be kept at reset value.

Bit 4 **IDLECF**: Idle line detected clear flag

Writing 1 to this bit clears the IDLE flag in the USART\_ISR register.

- Bit 3 **ORECF**: Overrun error clear flag  
Writing 1 to this bit clears the ORE flag in the USART\_ISR register.
- Bit 2 **NCF**: Noise detected clear flag  
Writing 1 to this bit clears the NF flag in the USART\_ISR register.
- Bit 1 **FECF**: Framing error clear flag  
Writing 1 to this bit clears the FE flag in the USART\_ISR register.
- Bit 0 **PECF**: Parity error clear flag  
Writing 1 to this bit clears the PE flag in the USART\_ISR register.

### 27.8.10 USART receive data register (USART\_RDR)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	RDR[8:0]														
								r	r	r	r	r	r	r	r

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

Bits 8:0 **RDR[8:0]**: Receive data value

Contains the received data character.

The RDR register provides the parallel interface between the input shift register and the internal bus (see [Figure 248](#)).

When receiving with the parity enabled, the value read in the MSB bit is the received parity bit.

### 27.8.11 USART transmit data register (USART\_TDR)

Address offset: 0x28

Reset value: 0x0000 0000

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

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

Bits 8:0 **TDR[8:0]**: Transmit data value

Contains the data character to be transmitted.

The TDR register provides the parallel interface between the internal bus and the output shift register (see *Figure 248*).

When transmitting with the parity enabled (PCE bit set to 1 in the USART\_CR1 register), the value written in the MSB (bit 7 or bit 8 depending on the data length) has no effect because it is replaced by the parity.

*Note:* This register must be written only when TXE=1.

## 27.8.12 USART register map

The table below gives the USART register map and reset values.

**Table 112. USART register map and reset values**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0							
0x00	<b>USART_CR1</b>	Res.	Res.	Res.	M1	EOBIE	RTOIE	DEAT4	DEAT3	DEAT2	DEAT1	DEAT0	DEAT4	MSBFIRST	DATAINV	TXINV	DEDT0	OVER8	CMIIE	MME	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
0x04	<b>USART_CR2</b>	ADD[7:4]				ADD[3:0]				RTEN	ABRMOD1	ABRMOD0	ABRMOD0	ABREN	MSBFIRST	DATAINV	TXINV	DEDT3	SWAP	LINEN	STOP[1:0]	DDRE	DEM	DEM	DEM	DEM	DEM	DEM												
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0						
0x08	<b>USART_CR3</b>	Res.	Res.	Res.	Res.	Res.	WUFIE	WUS	SCARCNT2:0	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DEP	SWAP	CMIE	MME	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0						
0x0C	<b>USART_BRR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.						
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0					
0x10	<b>USART_GTPR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.						
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0					
0x14	<b>USART_RTOR</b>	BLEN[7:0]				GT[7:0] PSC[7:0]																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0					
0x18	<b>USART_RQR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.					
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				

Table 112. USART register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
0x1C	<b>USART_ISR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		Reset value	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
0x20	<b>USART_ICR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		Reset value	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
0x24	<b>USART_RDR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		Reset value	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
0x28	<b>USART_TDR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		Reset value	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 28 Serial peripheral interface / integrated interchip sound (SPI/I<sup>2</sup>S)

### 28.1 Introduction

The SPI/I<sup>2</sup>S interface can be used to communicate with external devices using the SPI protocol or the I<sup>2</sup>S audio protocol. SPI or I<sup>2</sup>S mode is selectable by software. SPI Motorola mode is selected by default after a device reset.

The serial peripheral interface (SPI) protocol supports half-duplex, full-duplex and simplex synchronous, serial communication with external devices. The interface can be configured as master and in this case it provides the communication clock (SCK) to the external slave device. The interface is also capable of operating in multimaster configuration.

The integrated interchip sound (I<sup>2</sup>S) protocol is also a synchronous serial communication interface. It can operate in slave or master mode with half-duplex communication. Full-duplex operations are possible by combining two I<sup>2</sup>S blocks. It can address four different audio standards including the Philips I<sup>2</sup>S standard, the MSB- and LSB-justified standards and the PCM standard.

### 28.2 SPI main features

- Master or slave operation
- Full-duplex synchronous transfers on three lines
- Half-duplex synchronous transfer on two lines (with bidirectional data line)
- Simplex synchronous transfers on two lines (with unidirectional data line)
- 4 to 16-bit data size selection
- Multimaster mode capability
- 8 master mode baud rate prescalers up to  $f_{PCLK}/2$
- Slave mode frequency up to  $f_{PCLK}/2$ .
- NSS management by hardware or software for both master and slave: dynamic change of master/slave operations
- Programmable clock polarity and phase
- Programmable data order with MSB-first or LSB-first shifting
- Dedicated transmission and reception flags with interrupt capability
- SPI bus busy status flag
- SPI Motorola support
- Hardware CRC feature for reliable communication:
  - CRC value can be transmitted as last byte in Tx mode
  - Automatic CRC error checking for last received byte
- Master mode fault, overrun flags with interrupt capability
- CRC Error flag
- Two 32-bit embedded Rx and Tx FIFOs with DMA capability
- Enhanced TI and NSS pulse modes support

## 28.3 I2S main features

- Half-duplex communication (only transmitter or receiver)
- Master or slave operations
- 8-bit programmable linear prescaler to reach accurate audio sample frequencies (from 8 kHz to 192 kHz)
- Data format may be 16-bit, 24-bit or 32-bit
- Packet frame is fixed to 16-bit (16-bit data frame) or 32-bit (16-bit, 24-bit, 32-bit data frame) by audio channel
- Programmable clock polarity (steady state)
- Underrun flag in slave transmission mode, overrun flag in reception mode (master and slave) and Frame Error Flag in reception and transmitter mode (slave only)
- 16-bit register for transmission and reception with one data register for both channel sides
- Supported I<sup>2</sup>S protocols:
  - I<sup>2</sup>S Philips standard
  - MSB-justified standard (left-justified)
  - LSB-justified standard (right-justified)
  - PCM standard (with short and long frame synchronization on 16-bit channel frame or 16-bit data frame extended to 32-bit channel frame)
- Data direction is always MSB first
- DMA capability for transmission and reception (16-bit wide)
- Master clock can be output to drive an external audio component. The ratio is fixed at  $256 \times f_s$  for all I2S modes, and to  $128 \times f_s$  for all PCM modes (where  $f_s$  is the audio sampling frequency).

## 28.4 SPI/I2S implementation

The following table describes all the SPI instances and their features embedded in the devices.

**Table 113. STM32F0xx SPI and SPI/I2S implementation**

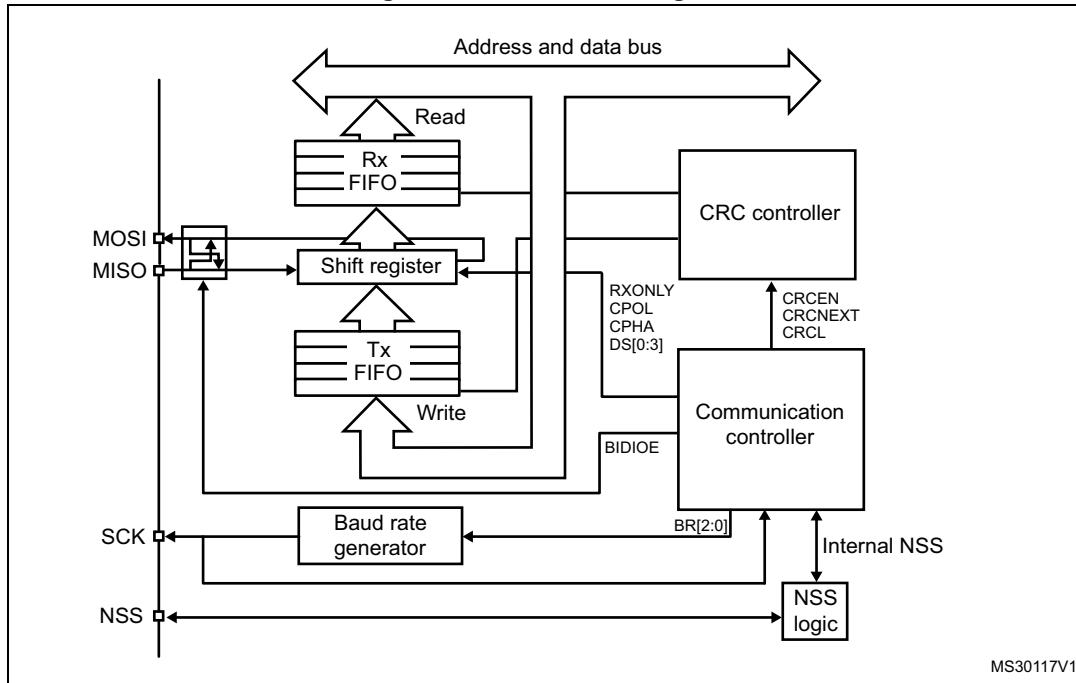
SPI Features	STM32F03x	STM32F04x STM32F05x		STM32F07x STM32F09x	
	SPI2S1	SPI2S1	SPI2	SPI2S1	SPI2S2
Enhanced NSSP & TI modes	Yes	Yes	Yes	Yes	Yes
Hardware CRC calculation	Yes	Yes	Yes	Yes	Yes
Data size configurable	from 4 to 16-bit	from 4 to 16-bit	from 4 to 16-bit	from 4 to 16-bit	from 4 to 16-bit
Rx/Tx FIFO size	32-bit	32-bit	32-bit	32-bit	32-bit
I <sup>2</sup> S support	Yes	Yes	No	Yes	Yes
Wake-up capability from Low-power Sleep	Yes	Yes	Yes	Yes	Yes

## 28.5 SPI functional description

### 28.5.1 General description

The SPI allows synchronous, serial communication between the MCU and external devices. Application software can manage the communication by polling the status flag or using dedicated SPI interrupt. The main elements of SPI and their interactions are shown in the following block diagram [Figure 273](#).

**Figure 273. SPI block diagram**



MS30117V1

Four I/O pins are dedicated to SPI communication with external devices.

- **MISO:** Master In / Slave Out data. In the general case, this pin is used to transmit data in slave mode and receive data in master mode.
- **MOSI:** Master Out / Slave In data. In the general case, this pin is used to transmit data in master mode and receive data in slave mode.
- **SCK:** Serial Clock output pin for SPI masters and input pin for SPI slaves.
- **NSS:** Slave select pin. Depending on the SPI and NSS settings, this pin can be used to either:
  - select an individual slave device for communication
  - synchronize the data frame or
  - detect a conflict between multiple masters

See [Section 28.5.5: Slave select \(NSS\) pin management](#) for details.

The SPI bus allows the communication between one master device and one or more slave devices. The bus consists of at least two wires - one for the clock signal and the other for synchronous data transfer. Other signals can be added depending on the data exchange between SPI nodes and their slave select signal management.

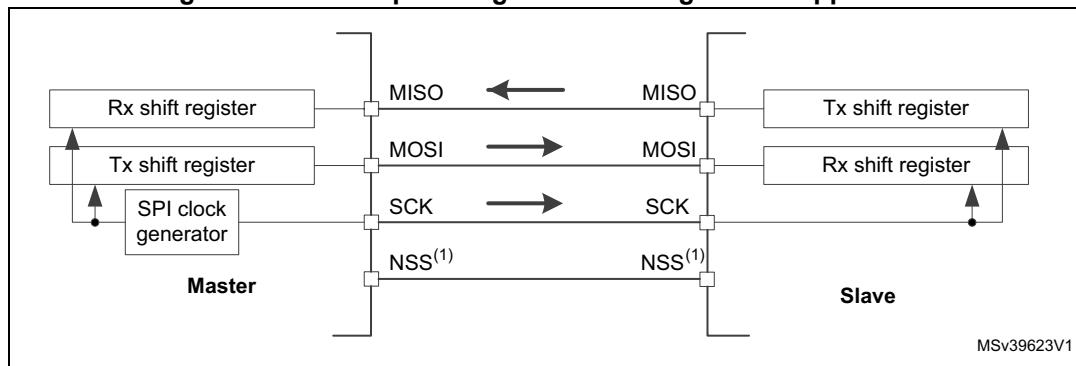
## 28.5.2 Communications between one master and one slave

The SPI allows the MCU to communicate using different configurations, depending on the device targeted and the application requirements. These configurations use 2 or 3 wires (with software NSS management) or 3 or 4 wires (with hardware NSS management). Communication is always initiated by the master.

### Full-duplex communication

By default, the SPI is configured for full-duplex communication. In this configuration, the shift registers of the master and slave are linked using two unidirectional lines between the MOSI and the MISO pins. During SPI communication, data is shifted synchronously on the SCK clock edges provided by the master. The master transmits the data to be sent to the slave via the MOSI line and receives data from the slave via the MISO line. When the data frame transfer is complete (all the bits are shifted) the information between the master and slave is exchanged.

**Figure 274. Full-duplex single master/ single slave application**

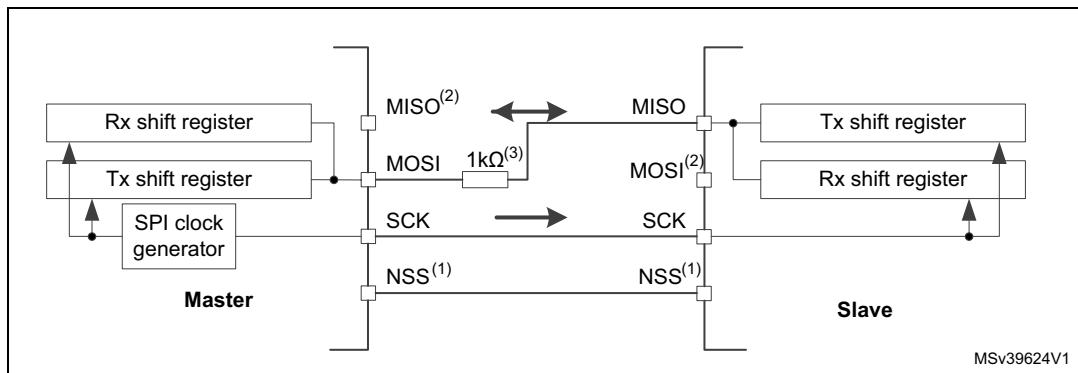


1. The NSS pins can be used to provide a hardware control flow between master and slave. Optionally, the pins can be left unused by the peripheral. Then the flow has to be handled internally for both master and slave. For more details see [Section 28.5.5: Slave select \(NSS\) pin management](#).

### Half-duplex communication

The SPI can communicate in half-duplex mode by setting the BIDIMODE bit in the SPIx\_CR1 register. In this configuration, one single cross connection line is used to link the shift registers of the master and slave together. During this communication, the data is synchronously shifted between the shift registers on the SCK clock edge in the transfer direction selected reciprocally by both master and slave with the BDIOE bit in their SPIx\_CR1 registers. In this configuration, the master's MISO pin and the slave's MOSI pin are free for other application uses and act as GPIOs.

Figure 275. Half-duplex single master/ single slave application



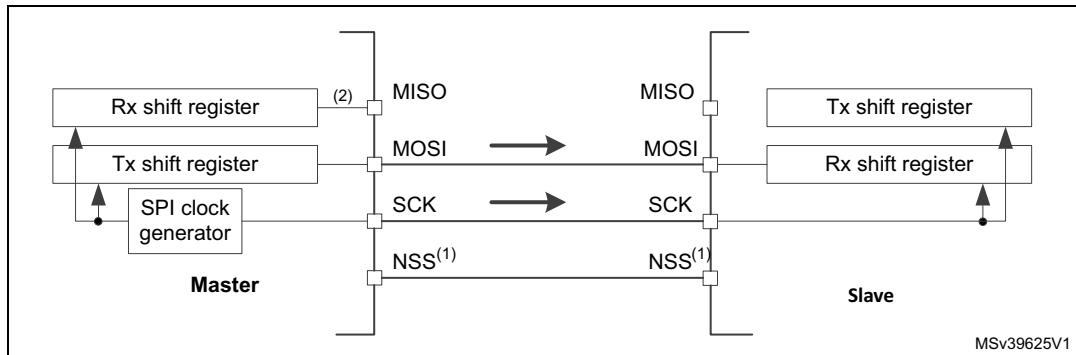
1. The NSS pins can be used to provide a hardware control flow between master and slave. Optionally, the pins can be left unused by the peripheral. Then the flow has to be handled internally for both master and slave. For more details see [Section 28.5.5: Slave select \(NSS\) pin management](#).
2. In this configuration, the master's MISO pin and the slave's MOSI pin can be used as GPIOs.
3. A critical situation can happen when communication direction is changed not synchronously between two nodes working at bidirectional mode and new transmitter accesses the common data line while former transmitter still keeps an opposite value on the line (the value depends on SPI configuration and communication data). Both nodes then fight while providing opposite output levels on the common line temporary till next node changes its direction settings correspondingly, too. It is suggested to insert a serial resistance between MISO and MOSI pins at this mode to protect the outputs and limit the current blowing between them at this situation.

### Simplex communications

The SPI can communicate in simplex mode by setting the SPI in transmit-only or in receive-only using the RXONLY bit in the SPIx\_CR1 register. In this configuration, only one line is used for the transfer between the shift registers of the master and slave. The remaining MISO and MOSI pins pair is not used for communication and can be used as standard GPIOs.

- **Transmit-only mode (RXONLY=0):** The configuration settings are the same as for full-duplex. The application has to ignore the information captured on the unused input pin. This pin can be used as a standard GPIO.
- **Receive-only mode (RXONLY=1):** The application can disable the SPI output function by setting the RXONLY bit. In slave configuration, the MISO output is disabled and the pin can be used as a GPIO. The slave continues to receive data from the MOSI pin while its slave select signal is active (see [28.5.5: Slave select \(NSS\) pin management](#)). Received data events appear depending on the data buffer configuration. In the master configuration, the MOSI output is disabled and the pin can be used as a GPIO. The clock signal is generated continuously as long as the SPI is enabled. The only way to stop the clock is to clear the RXONLY bit or the SPE bit and wait until the incoming pattern from the MISO pin is finished and fills the data buffer structure, depending on its configuration.

**Figure 276. Simplex single master/single slave application (master in transmit-only/slave in receive-only mode)**



1. The NSS pins can be used to provide a hardware control flow between master and slave. Optionally, the pins can be left unused by the peripheral. Then the flow has to be handled internally for both master and slave. For more details see [Section 28.5.5: Slave select \(NSS\) pin management](#).
2. An accidental input information is captured at the input of transmitter Rx shift register. All the events associated with the transmitter receive flow must be ignored in standard transmit only mode (e.g. OVR flag).
3. In this configuration, both the MISO pins can be used as GPIOs.

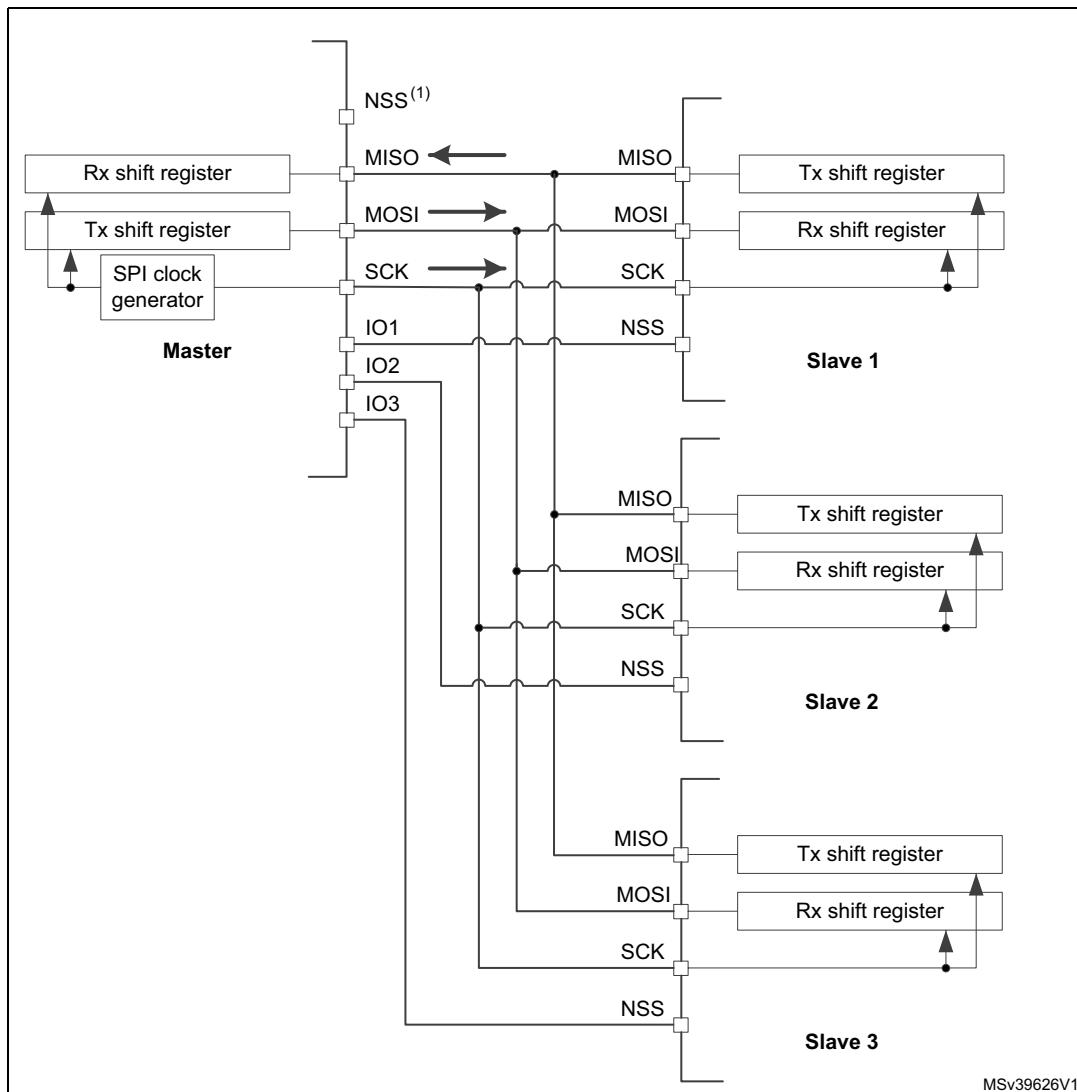
**Note:**

*Any simplex communication can be alternatively replaced by a variant of the half-duplex communication with a constant setting of the transaction direction (bidirectional mode is enabled while BDIO bit is not changed).*

### 28.5.3 Standard multi-slave communication

In a configuration with two or more independent slaves, the master uses GPIO pins to manage the chip select lines for each slave (see [Figure 277](#)). The master must select one of the slaves individually by pulling low the GPIO connected to the slave NSS input. When this is done, a standard master and dedicated slave communication is established.

Figure 277. Master and three independent slaves



1. NSS pin is not used on master side at this configuration. It has to be managed internally ( $SSM=1$ ,  $SSI=1$ ) to prevent any MODF error.
2. As MISO pins of the slaves are connected together, all slaves must have the GPIO configuration of their MISO pin set as alternate function open-drain (see I/O alternate function input/output section (GPIO)).

#### 28.5.4 Multi-master communication

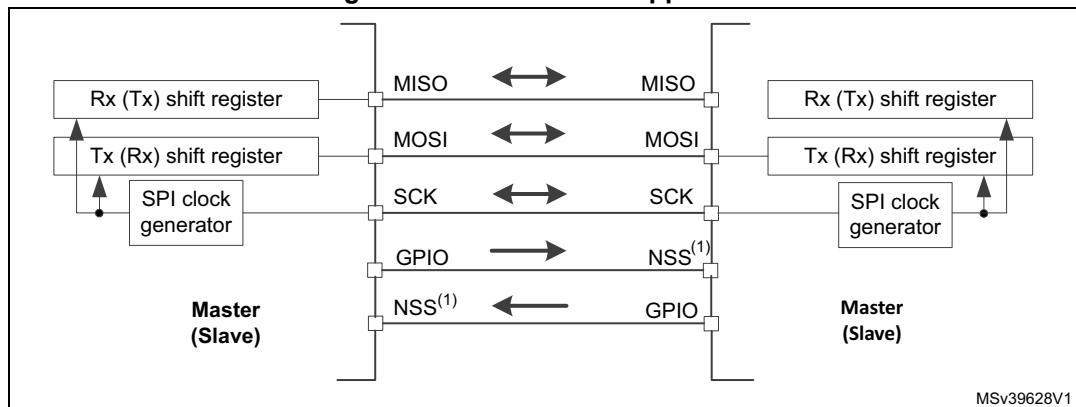
Unless SPI bus is not designed for a multi-master capability primarily, the user can use build in feature which detects a potential conflict between two nodes trying to master the bus at the same time. For this detection, NSS pin is used configured at hardware input mode.

The connection of more than two SPI nodes working at this mode is impossible as only one node can apply its output on a common data line at time.

When nodes are non active, both stay at slave mode by default. Once one node wants to overtake control on the bus, it switches itself into master mode and applies active level on the slave select input of the other node via dedicated GPIO pin. After the session is completed, the active slave select signal is released and the node mastering the bus temporary returns back to passive slave mode waiting for next session start.

If potentially both nodes raised their mastering request at the same time a bus conflict event appears (see mode fault MODF event). Then the user can apply some simple arbitration process (e.g. to postpone next attempt by predefined different time-outs applied at both nodes).

**Figure 278. Multi-master application**



1. The NSS pin is configured at hardware input mode at both nodes. Its active level enables the MISO line output control as the passive node is configured as a slave.

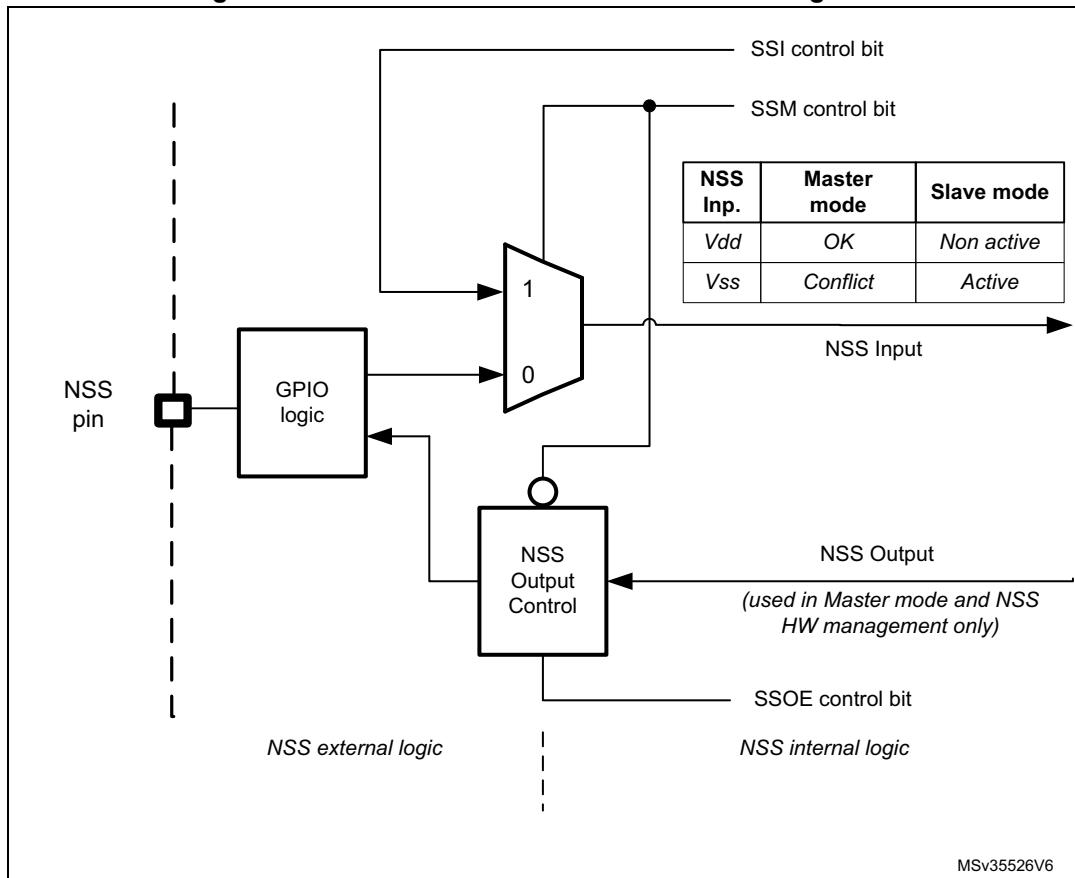
### 28.5.5 Slave select (NSS) pin management

In slave mode, the NSS works as a standard “chip select” input and lets the slave communicate with the master. In master mode, NSS can be used either as output or input. As an input it can prevent multimaster bus collision, and as an output it can drive a slave select signal of a single slave.

Hardware or software slave select management can be set using the SSM bit in the SPIx\_CR1 register:

- **Software NSS management (SSM = 1):** in this configuration, slave select information is driven internally by the SSI bit value in register SPIx\_CR1. The external NSS pin is free for other application uses.
- **Hardware NSS management (SSM = 0):** in this case, there are two possible configurations. The configuration used depends on the NSS output configuration (SSOE bit in register SPIx\_CR1).
  - **NSS output enable (SSM=0,SSOE = 1):** this configuration is only used when the MCU is set as master. The NSS pin is managed by the hardware. The NSS signal is driven low as soon as the SPI is enabled in master mode (SPE=1), and is kept low until the SPI is disabled (SPE =0). A pulse can be generated between continuous communications if NSS pulse mode is activated (NSSP=1). The SPI cannot work in multimaster configuration with this NSS setting.
  - **NSS output disable (SSM=0, SSOE = 0):** if the microcontroller is acting as the master on the bus, this configuration allows multimaster capability. If the NSS pin is pulled low in this mode, the SPI enters master mode fault state and the device is automatically reconfigured in slave mode. In slave mode, the NSS pin works as a standard “chip select” input and the slave is selected while NSS line is at low level.

Figure 279. Hardware/software slave select management



## 28.5.6 Communication formats

During SPI communication, receive and transmit operations are performed simultaneously. The serial clock (SCK) synchronizes the shifting and sampling of the information on the data lines. The communication format depends on the clock phase, the clock polarity and the data frame format. To be able to communicate together, the master and slaves devices must follow the same communication format.

### Clock phase and polarity controls

Four possible timing relationships may be chosen by software, using the CPOL and CPHA bits in the SPIx\_CR1 register. The CPOL (clock polarity) bit controls the idle state value of the clock when no data is being transferred. This bit affects both master and slave modes. If CPOL is reset, the SCK pin has a low-level idle state. If CPOL is set, the SCK pin has a high-level idle state.

If the CPHA bit is set, the second edge on the SCK pin captures the first data bit transacted (falling edge if the CPOL bit is reset, rising edge if the CPOL bit is set). Data are latched on each occurrence of this clock transition type. If the CPHA bit is reset, the first edge on the SCK pin captures the first data bit transacted (falling edge if the CPOL bit is set, rising edge if the CPOL bit is reset). Data are latched on each occurrence of this clock transition type.

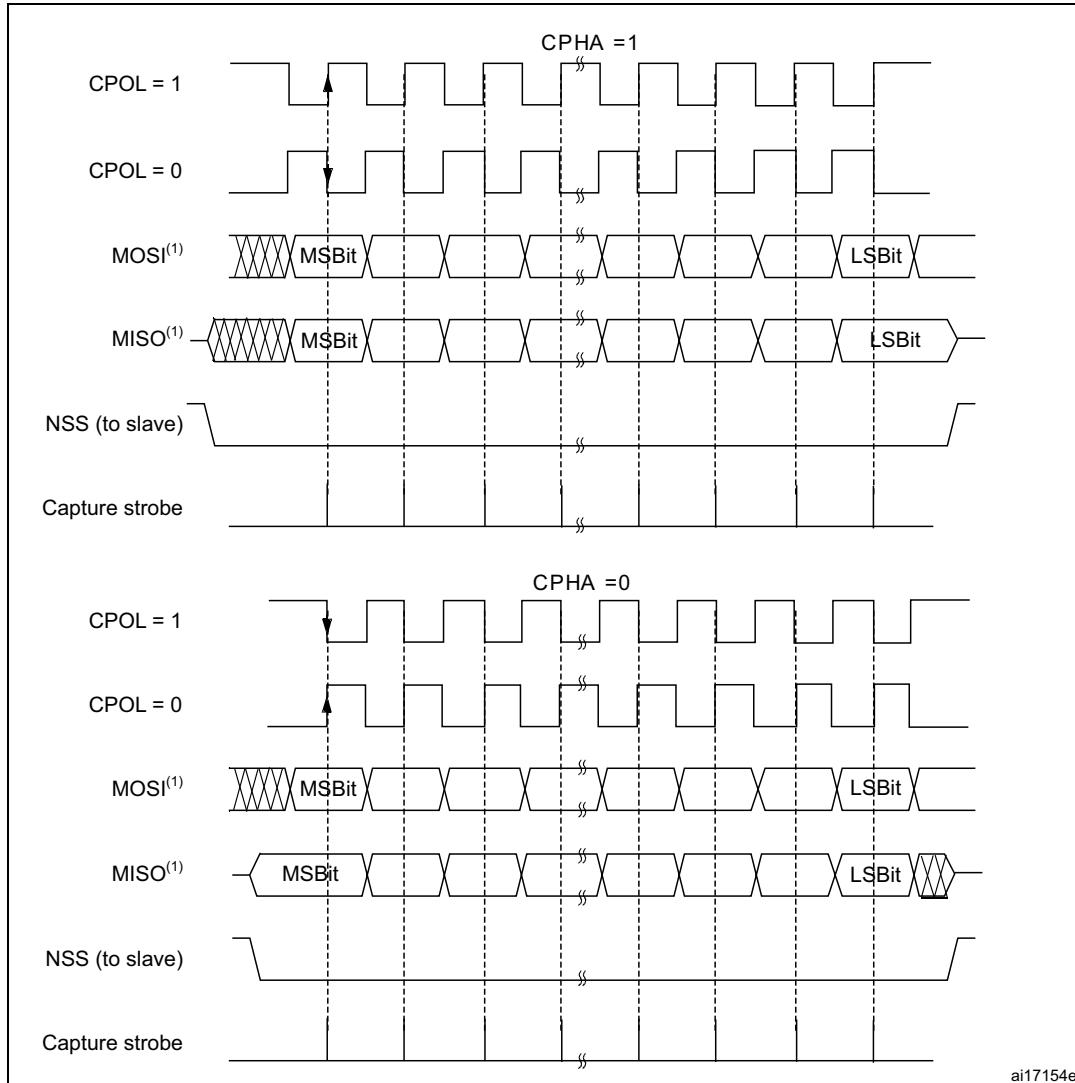
The combination of CPOL (clock polarity) and CPHA (clock phase) bits selects the data capture clock edge.

**Figure 280**, shows an SPI full-duplex transfer with the four combinations of the CPHA and CPOL bits.

**Note:** Prior to changing the CPOL/CPHA bits the SPI must be disabled by resetting the SPE bit.

The idle state of SCK must correspond to the polarity selected in the SPIx\_CR1 register (by pulling up SCK if CPOL=1 or pulling down SCK if CPOL=0).

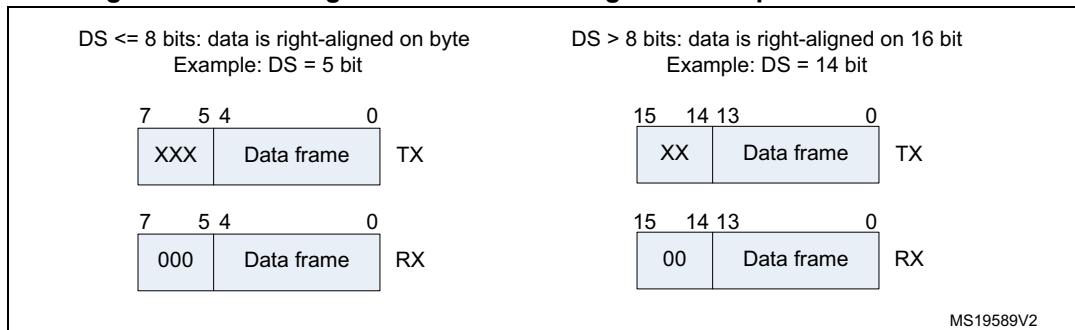
**Figure 280. Data clock timing diagram**



1. The order of data bits depends on LSBFIRST bit setting.

### Data frame format

The SPI shift register can be set up to shift out MSB-first or LSB-first, depending on the value of the LSBFIRST bit. The data frame size is chosen by using the DS bits. It can be set from 4-bit up to 16-bit length and the setting applies for both transmission and reception. Whatever the selected data frame size, read access to the FIFO must be aligned with the FRXTH level. When the SPIx\_DR register is accessed, data frames are always right-aligned into either a byte (if the data fits into a byte) or a half-word (see **Figure 281**). During communication, only bits within the data frame are clocked and transferred.

**Figure 281. Data alignment when data length is not equal to 8-bit or 16-bit**

**Note:** The minimum data length is 4 bits. If a data length of less than 4 bits is selected, it is forced to an 8-bit data frame size.

### 28.5.7 Configuration of SPI

The configuration procedure is almost the same for master and slave. For specific mode setups, follow the dedicated sections. When a standard communication is to be initialized, perform these steps:

1. Write proper GPIO registers: Configure GPIO for MOSI, MISO and SCK pins.
2. Write to the SPI\_CR1 register:
  - a) Configure the serial clock baud rate using the BR[2:0] bits (Note: 4).
  - b) Configure the CPOL and CPHA bits combination to define one of the four relationships between the data transfer and the serial clock (CPHA must be cleared in NSSP mode). (Note: 2 - except the case when CRC is enabled at TI mode).
  - c) Select simplex or half-duplex mode by configuring RXONLY or BIDIMODE and BIDIOE (RXONLY and BIDIMODE cannot be set at the same time).
  - d) Configure the LSBFIRST bit to define the frame format (Note: 2).
  - e) Configure the CRCL and CRCEN bits if CRC is needed (while SCK clock signal is at idle state).
  - f) Configure SSM and SSI (Notes: 2 & 3).
  - g) Configure the MSTR bit (in multimaster NSS configuration, avoid conflict state on NSS if master is configured to prevent MODF error).
3. Write to SPI\_CR2 register:
  - a) Configure the DS[3:0] bits to select the data length for the transfer.
  - b) Configure SSOE (Notes: 1 & 2 & 3).
  - c) Set the FRF bit if the TI protocol is required (keep NSSP bit cleared in TI mode).
  - d) Set the NSSP bit if the NSS pulse mode between two data units is required (keep CHPA and TI bits cleared in NSSP mode).
  - e) Configure the RXTH bit. The RXFIFO threshold must be aligned to the read access size for the SPIx\_DR register.
  - f) Initialize LDMA\_TX and LDMA\_RX bits if DMA is used in packed mode.
4. Write to SPI\_CRCPR register: Configure the CRC polynomial if needed.
5. Write proper DMA registers: Configure DMA streams dedicated for SPI Tx and Rx in DMA registers if the DMA streams are used.

- Note:
- (1) Step is not required in slave mode.
  - (2) Step is not required in TI mode.
  - (3) Step is not required in NSSP mode.
  - (4) The step is not required in slave mode except slave working at TI mode

For code example refer to the Appendix sections [A.17.1: SPI master configuration code example](#) and [A.17.2: SPI slave configuration code example](#).

## 28.5.8 Procedure for enabling SPI

It is recommended to enable the SPI slave before the master sends the clock. If not, undesired data transmission might occur. The data register of the slave must already contain data to be sent before starting communication with the master (either on the first edge of the communication clock, or before the end of the ongoing communication if the clock signal is continuous). The SCK signal must be settled at an idle state level corresponding to the selected polarity before the SPI slave is enabled.

The master at full-duplex (or in any transmit-only mode) starts to communicate when the SPI is enabled and TXFIFO is not empty, or with the next write to TXFIFO.

In any master receive only mode (RXONLY=1 or BIDIMODE=1 & BIDIOE=0), master starts to communicate and the clock starts running immediately after SPI is enabled.

For handling DMA, follow the dedicated section.

## 28.5.9 Data transmission and reception procedures

### RXFIFO and TXFIFO

All SPI data transactions pass through the 32-bit embedded FIFOs. This enables the SPI to work in a continuous flow, and prevents overruns when the data frame size is short. Each direction has its own FIFO called TXFIFO and RXFIFO. These FIFOs are used in all SPI modes except for receiver-only mode (slave or master) with CRC calculation enabled (see [Section 28.5.14: CRC calculation](#)).

The handling of FIFOs depends on the data exchange mode (duplex, simplex), data frame format (number of bits in the frame), access size performed on the FIFO data registers (8-bit or 16-bit), and whether or not data packing is used when accessing the FIFOs (see [Section 28.5.13: TI mode](#)).

A read access to the SPIx\_DR register returns the oldest value stored in RXFIFO that has not been read yet. A write access to the SPIx\_DR stores the written data in the TXFIFO at the end of a send queue. The read access must be always aligned with the RXFIFO threshold configured by the FRXTH bit in SPIx\_CR2 register. FTLVL[1:0] and FRLVL[1:0] bits indicate the current occupancy level of both FIFOs.

A read access to the SPIx\_DR register must be managed by the RXNE event. This event is triggered when data is stored in RXFIFO and the threshold (defined by FRXTH bit) is reached. When RXNE is cleared, RXFIFO is considered to be empty. In a similar way, write access of a data frame to be transmitted is managed by the TXE event. This event is triggered when the TXFIFO level is less than or equal to half of its capacity. Otherwise TXE is cleared and the TXFIFO is considered as full. In this way, RXFIFO can store up to four data frames, whereas TXFIFO can only store up to three when the data frame format is not greater than 8 bits. This difference prevents possible corruption of 3x 8-bit data frames already stored in the TXFIFO when software tries to write more data in 16-bit mode into

TXFIFO. Both TXE and RXNE events can be polled or handled by interrupts. See [Figure 283](#) through [Figure 286](#).

Another way to manage the data exchange is to use DMA (see [Communication using DMA \(direct memory addressing\)](#)).

If the next data is received when the RXFIFO is full, an overrun event occurs (see description of OVR flag at [Section 28.5.10: SPI status flags](#)). An overrun event can be polled or handled by an interrupt.

The BSY bit being set indicates ongoing transaction of a current data frame. When the clock signal runs continuously, the BSY flag stays set between data frames at master but becomes low for a minimum duration of one SPI clock at slave between each data frame transfer.

### Sequence handling

A few data frames can be passed at single sequence to complete a message. When transmission is enabled, a sequence begins and continues while any data is present in the TXFIFO of the master. The clock signal is provided continuously by the master until TXFIFO becomes empty, then it stops waiting for additional data.

In receive-only modes, half-duplex (BIDIMODE=1, BIDIOE=0) or simplex (BIDIMODE=0, RXONLY=1) the master starts the sequence immediately when both SPI is enabled and receive-only mode is activated. The clock signal is provided by the master and it does not stop until either SPI or receive-only mode is disabled by the master. The master receives data frames continuously up to this moment.

While the master can provide all the transactions in continuous mode (SCK signal is continuous) it has to respect slave capability to handle data flow and its content at anytime. When necessary, the master must slow down the communication and provide either a slower clock or separate frames or data sessions with sufficient delays. Be aware there is no underflow error signal for master or slave in SPI mode, and data from the slave is always transacted and processed by the master even if the slave could not prepare it correctly in time. It is preferable for the slave to use DMA, especially when data frames are shorter and bus rate is high.

Each sequence must be encased by the NSS pulse in parallel with the multislave system to select just one of the slaves for communication. In a single slave system it is not necessary to control the slave with NSS, but it is often better to provide the pulse here too, to synchronize the slave with the beginning of each data sequence. NSS can be managed by both software and hardware (see [Section 28.5.5: Slave select \(NSS\) pin management](#)).

When the BSY bit is set it signifies an ongoing data frame transaction. When the dedicated frame transaction is finished, the RXNE flag is raised. The last bit is just sampled and the complete data frame is stored in the RXFIFO.

### Procedure for disabling the SPI

When SPI is disabled, it is mandatory to follow the disable procedures described in this paragraph. It is important to do this before the system enters a low-power mode when the peripheral clock is stopped. Ongoing transactions can be corrupted in this case. In some modes the disable procedure is the only way to stop continuous communication running.

Master in full-duplex or transmit only mode can finish any transaction when it stops providing data for transmission. In this case, the clock stops after the last data transaction. Special care must be taken in packing mode when an odd number of data frames are

transacted to prevent some dummy byte exchange (refer to [Data packing](#) section). Before the SPI is disabled in these modes, the user must follow standard disable procedure. When the SPI is disabled at the master transmitter while a frame transaction is ongoing or next data frame is stored in TXFIFO, the SPI behavior is not guaranteed.

When the master is in any receive only mode, the only way to stop the continuous clock is to disable the peripheral by SPE=0. This must occur in specific time window within last data frame transaction just between the sampling time of its first bit and before its last bit transfer starts (in order to receive a complete number of expected data frames and to prevent any additional “dummy” data reading after the last valid data frame). Specific procedure must be followed when disabling SPI in this mode.

Data received but not read remains stored in RXFIFO when the SPI is disabled, and must be processed the next time the SPI is enabled, before starting a new sequence. To prevent having unread data, ensure that RXFIFO is empty when disabling the SPI, by using the correct disabling procedure, or by initializing all the SPI registers with a software reset via the control of a specific register dedicated to peripheral reset (see the SPiRST bits in the RCC\_APBIRSTR registers).

Standard disable procedure is based on pulling BSY status together with FTLVL[1:0] to check if a transmission session is fully completed. This check can be done in specific cases, too, when it is necessary to identify the end of ongoing transactions, for example:

- When NSS signal is managed by software and master has to provide proper end of NSS pulse for slave, or
- When transactions’ streams from DMA or FIFO are completed while the last data frame or CRC frame transaction is still ongoing in the peripheral bus.

The correct disable procedure is (except when receive only mode is used):

1. Wait until FTLVL[1:0] = 00 (no more data to transmit).
2. Wait until BSY=0 (the last data frame is processed).
3. Disable the SPI (SPE=0).
4. Read data until FRLVL[1:0] = 00 (read all the received data).

The correct disable procedure for certain receive only modes is:

1. Interrupt the receive flow by disabling SPI (SPE=0) in the specific time window while the last data frame is ongoing.
2. Wait until BSY=0 (the last data frame is processed).
3. Read data until FRLVL[1:0] = 00 (read all the received data).

**Note:**

*If packing mode is used and an odd number of data frames with a format less than or equal to 8 bits (fitting into one byte) has to be received, FRXTH must be set when FRLVL[1:0] = 01, in order to generate the RXNE event to read the last odd data frame and to keep good FIFO pointer alignment.*

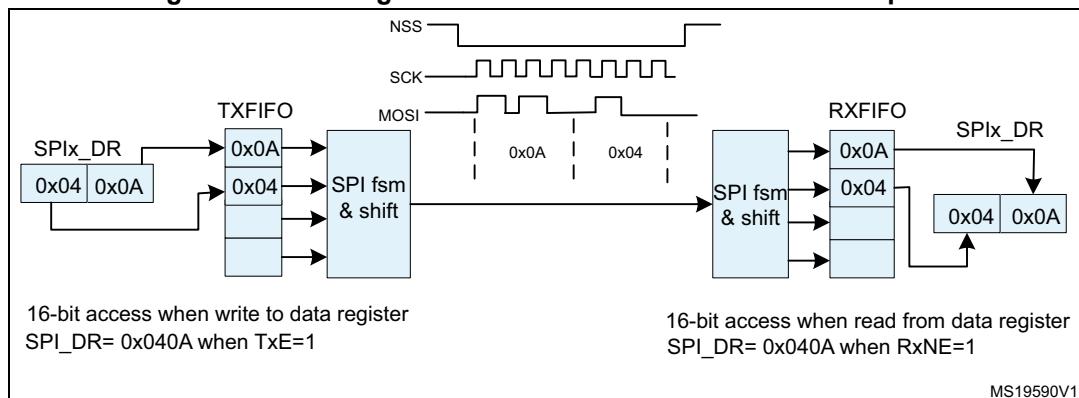
## **Data packing**

When the data frame size fits into one byte (less than or equal to 8 bits), data packing is used automatically when any read or write 16-bit access is performed on the SPIx\_DR register. The double data frame pattern is handled in parallel in this case. At first, the SPI operates using the pattern stored in the LSB of the accessed word, then with the other half stored in the MSB. [Figure 282](#) provides an example of data packing mode sequence handling. Two data frames are sent after the single 16-bit access the SPIx\_DR register of the transmitter. This sequence can generate just one RXNE event in the receiver if the

RXFIFO threshold is set to 16 bits (FRXTH=0). The receiver then has to access both data frames by a single 16-bit read of SPIx\_DR as a response to this single RXNE event. The RxFIFO threshold setting and the following read access must be always kept aligned at the receiver side, as data can be lost if it is not in line.

A specific problem appears if an odd number of such “fit into one byte” data frames must be handled. On the transmitter side, writing the last data frame of any odd sequence with an 8-bit access to SPIx\_DR is enough. The receiver has to change the Rx\_FIFO threshold level for the last data frame received in the odd sequence of frames in order to generate the RXNE event.

**Figure 282. Packing data in FIFO for transmission and reception**



1. In this example: Data size DS[3:0] is 4-bit configured, CPOL=0, CPHA=1 and LSBFIRST =0. The Data storage is always right aligned while the valid bits are performed on the bus only, the content of LSB byte goes first on the bus, the unused bits are not taken into account on the transmitter side and padded by zeros at the receiver side.

### Communication using DMA (direct memory addressing)

To operate at its maximum speed and to facilitate the data register read/write process required to avoid overrun, the SPI features a DMA capability, which implements a simple request/acknowledge protocol.

A DMA access is requested when the TXDMAEN or RXDMAEN enable bit in the SPIx\_CR2 register is set. Separate requests must be issued to the Tx and Rx buffers.

- In transmission, a DMA request is issued each time TXE is set to 1. The DMA then writes to the SPIx\_DR register.
- In reception, a DMA request is issued each time RXNE is set to 1. The DMA then reads the SPIx\_DR register.

See [Figure 283](#) through [Figure 286](#).

When the SPI is used only to transmit data, it is possible to enable only the SPI Tx DMA channel. In this case, the OVR flag is set because the data received is not read. When the SPI is used only to receive data, it is possible to enable only the SPI Rx DMA channel.

In transmission mode, when the DMA has written all the data to be transmitted (the TCIF flag is set in the DMA\_ISR register), the BSY flag can be monitored to ensure that the SPI communication is complete. This is required to avoid corrupting the last transmission before disabling the SPI or entering the Stop mode. The software must first wait until FTLVL[1:0]=00 and then until BSY=0.

When starting communication using DMA, to prevent DMA channel management raising error events, these steps must be followed in order:

1. Enable DMA Rx buffer in the RXDMAEN bit in the SPI\_CR2 register, if DMA Rx is used.
2. Enable DMA streams for Tx and Rx in DMA registers, if the streams are used.
3. Enable DMA Tx buffer in the TXDMAEN bit in the SPI\_CR2 register, if DMA Tx is used.
4. Enable the SPI by setting the SPE bit.

For code example refer to the Appendix sections [A.17.5: SPI master configuration with DMA code example](#) and [A.17.6: SPI slave configuration with DMA code example](#).

To close communication it is mandatory to follow these steps in order:

1. Disable DMA streams for Tx and Rx in the DMA registers, if the streams are used.
2. Disable the SPI by following the SPI disable procedure.
3. Disable DMA Tx and Rx buffers by clearing the TXDMAEN and RXDMAEN bits in the SPI\_CR2 register, if DMA Tx and/or DMA Rx are used.

### Packing with DMA

If the transfers are managed by DMA (TXDMAEN and RXDMAEN set in the SPIx\_CR2 register) packing mode is enabled/disabled automatically depending on the PSIZE value configured for SPI TX and the SPI RX DMA channel. If the DMA channel PSIZE value is equal to 16-bit and SPI data size is less than or equal to 8-bit, then packing mode is enabled. The DMA then automatically manages the write operations to the SPIx\_DR register.

If data packing mode is used and the number of data to transfer is not a multiple of two, the LDMA\_TX/LDMA\_RX bits must be set. The SPI then considers only one data for the transmission or reception to serve the last DMA transfer (for more details refer to [Data packing on page 780](#).)

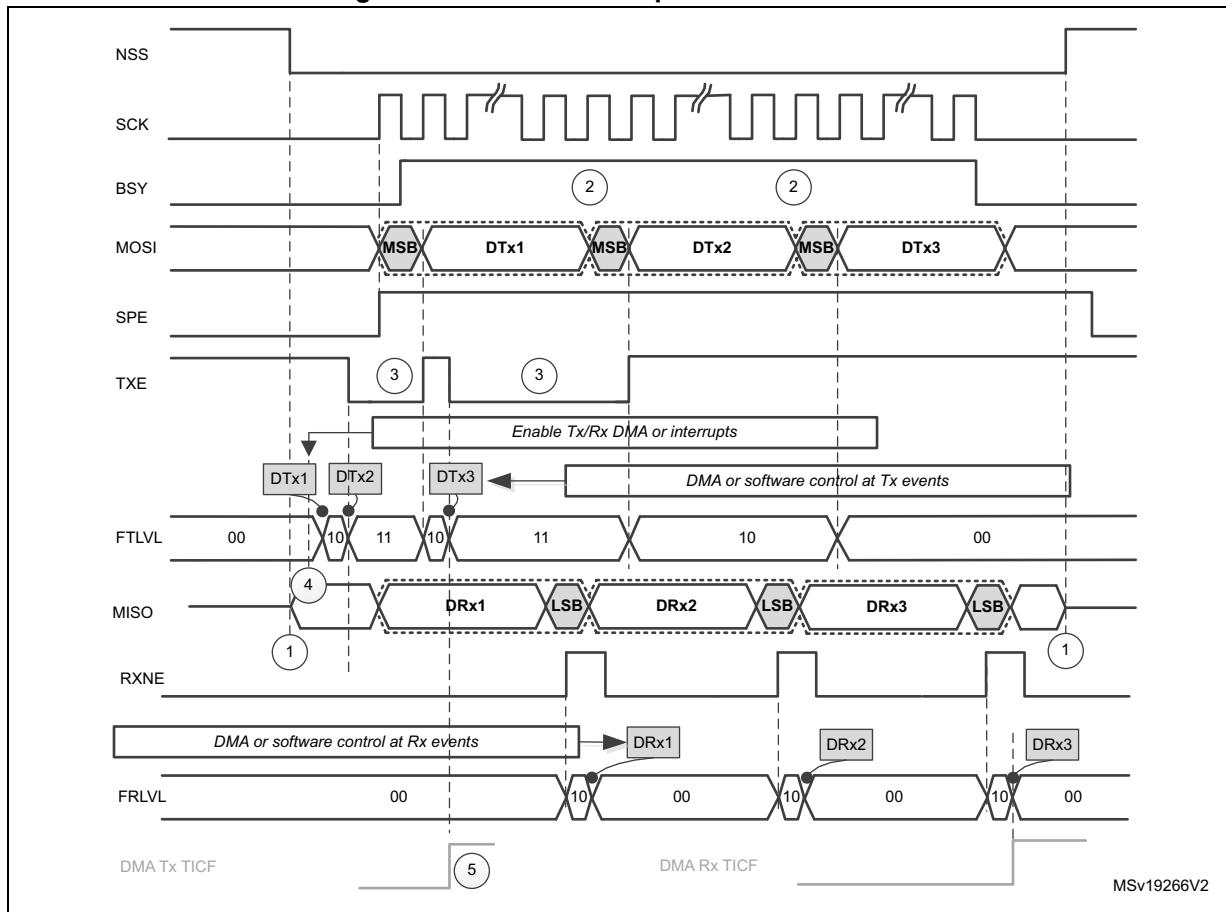
## Communication diagrams

Some typical timing schemes are explained in this section. These schemes are valid no matter if the SPI events are handled by polling, interrupts or DMA. For simplicity, the LSBFIRST=0, CPOL=0 and CPHA=1 setting is used as a common assumption here. No complete configuration of DMA streams is provided.

The following numbered notes are common for [Figure 283 on page 784](#) through [Figure 286 on page 787](#):

1. The slave starts to control MISO line as NSS is active and SPI is enabled, and is disconnected from the line when one of them is released. Sufficient time must be provided for the slave to prepare data dedicated to the master in advance before its transaction starts.  
At the master, the SPI peripheral takes control at MOSI and SCK signals (occasionally at NSS signal as well) only if SPI is enabled. If SPI is disabled the SPI peripheral is disconnected from GPIO logic, so the levels at these lines depends on GPIO setting exclusively.
2. At the master, BSY stays active between frames if the communication (clock signal) is continuous. At the slave, BSY signal always goes down for at least one clock cycle between data frames.
3. The TXE signal is cleared only if TXFIFO is full.
4. The DMA arbitration process starts just after the TXDMAEN bit is set. The TXE interrupt is generated just after the TXEIE is set. As the TXE signal is at an active level, data transfers to TxFIFO start, until TxFIFO becomes full or the DMA transfer completes.
5. If all the data to be sent can fit into TxFIFO, the DMA Tx TCIF flag can be raised even before communication on the SPI bus starts. This flag always rises before the SPI transaction is completed.
6. The CRC value for a package is calculated continuously frame by frame in the SPIx\_TXCRCR and SPIx\_RXCRCR registers. The CRC information is processed after the entire data package has completed, either automatically by DMA (Tx channel must be set to the number of data frames to be processed) or by SW (the user must handle CRCNEXT bit during the last data frame processing).  
While the CRC value calculated in SPIx\_TXCRCR is simply sent out by transmitter, received CRC information is loaded into Rx FIFO and then compared with the SPIx\_RXCRCR register content (CRC error flag can be raised here if any difference). This is why the user must take care to flush this information from the FIFO, either by software reading out all the stored content of Rx FIFO, or by DMA when the proper number of data frames is preset for Rx channel (number of data frames + number of CRC frames) (see the settings at the example assumption).
7. In data packed mode, TxE and RxNE events are paired and each read/write access to the FIFO is 16 bits wide until the number of data frames are even. If the Tx FIFO is  $\frac{3}{4}$  full FTLVL status stays at FIFO full level. That is why the last odd data frame cannot be stored before the Tx FIFO becomes  $\frac{1}{2}$  full. This frame is stored into Tx FIFO with an 8-bit access either by software or automatically by DMA when LDMA\_TX control is set.
8. To receive the last odd data frame in packed mode, the Rx threshold must be changed to 8-bit when the last data frame is processed, either by software setting FRXTH=1 or automatically by a DMA internal signal when LDMA\_RX is set.

Figure 283. Master full-duplex communication



Assumptions for master full-duplex communication example:

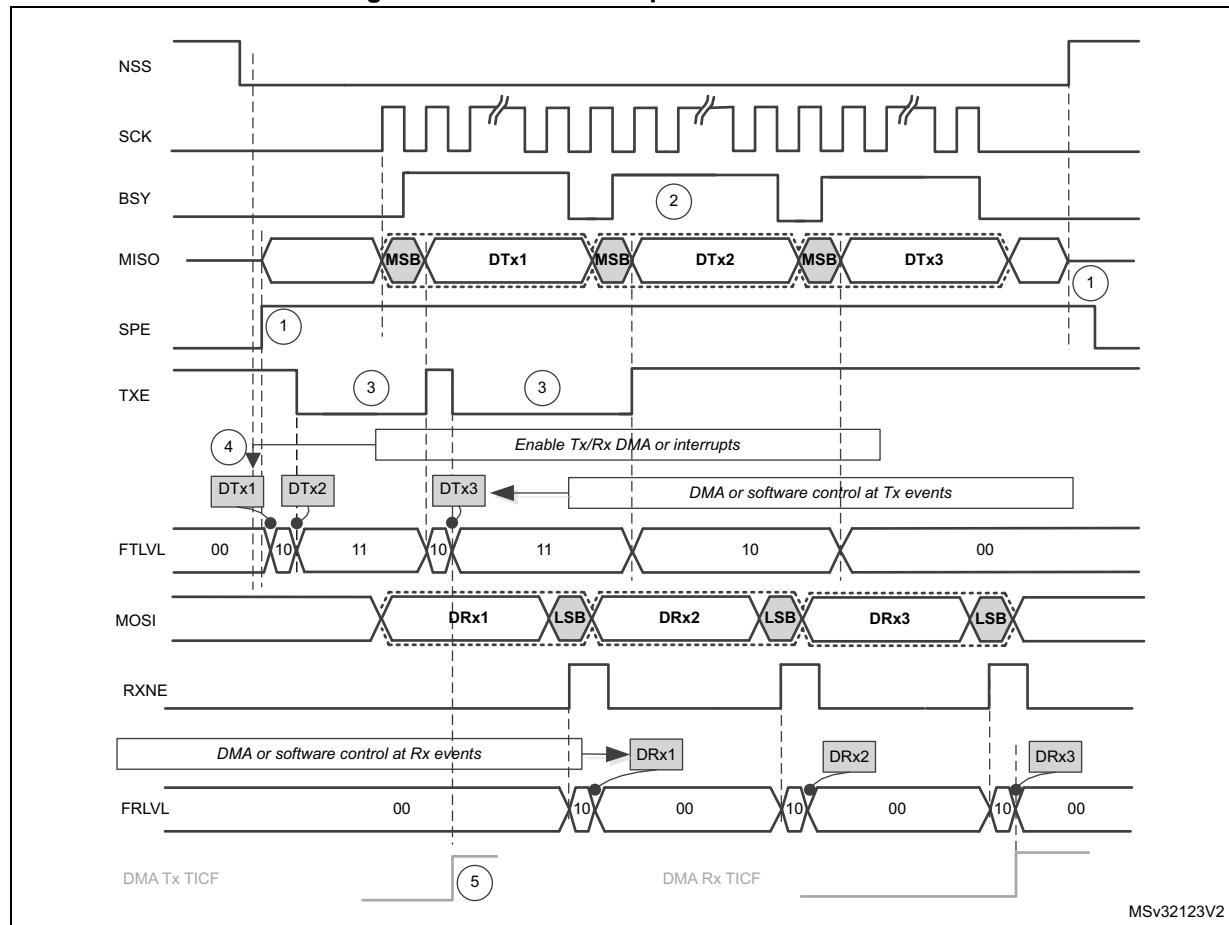
- Data size > 8 bit

If DMA is used:

- Number of Tx frames transacted by DMA is set to 3
- Number of Rx frames transacted by DMA is set to 3

See also : [Communication diagrams on page 783](#) for details about common assumptions and notes.

Figure 284. Slave full-duplex communication



Assumptions for slave full-duplex communication example:

- Data size > 8 bit

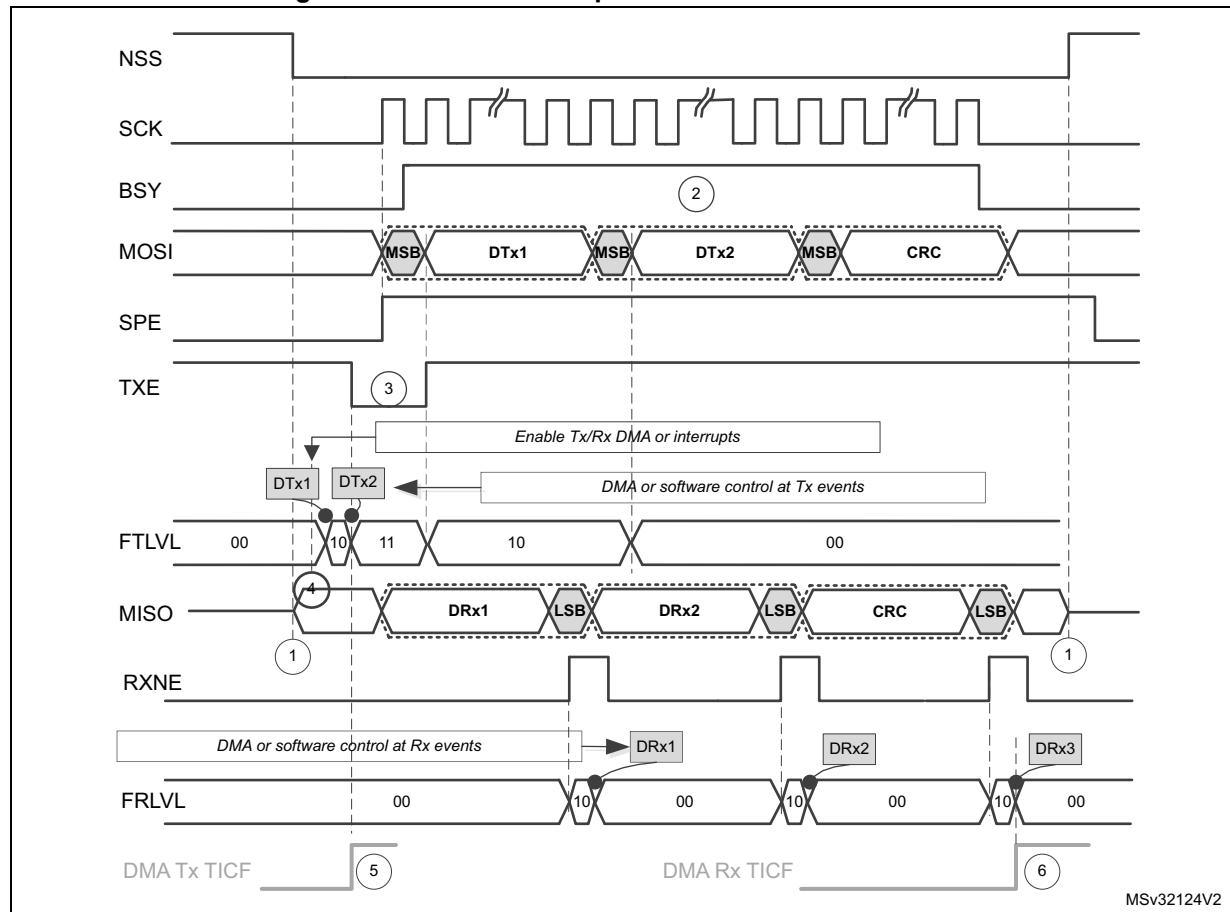
If DMA is used:

- Number of Tx frames transacted by DMA is set to 3
- Number of Rx frames transacted by DMA is set to 3

See also [Communication diagrams on page 783](#) for details about common assumptions and notes.

For code example refer to the Appendix section [A.17.3: SPI full duplex communication code example](#).

Figure 285. Master full-duplex communication with CRC



Assumptions for master full-duplex communication with CRC example:

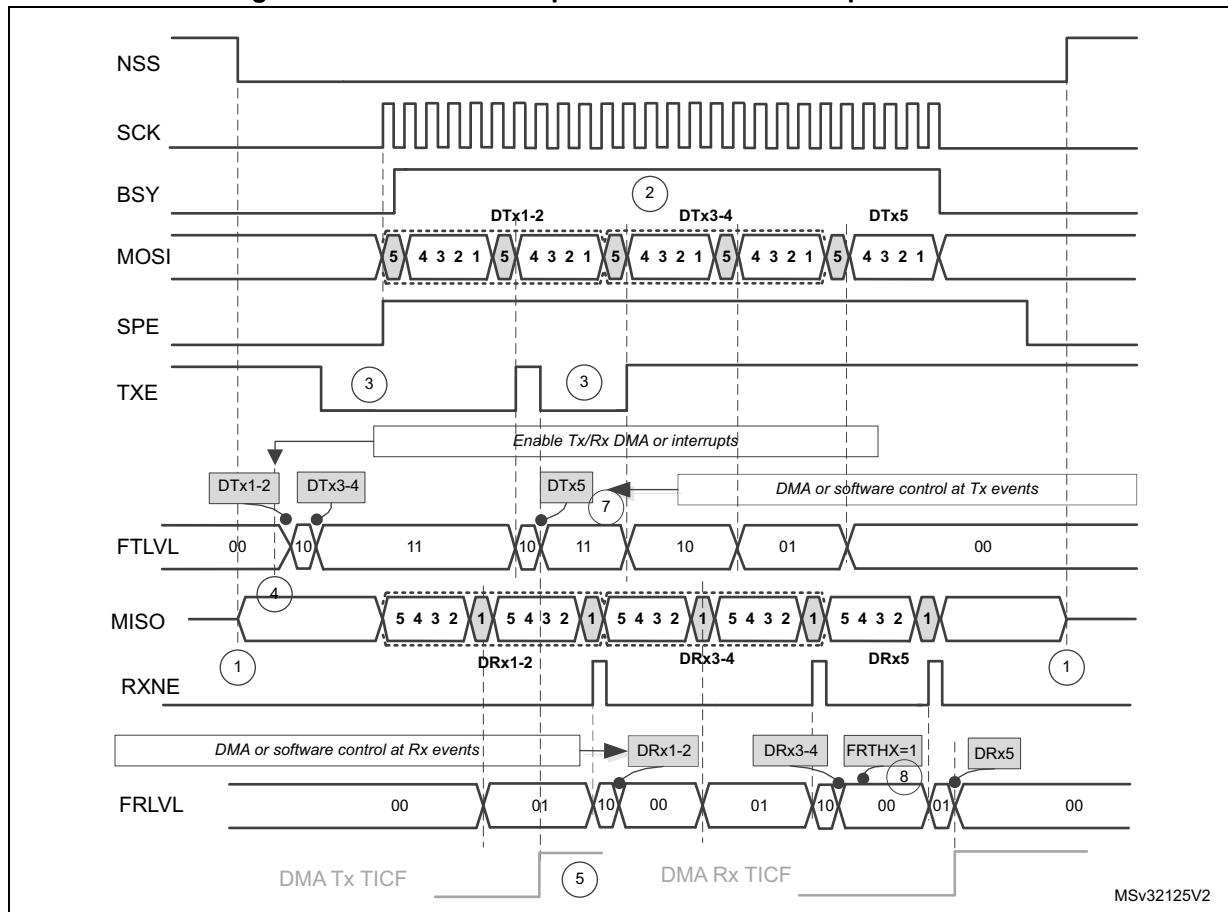
- Data size = 16 bit
- CRC enabled

If DMA is used:

- Number of Tx frames transacted by DMA is set to 2
- Number of Rx frames transacted by DMA is set to 3

See also : [Communication diagrams on page 783](#) for details about common assumptions and notes.

Figure 286. Master full-duplex communication in packed mode



Assumptions for master full-duplex communication in packed mode example:

- Data size = 5 bit
- Read/write FIFO is performed mostly by 16-bit access
- FRXTH=0

If DMA is used:

- Number of Tx frames to be transacted by DMA is set to 3
- Number of Rx frames to be transacted by DMA is set to 3
- PSIZE for both Tx and Rx DMA channel is set to 16-bit
- LDMA\_TX=1 and LDMA\_RX=1

See also : [Communication diagrams on page 783](#) for details about common assumptions and notes.

### 28.5.10 SPI status flags

Three status flags are provided for the application to completely monitor the state of the SPI bus.

#### Tx buffer empty flag (TXE)

The TXE flag is set when transmission TXFIFO has enough space to store data to send. TXE flag is linked to the TXFIFO level. The flag goes high and stays high until the TXFIFO level is lower or equal to 1/2 of the FIFO depth. An interrupt can be generated if the TXEIE bit in the SPIx\_CR2 register is set. The bit is cleared automatically when the TXFIFO level becomes greater than 1/2.

#### Rx buffer not empty (RXNE)

The RXNE flag is set depending on the FRXTH bit value in the SPIx\_CR2 register:

- If FRXTH is set, RXNE goes high and stays high until the RXFIFO level is greater or equal to 1/4 (8-bit).
- If FRXTH is cleared, RXNE goes high and stays high until the RXFIFO level is greater than or equal to 1/2 (16-bit).

An interrupt can be generated if the RXNEIE bit in the SPIx\_CR2 register is set.

The RXNE is cleared by hardware automatically when the above conditions are no longer true.

#### Busy flag (BSY)

The BSY flag is set and cleared by hardware (writing to this flag has no effect).

When BSY is set, it indicates that a data transfer is in progress on the SPI (the SPI bus is busy).

The BSY flag can be used in certain modes to detect the end of a transfer so that the software can disable the SPI or its peripheral clock before entering a low-power mode which does not provide a clock for the peripheral. This avoids corrupting the last transfer.

The BSY flag is also useful for preventing write collisions in a multimaster system.

The BSY flag is cleared under any one of the following conditions:

- When the SPI is correctly disabled
- When a fault is detected in Master mode (MODF bit set to 1)
- In Master mode, when it finishes a data transmission and no new data is ready to be sent
- In Slave mode, when the BSY flag is set to '0' for at least one SPI clock cycle between each data transfer.

Note:

*When the next transmission can be handled immediately by the master (e.g. if the master is in Receive-only mode or its Transmit FIFO is not empty), communication is continuous and the BSY flag remains set to '1' between transfers on the master side. Although this is not the case with a slave, it is recommended to use always the TXE and RXNE flags (instead of the BSY flags) to handle data transmission or reception operations.*

### 28.5.11 SPI error flags

An SPI interrupt is generated if one of the following error flags is set and interrupt is enabled by setting the ERRIE bit.

#### Overrun flag (OVR)

An overrun condition occurs when data is received by a master or slave and the RXFIFO has not enough space to store this received data. This can happen if the software or the DMA did not have enough time to read the previously received data (stored in the RXFIFO) or when space for data storage is limited e.g. the RXFIFO is not available when CRC is enabled in receive only mode so in this case the reception buffer is limited into a single data frame buffer (see [Section 28.5.14: CRC calculation](#)).

When an overrun condition occurs, the newly received value does not overwrite the previous one in the RXFIFO. The newly received value is discarded and all data transmitted subsequently is lost. Clearing the OVR bit is done by a read access to the SPI\_DR register followed by a read access to the SPI\_SR register.

#### Mode fault (MODF)

Mode fault occurs when the master device has its internal NSS signal (NSS pin in NSS hardware mode, or SSI bit in NSS software mode) pulled low. This automatically sets the MODF bit. Master mode fault affects the SPI interface in the following ways:

- The MODF bit is set and an SPI interrupt is generated if the ERRIE bit is set.
- The SPE bit is cleared. This blocks all output from the device and disables the SPI interface.
- The MSTR bit is cleared, thus forcing the device into slave mode.

Use the following software sequence to clear the MODF bit:

1. Make a read or write access to the SPIx\_SR register while the MODF bit is set.
2. Then write to the SPIx\_CR1 register.

To avoid any multiple slave conflicts in a system comprising several MCUs, the NSS pin must be pulled high during the MODF bit clearing sequence. The SPE and MSTR bits can be restored to their original state after this clearing sequence. As a security, hardware does not allow the SPE and MSTR bits to be set while the MODF bit is set. In a slave device the MODF bit cannot be set except as the result of a previous multimaster conflict.

#### CRC error (CRCERR)

This flag is used to verify the validity of the value received when the CRCEN bit in the SPIx\_CR1 register is set. The CRCERR flag in the SPIx\_SR register is set if the value received in the shift register does not match the receiver SPIx\_RXCRCR value. The flag is cleared by the software.

#### TI mode frame format error (FRE)

A TI mode frame format error is detected when an NSS pulse occurs during an ongoing communication when the SPI is operating in slave mode and configured to conform to the TI mode protocol. When this error occurs, the FRE flag is set in the SPIx\_SR register. The SPI is not disabled when an error occurs, the NSS pulse is ignored, and the SPI waits for the next NSS pulse before starting a new transfer. The data may be corrupted since the error detection may result in the loss of two data bytes.

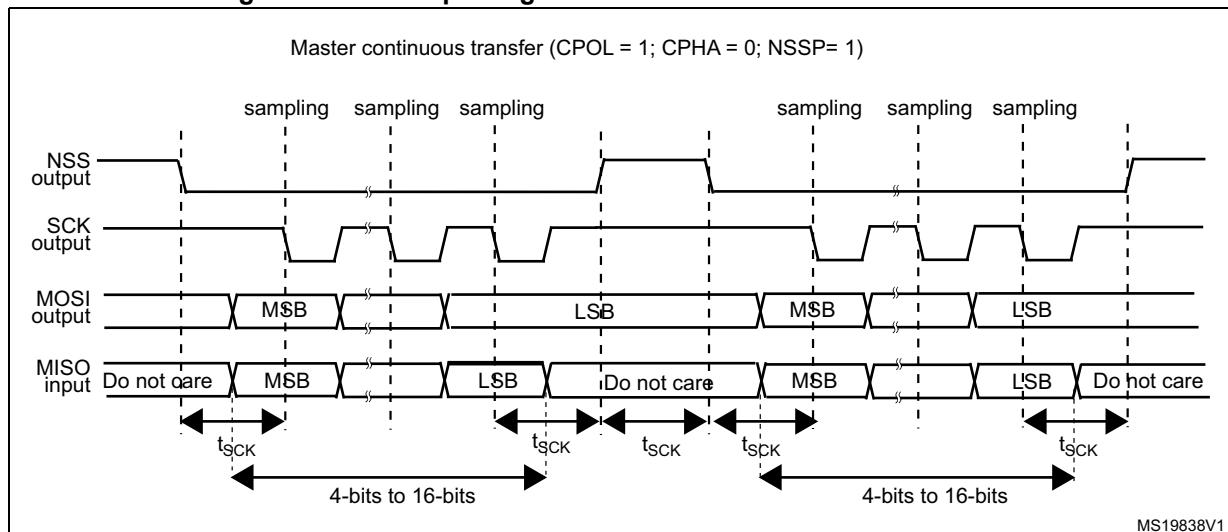
The FRE flag is cleared when SPIx\_SR register is read. If the ERRIE bit is set, an interrupt is generated on the NSS error detection. In this case, the SPI should be disabled because data consistency is no longer guaranteed and communications should be reinitiated by the master when the slave SPI is enabled again.

### 28.5.12 NSS pulse mode

This mode is activated by the NSSP bit in the SPIx\_CR2 register and it takes effect only if the SPI interface is configured as Motorola SPI master (FRF=0) with capture on the first edge (SPIx\_CR1 CPHA = 0, CPOL setting is ignored). When activated, an NSS pulse is generated between two consecutive data frame transfers when NSS stays at high level for the duration of one clock period at least. This mode allows the slave to latch data. NSSP pulse mode is designed for applications with a single master-slave pair.

*Figure 287* illustrates NSS pin management when NSSP pulse mode is enabled.

**Figure 287. NSSP pulse generation in Motorola SPI master mode**



**Note:** Similar behavior is encountered when CPOL = 0. In this case the sampling edge is the *rising* edge of SCK, and NSS assertion and deassertion refer to this sampling edge.

### 28.5.13 TI mode

#### TI protocol in master mode

The SPI interface is compatible with the TI protocol. The FRF bit of the SPIx\_CR2 register can be used to configure the SPI to be compliant with this protocol.

The clock polarity and phase are forced to conform to the TI protocol requirements whatever the values set in the SPIx\_CR1 register. NSS management is also specific to the TI protocol which makes the configuration of NSS management through the SPIx\_CR1 and SPIx\_CR2 registers (SSM, SSI, SSOE) impossible in this case.

In slave mode, the SPI baud rate prescaler is used to control the moment when the MISO pin state changes to HiZ when the current transaction finishes (see *Figure 288*). Any baud rate can be used, making it possible to determine this moment with optimal flexibility. However, the baud rate is generally set to the external master clock baud rate. The delay for the MISO signal to become HiZ ( $t_{release}$ ) depends on internal resynchronization and on the

baud rate value set in through the BR[2:0] bits in the SPIx\_CR1 register. It is given by the formula:

$$\frac{t_{\text{baud\_rate}}}{2} + 4 \times t_{\text{pclk}} < t_{\text{release}} < \frac{t_{\text{baud\_rate}}}{2} + 6 \times t_{\text{pclk}}$$

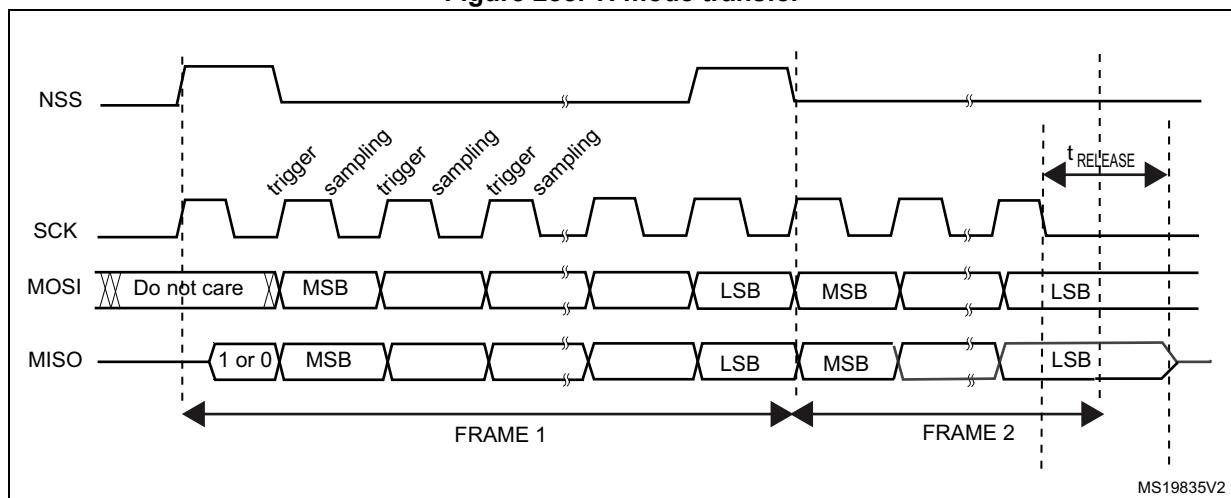
If the slave detects a misplaced NSS pulse during a data frame transaction the TIFRE flag is set.

If the data size is equal to 4-bits or 5-bits, the master in full-duplex mode or transmit-only mode uses a protocol with one more dummy data bit added after LSB. TI NSS pulse is generated above this dummy bit clock cycle instead of the LSB in each period.

This feature is not available for Motorola SPI communications (FRF bit set to 0).

*Figure 288: TI mode transfer* shows the SPI communication waveforms when TI mode is selected.

**Figure 288. TI mode transfer**



### 28.5.14 CRC calculation

Two separate CRC calculators are implemented in order to check the reliability of transmitted and received data. The SPI offers CRC8 or CRC16 calculation independently of the frame data length, which can be fixed to 8-bit or 16-bit. For all the other data frame lengths, no CRC is available.

#### CRC principle

CRC calculation is enabled by setting the CRCEN bit in the SPIx\_CR1 register before the SPI is enabled (SPE = 1). The CRC value is calculated using an odd programmable polynomial on each bit. The calculation is processed on the sampling clock edge defined by the CPHA and CPOL bits in the SPIx\_CR1 register. The calculated CRC value is checked automatically at the end of the data block as well as for transfer managed by CPU or by the DMA. When a mismatch is detected between the CRC calculated internally on the received data and the CRC sent by the transmitter, a CRCERR flag is set to indicate a data corruption error. The right procedure for handling the CRC calculation depends on the SPI configuration and the chosen transfer management.

**Note:** *The polynomial value should only be odd. No even values are supported.*

### CRC transfer managed by CPU

Communication starts and continues normally until the last data frame has to be sent or received in the SPIx\_DR register. Then CRCNEXT bit has to be set in the SPIx\_CR1 register to indicate that the CRC frame transaction follows after the transaction of the currently processed data frame. The CRCNEXT bit must be set before the end of the last data frame transaction. CRC calculation is frozen during CRC transaction.

The received CRC is stored in the RXFIFO like a data byte or word. That is why in CRC mode only, the reception buffer has to be considered as a single 16-bit buffer used to receive only one data frame at a time.

A CRC-format transaction usually takes one more data frame to communicate at the end of data sequence. However, when setting an 8-bit data frame checked by 16-bit CRC, two more frames are necessary to send the complete CRC.

When the last CRC data is received, an automatic check is performed comparing the received value and the value in the SPIx\_RXCRC register. Software has to check the CRCERR flag in the SPIx\_SR register to determine if the data transfers were corrupted or not. Software clears the CRCERR flag by writing '0' to it.

After the CRC reception, the CRC value is stored in the RXFIFO and must be read in the SPIx\_DR register in order to clear the RXNE flag.

### CRC transfer managed by DMA

When SPI communication is enabled with CRC communication and DMA mode, the transmission and reception of the CRC at the end of communication is automatic (with the exception of reading CRC data in receive only mode). The CRCNEXT bit does not have to be handled by the software. The counter for the SPI transmission DMA channel has to be set to the number of data frames to transmit excluding the CRC frame. On the receiver side, the received CRC value is handled automatically by DMA at the end of the transaction but user must take care to flush out received CRC information from RXFIFO as it is always loaded into it. In full-duplex mode, the counter of the reception DMA channel can be set to the number of data frames to receive including the CRC, which means, for example, in the specific case of an 8-bit data frame checked by 16-bit CRC:

$$\text{DMA\_RX} = \text{Numb\_of\_data} + 2$$

In receive only mode, the DMA reception channel counter should contain only the amount of data transferred, excluding the CRC calculation. Then based on the complete transfer from DMA, all the CRC values must be read back by software from FIFO as it works as a single buffer in this mode.

At the end of the data and CRC transfers, the CRCERR flag in the SPIx\_SR register is set if corruption occurred during the transfer.

If packing mode is used, the LDMA\_RX bit needs managing if the number of data is odd.

### Resetting the SPIx\_TXCRC and SPIx\_RXCRC values

The SPIx\_TXCRC and SPIx\_RXCRC values are cleared automatically when new data is sampled after a CRC phase. This allows the use of DMA circular mode (not available in receive-only mode) in order to transfer data without any interruption, (several data blocks covered by intermediate CRC checking phases).

If the SPI is disabled during a communication the following sequence must be followed:

1. Disable the SPI
2. Clear the CRCEN bit
3. Enable the CRCEN bit
4. Enable the SPI

Note:

*When the SPI interface is configured as a slave, the NSS internal signal needs to be kept low during transaction of the CRC phase once the CRCNEXT signal is released. That is why the CRC calculation cannot be used at NSS Pulse mode when NSS hardware mode should be applied at slave normally.*

*At TI mode, despite the fact that clock phase and clock polarity setting is fixed and independent on SPIx\_CR1 register, the corresponding setting CPOL=0 CPHA=1 has to be kept at the SPIx\_CR1 register anyway if CRC is applied. In addition, the CRC calculation has to be reset between sessions by SPI disable sequence with re-enable the CRCEN bit described above at both master and slave side, else CRC calculation can be corrupted at this specific mode.*

## 28.6 SPI interrupts

During SPI communication an interrupt can be generated by the following events:

- Transmit TXFIFO ready to be loaded
- Data received in Receive RXFIFO
- Master mode fault
- Overrun error
- TI frame format error
- CRC protocol error

Interrupts can be enabled and disabled separately.

Table 114. SPI interrupt requests

Interrupt event	Event flag	Enable Control bit
Transmit TXFIFO ready to be loaded	TXE	TXEIE
Data received in RXFIFO	RXNE	RXNEIE
Master Mode fault event	MODF	ERRIE
Overrun error	OVR	
TI frame format error	FRE	
CRC protocol error	CRCERR	

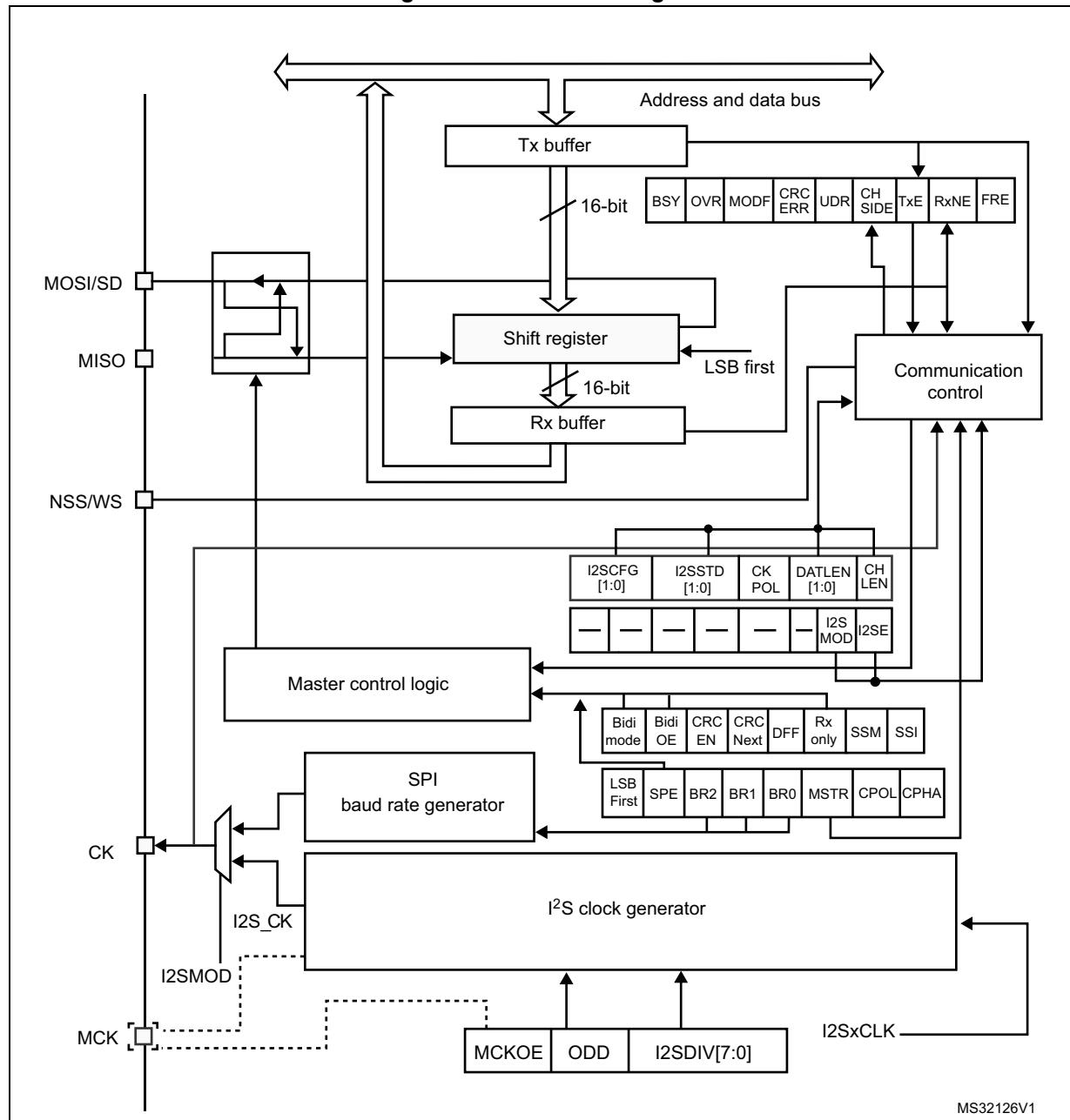
For code example refer to the Appendix section [A.17.4: SPI interrupt code example](#).

## 28.7 I2S functional description

### 28.7.1 I2S general description

The block diagram of the I2S is shown in [Figure 289](#).

**Figure 289. I2S block diagram**



1. MCK is mapped on the MISO pin.

The SPI can function as an audio I2S interface when the I2S capability is enabled (by setting the I2SMOD bit in the SPIx\_I2SCFGR register). This interface mainly uses the same pins, flags and interrupts as the SPI.

The I<sup>2</sup>S shares three common pins with the SPI:

- SD: Serial Data (mapped on the MOSI pin) to transmit or receive the two time-multiplexed data channels (in half-duplex mode only).
- WS: Word Select (mapped on the NSS pin) is the data control signal output in master mode and input in slave mode.
- CK: Serial Clock (mapped on the SCK pin) is the serial clock output in master mode and serial clock input in slave mode.

An additional pin can be used when a master clock output is needed for some external audio devices:

- MCK: Master Clock (mapped separately) is used, when the I<sup>2</sup>S is configured in master mode (and when the MCKOE bit in the SPIx\_I2SPR register is set), to output this additional clock generated at a preconfigured frequency rate equal to  $256 \times f_S$  for all I<sup>2</sup>S modes, and to  $128 \times f_S$  for all PCM modes, where  $f_S$  is the audio sampling frequency.

The I<sup>2</sup>S uses its own clock generator to produce the communication clock when it is set in master mode. This clock generator is also the source of the master clock output. Two additional registers are available in I<sup>2</sup>S mode. One is linked to the clock generator configuration SPIx\_I2SPR and the other one is a generic I<sup>2</sup>S configuration register SPIx\_I2SCFGR (audio standard, slave/master mode, data format, packet frame, clock polarity, etc.).

The SPIx\_CR1 register and all CRC registers are not used in the I<sup>2</sup>S mode. Likewise, the SSOE bit in the SPIx\_CR2 register and the MODF and CRCERR bits in the SPIx\_SR are not used.

The I<sup>2</sup>S uses the same SPI register for data transfer (SPIx\_DR) in 16-bit wide mode.

## 28.7.2 I<sup>2</sup>S full duplex

*Figure 290* shows how to perform full-duplex communications using two SPI/I<sup>2</sup>S instances. In this case, the WS and CK IOs of both SPI/I<sup>2</sup>S must be connected together.

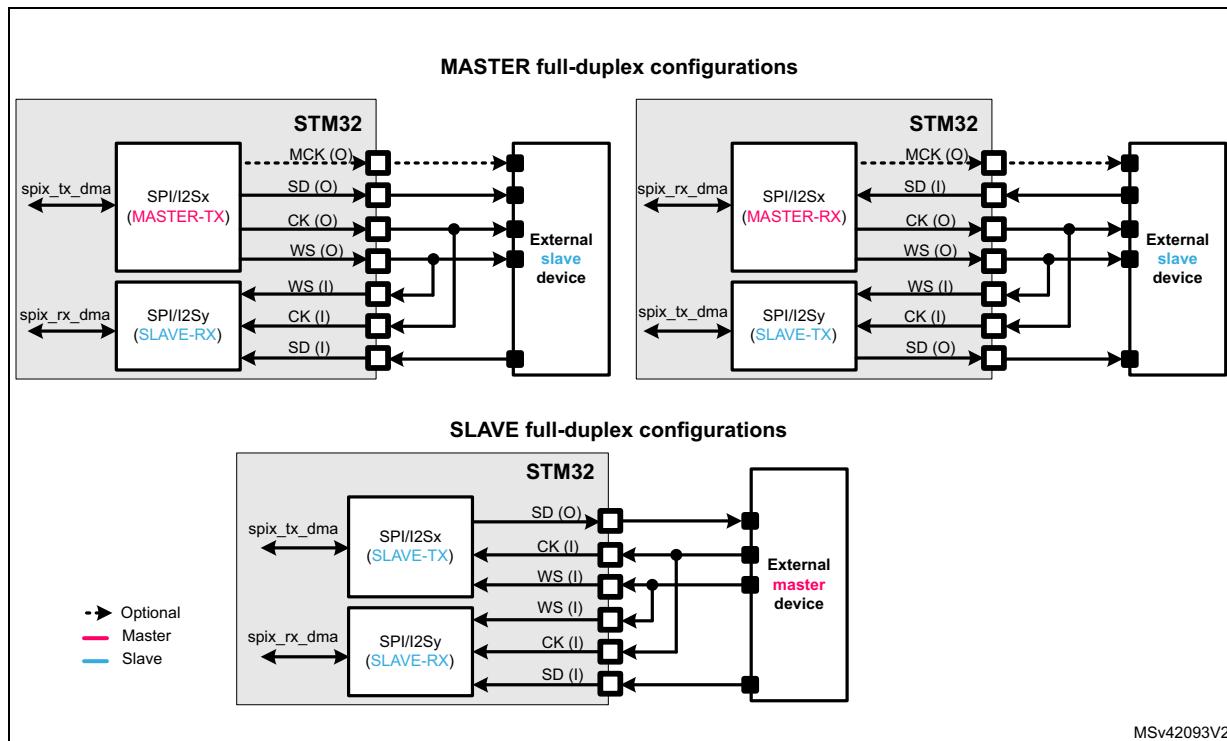
For the master full-duplex mode, one of the SPI/I<sup>2</sup>S block must be programmed in master (I2SCFG = '10' or '11'), and the other SPI/I<sup>2</sup>S block must be programmed in slave (I2SCFG = '00' or '01'). The MCK can be generated or not, depending on the application needs.

For the slave full-duplex mode, both SPI/I<sup>2</sup>S blocks must be programmed in slave. One of them in the slave receiver (I2SCFG = '01'), and the other in the slave transmitter (I2SCFG = '00'). The master external device then provides the bit clock (CK) and the frame synchronization (WS).

Note that the full-duplex mode can be used for all the supported standards: I<sup>2</sup>S Philips, MSB-justified, LSB-justified and PCM.

For the full-duplex mode, both SPI/I<sup>2</sup>S instances must use the same standard, with the same parameters: I2SMOD, I2SSTD, CKPOL, PCMSYNC, DATLEN and CHLEN must contain the same value on both instances.

Figure 290. Full-duplex communication



### 28.7.3 Supported audio protocols

The three-line bus has to handle only audio data generally time-multiplexed on two channels: the right channel and the left channel. However there is only one 16-bit register for transmission or reception. So, it is up to the software to write into the data register the appropriate value corresponding to each channel side, or to read the data from the data register and to identify the corresponding channel by checking the CHSIDE bit in the SPIx\_SR register. Channel left is always sent first followed by the channel right (CHSIDE has no meaning for the PCM protocol).

Four data and packet frames are available. Data may be sent with a format of:

- 16-bit data packed in a 16-bit frame
- 16-bit data packed in a 32-bit frame
- 24-bit data packed in a 32-bit frame
- 32-bit data packed in a 32-bit frame

When using 16-bit data extended on 32-bit packet, the first 16 bits (MSB) are the significant bits, the 16-bit LSB is forced to 0 without any need for software action or DMA request (only one read/write operation).

The 24-bit and 32-bit data frames need two CPU read or write operations to/from the SPIx\_DR register or two DMA operations if the DMA is preferred for the application. For 24-bit data frame specifically, the 8 non-significant bits are extended to 32 bits with 0-bits (by hardware).

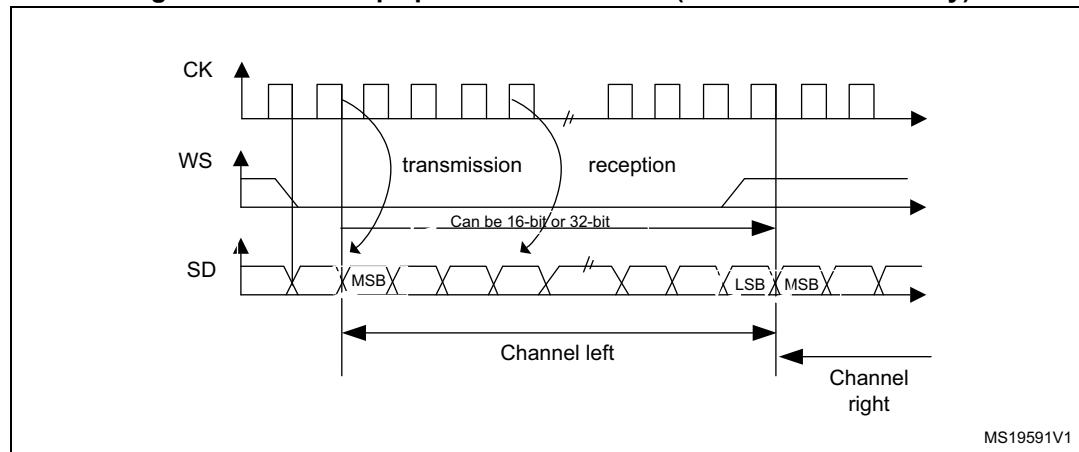
For all data formats and communication standards, the most significant bit is always sent first (MSB first).

The I<sup>2</sup>S interface supports four audio standards, configurable using the I2SSTD[1:0] and PCMSYNC bits in the SPIx\_I2SCFGR register.

### I<sup>2</sup>S Philips standard

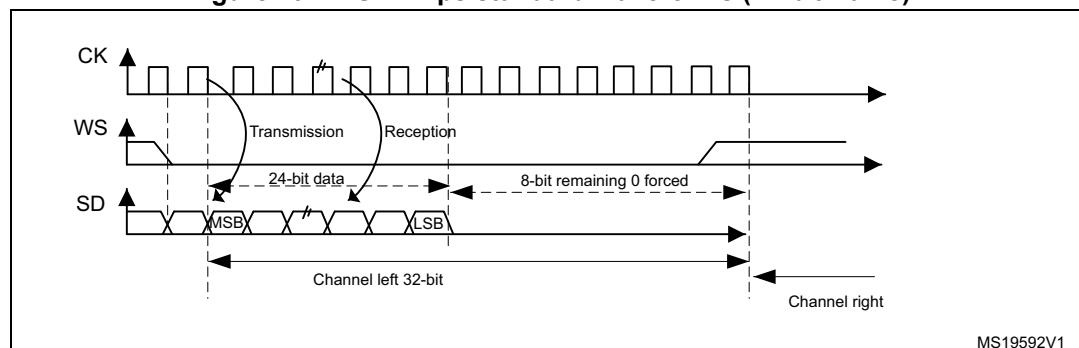
For this standard, the WS signal is used to indicate which channel is being transmitted. It is activated one CK clock cycle before the first bit (MSB) is available.

**Figure 291. I<sup>2</sup>S Philips protocol waveforms (16/32-bit full accuracy)**



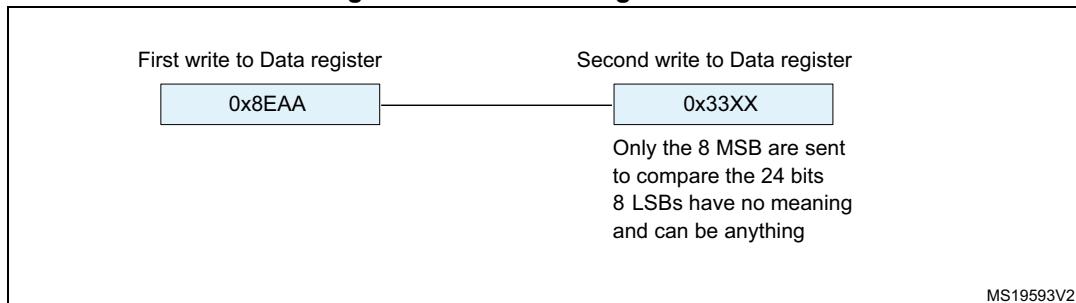
Data are latched on the falling edge of CK (for the transmitter) and are read on the rising edge (for the receiver). The WS signal is also latched on the falling edge of CK.

**Figure 292. I<sup>2</sup>S Philips standard waveforms (24-bit frame)**

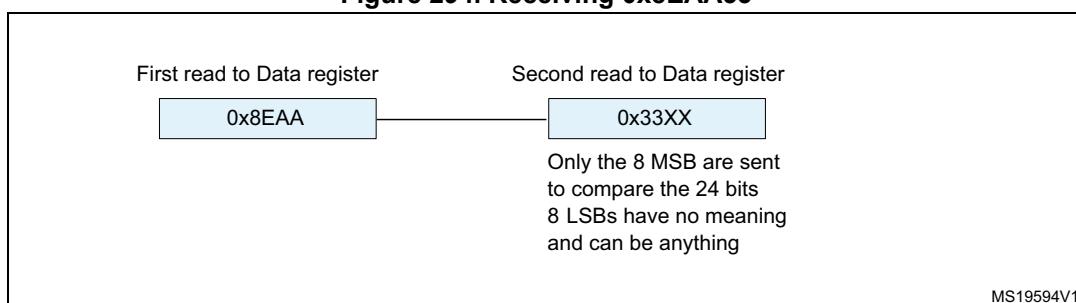
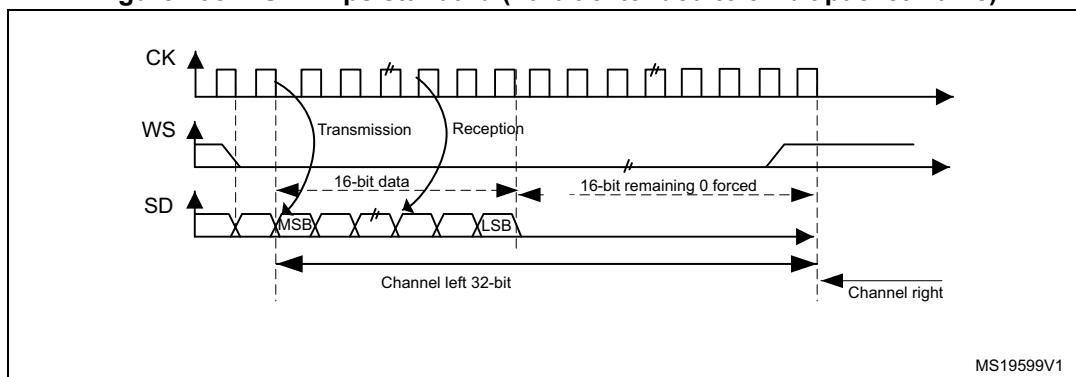


This mode needs two write or read operations to/from the SPIx\_DR register.

- In transmission mode:  
If 0x8EAA33 has to be sent (24-bit):

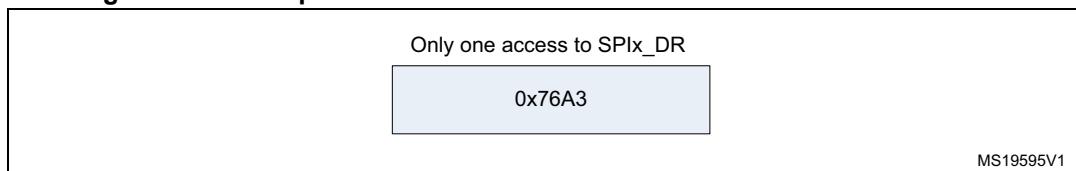
**Figure 293. Transmitting 0x8EAA33**

- In reception mode:  
If data 0x8EAA33 is received:

**Figure 294. Receiving 0x8EAA33****Figure 295. I<sup>2</sup>S Philips standard (16-bit extended to 32-bit packet frame)**

When 16-bit data frame extended to 32-bit channel frame is selected during the I2S configuration phase, only one access to the SPIx\_DR register is required. The 16 remaining bits are forced by hardware to 0x0000 to extend the data to 32-bit format.

If the data to transmit or the received data are 0x76A3 (0x76A30000 extended to 32-bit), the operation shown in [Figure 296](#) is required.

**Figure 296. Example of 16-bit data frame extended to 32-bit channel frame**

For transmission, each time an MSB is written to SPIx\_DR, the TXE flag is set and its interrupt, if allowed, is generated to load the SPIx\_DR register with the new value to send. This takes place even if 0x0000 have not yet been sent because it is done by hardware.

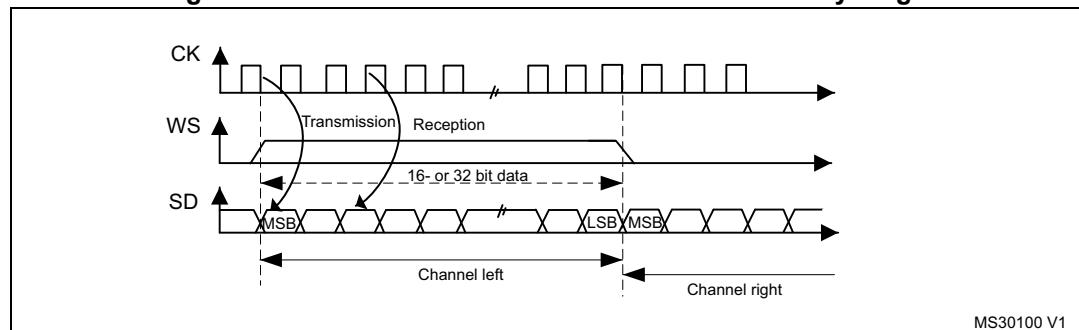
For reception, the RXNE flag is set and its interrupt, if allowed, is generated when the first 16 MSB half-word is received.

In this way, more time is provided between two write or read operations, which prevents underrun or overrun conditions (depending on the direction of the data transfer).

### MSB justified standard

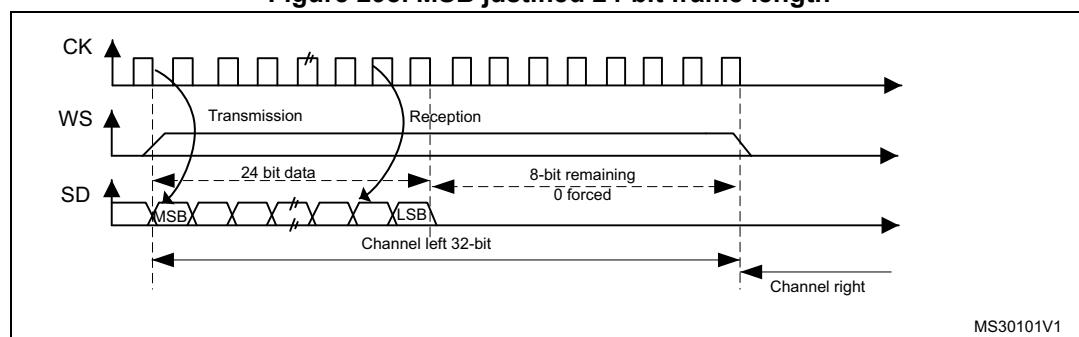
For this standard, the WS signal is generated at the same time as the first data bit, which is the MSBit.

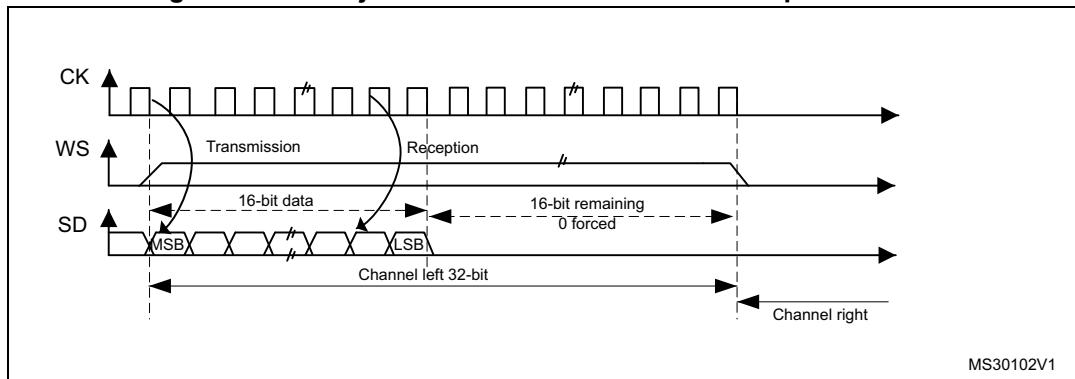
**Figure 297. MSB Justified 16-bit or 32-bit full-accuracy length**



Data are latched on the falling edge of CK (for transmitter) and are read on the rising edge (for the receiver).

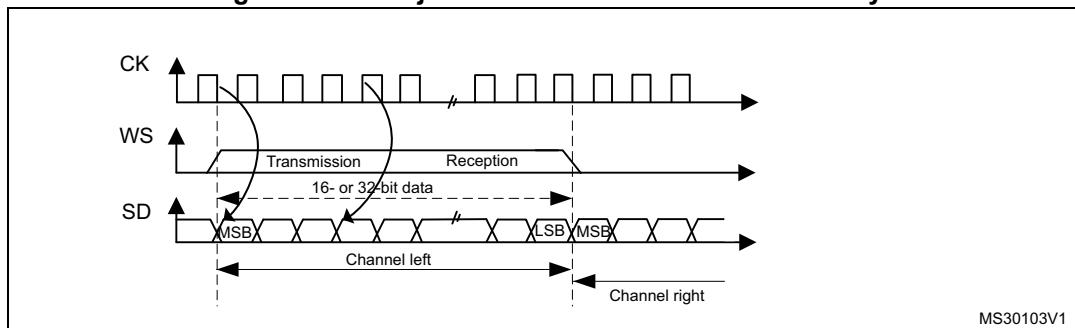
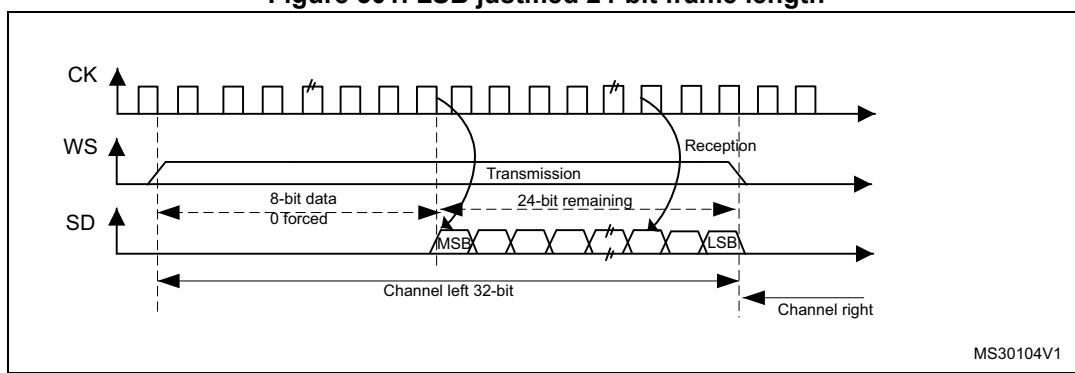
**Figure 298. MSB justified 24-bit frame length**



**Figure 299. MSB justified 16-bit extended to 32-bit packet frame****LSB justified standard**

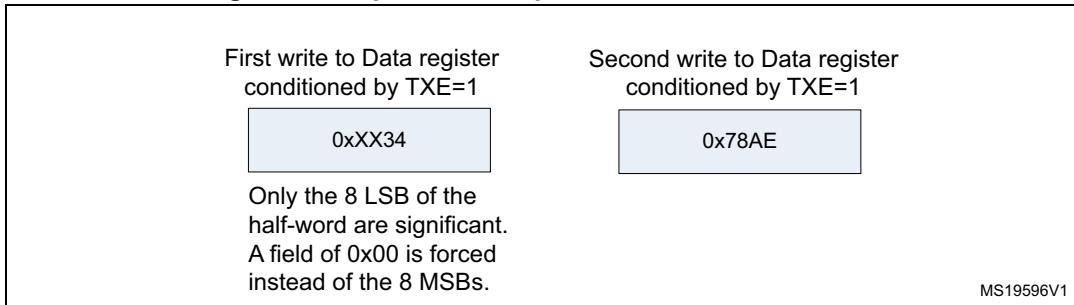
This standard is similar to the MSB justified standard (no difference for the 16-bit and 32-bit full-accuracy frame formats).

The sampling of the input and output signals is the same as for the I<sup>2</sup>S Philips standard.

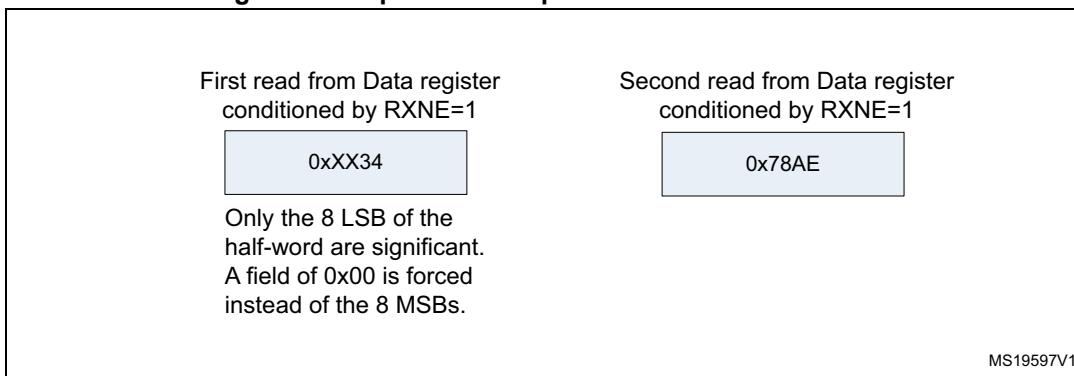
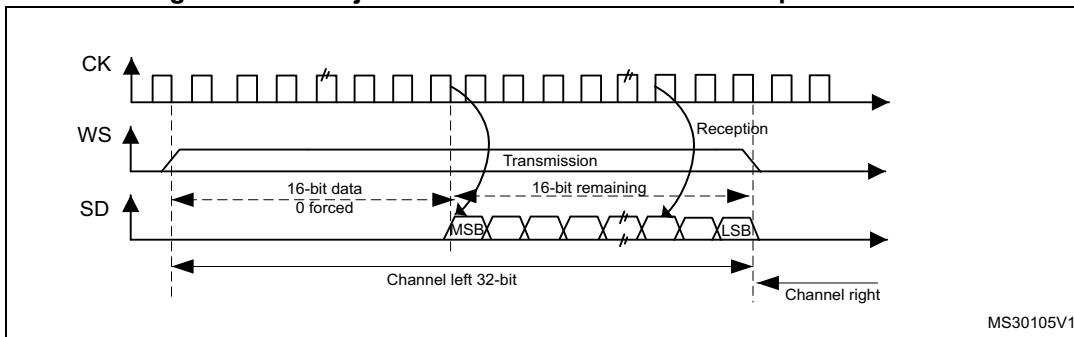
**Figure 300. LSB justified 16-bit or 32-bit full-accuracy****Figure 301. LSB justified 24-bit frame length**

- In transmission mode:

If data 0x3478AE have to be transmitted, two write operations to the SPIx\_DR register are required by software or by DMA. The operations are shown below.

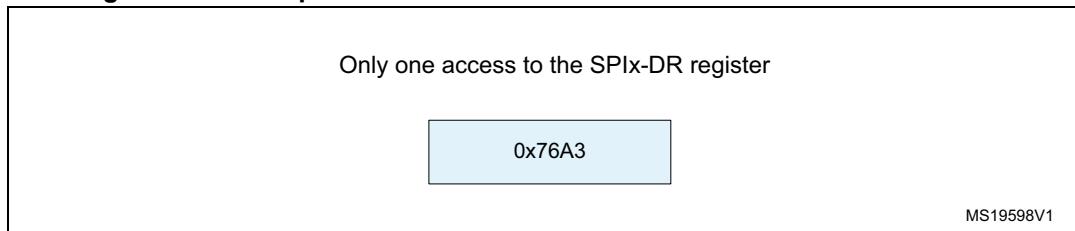
**Figure 302. Operations required to transmit 0x3478AE**

- In reception mode:  
If data 0x3478AE are received, two successive read operations from the SPIx\_DR register are required on each RXNE event.

**Figure 303. Operations required to receive 0x3478AE****Figure 304. LSB justified 16-bit extended to 32-bit packet frame**

When 16-bit data frame extended to 32-bit channel frame is selected during the I2S configuration phase, Only one access to the SPIx\_DR register is required. The 16 remaining bits are forced by hardware to 0x0000 to extend the data to 32-bit format. In this case it corresponds to the half-word MSB.

If the data to transmit or the received data are 0x76A3 (0x0000 76A3 extended to 32-bit), the operation shown in [Figure 305](#) is required.

**Figure 305. Example of 16-bit data frame extended to 32-bit channel frame**

In transmission mode, when a TXE event occurs, the application has to write the data to be transmitted (in this case 0x76A3). The 0x000 field is transmitted first (extension on 32-bit). The TXE flag is set again as soon as the effective data (0x76A3) is sent on SD.

In reception mode, RXNE is asserted as soon as the significant half-word is received (and not the 0x0000 field).

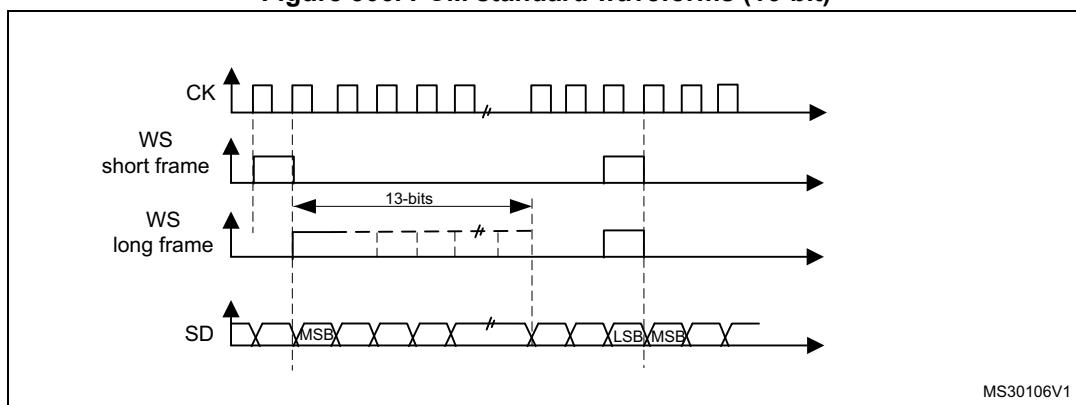
In this way, more time is provided between two write or read operations to prevent underrun or overrun conditions.

### PCM standard

For the PCM standard, there is no need to use channel-side information. The two PCM modes (short and long frame) are available and configurable using the PCMSYNC bit in SPIx\_I2SCFGR register.

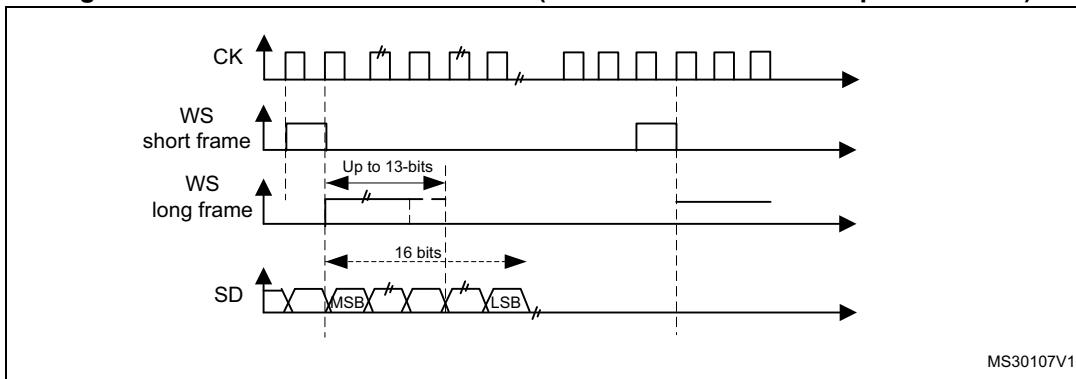
In PCM mode, the output signals (WS, SD) are sampled on the rising edge of CK signal. The input signals (WS, SD) are captured on the falling edge of CK.

Note that CK and WS are configured as output in MASTER mode.

**Figure 306. PCM standard waveforms (16-bit)**

For long frame synchronization, the WS signal assertion time is fixed to 13 bits in master mode.

For short frame synchronization, the WS synchronization signal is only one cycle long.

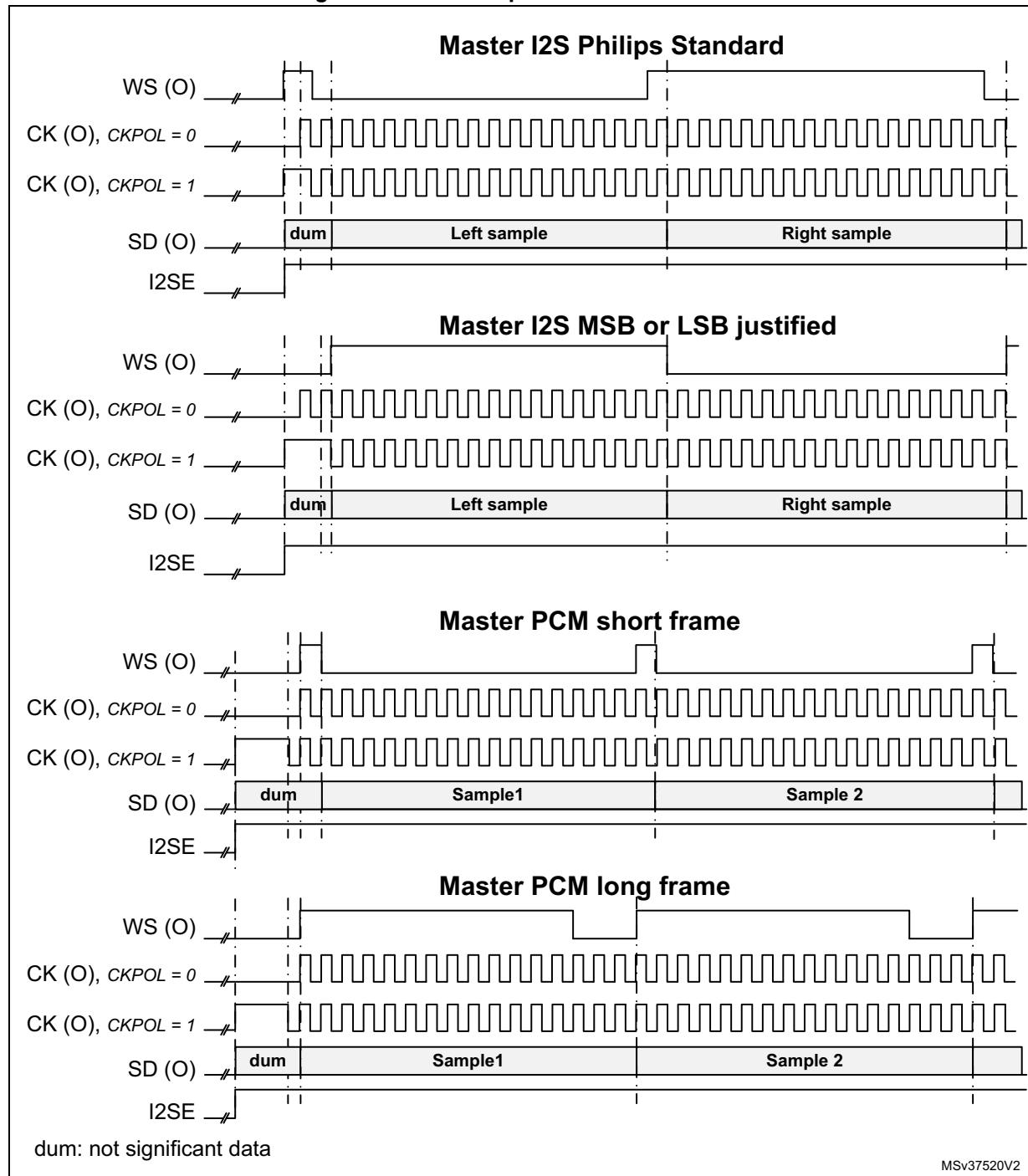
**Figure 307. PCM standard waveforms (16-bit extended to 32-bit packet frame)**

**Note:** For both modes (master and slave) and for both synchronizations (short and long), the number of bits between two consecutive pieces of data (and so two synchronization signals) needs to be specified (DATLEN and CHLEN bits in the SPIx\_I2SCFGR register) even in slave mode.

#### 28.7.4 Start-up description

The [Figure 308](#) shows how the serial interface is handled in MASTER mode, when the SPI/I2S is enabled (via I2SE bit). It shows as well the effect of CKPOL on the generated signals.

Figure 308. Start sequence in master mode



In slave mode, the user has to enable the audio interface before the WS becomes active. This means that the I<sup>2</sup>SE bit must be set to 1 when WS = 1 for I<sup>2</sup>S Philips standard, or when WS = 0 for other standards.

### 28.7.5 Clock generator

The I<sup>2</sup>S bit rate determines the data flow on the I<sup>2</sup>S data line and the I<sup>2</sup>S clock signal frequency.

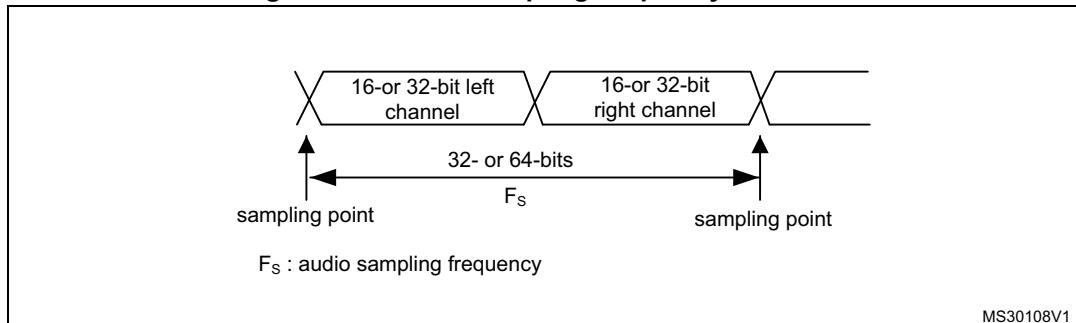
I<sup>2</sup>S bit rate = number of bits per channel × number of channels × sampling audio frequency

For a 16-bit audio, left and right channel, the I<sup>2</sup>S bit rate is calculated as follows:

$$\text{I}^2\text{S bit rate} = 16 \times 2 \times f_s$$

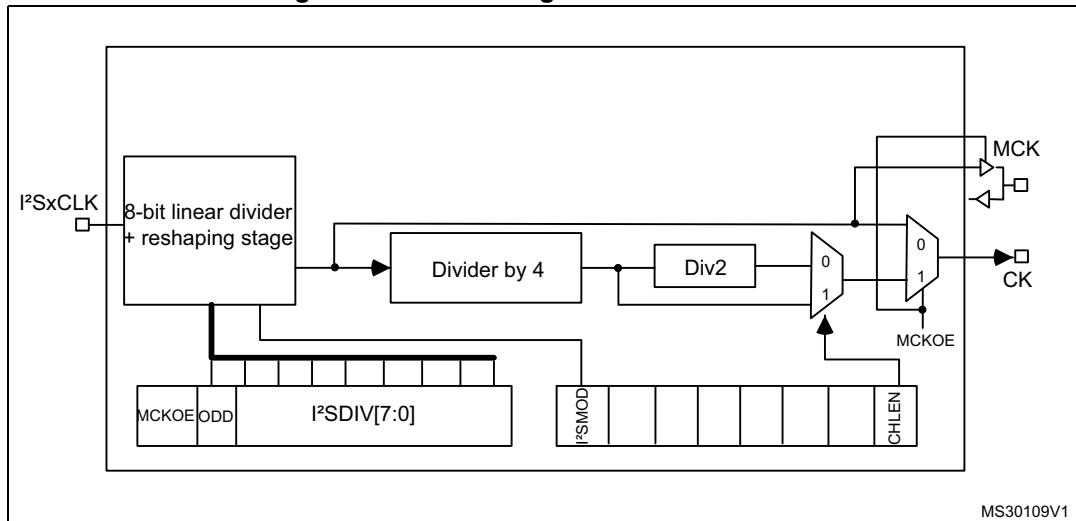
It is: I<sup>2</sup>S bit rate =  $32 \times 2 \times f_s$  if the packet length is 32-bit wide.

**Figure 309. Audio sampling frequency definition**



When the master mode is configured, a specific action needs to be taken to properly program the linear divider in order to communicate with the desired audio frequency.

**Figure 310. I<sup>2</sup>S clock generator architecture**



1. Where x can be 2 or 3.

*Figure 310* presents the communication clock architecture. The I2SxCLK clock is provided by the reset and clock controller (RCC) of the product. The I2SxCLK clock can be asynchronous with respect to the SPI/I2S APB clock.

---

**Warning:** In addition, it is mandatory to keep the I2SxCLK frequency higher or equal to the APB clock used by the SPI/I2S block. If this condition is not respected the SPI/I2S does not work properly.

---

The audio sampling frequency may be 192 kHz, 96 kHz, 48 kHz, 44.1 kHz, 32 kHz, 22.05 kHz, 16 kHz, 11.025 kHz or 8 kHz (or any other value within this range).

In order to reach the desired frequency, the linear divider needs to be programmed according to the formulas below:

### For I<sup>2</sup>S modes:

When the master clock is generated (MCKOE in the SPIx\_I2SPR register is set):

$$F_S = \frac{F_{I2SxCLK}}{256 \times ((2 \times I2SDIV) + ODD)}$$

When the master clock is disabled (MCKOE bit cleared):

$$F_S = \frac{F_{I2SxCLK}}{32 \times (CHLEN + 1) \times ((2 \times I2SDIV) + ODD)}$$

CHLEN = 0 when the channel frame is 16-bit wide and,  
CHLEN = 1 when the channel frame is 32-bit wide.

### For PCM modes:

When the master clock is generated (MCKOE in the SPIx\_I2SPR register is set):

$$F_S = \frac{F_{I2SxCLK}}{128 \times ((2 \times I2SDIV) + ODD)}$$

When the master clock is disabled (MCKOE bit cleared):

$$F_S = \frac{F_{I2SxCLK}}{16 \times (CHLEN + 1) \times ((2 \times I2SDIV) + ODD)}$$

CHLEN = 0 when the channel frame is 16-bit wide and,  
CHLEN = 1 when the channel frame is 32-bit wide.

Where  $F_S$  is the audio sampling frequency, and  $F_{I2SxCLK}$  is the frequency of the kernel clock provided to the SPI/I2S block.

**Note:** I<sub>2</sub>S DIV must be strictly higher than 1.

The following table provides example precision values for different clock configurations.

**Note:** Other configurations are possible that allow optimum clock precision.

**Table 115. Audio-frequency precision using 48 MHz clock derived from HSE<sup>(1)</sup>**

SYSCLK (MHz)	Data length	I <sub>2</sub> S DIV	I <sub>2</sub> S ODD	MCLK	Target fs (Hz)	Real fs (kHz)	Error
48	16	8	0	No	96000	93750	2.3438%
48	32	4	0	No	96000	93750	2.3438%
48	16	15	1	No	48000	48387.0968	0.8065%
48	32	8	0	No	48000	46875	2.3438%
48	16	17	0	No	44100	44117.647	0.0400%
48	32	8	1	No	44100	44117.647	0.0400%
48	16	23	1	No	32000	31914.8936	0.2660%
48	32	11	1	No	32000	32608.696	1.9022%
48	16	34	0	No	22050	22058.8235	0.0400%
48	32	17	0	No	22050	22058.8235	0.0400%
48	16	47	0	No	16000	15957.4468	0.2660%
48	32	23	1	No	16000	15957.447	0.2660%
48	16	68	0	No	11025	11029.4118	0.0400%
48	32	34	0	No	11025	11029.412	0.0400%
48	16	94	0	No	8000	7978.7234	0.2660%
48	32	47	0	No	8000	7978.7234	0.2660%
48	16	2	0	Yes	48000	46875	2.3430%
48	32	2	0	Yes	48000	46875	2.3430%
48	16	2	0	Yes	44100	46875	6.2925%
48	32	2	0	Yes	44100	46875	6.2925%
48	16	3	0	Yes	32000	31250	2.3438%
48	32	3	0	Yes	32000	31250	2.3438%
48	16	4	1	Yes	22050	20833.333	5.5178%
48	32	4	1	Yes	22050	20833.333	5.5178%
48	16	6	0	Yes	16000	15625	2.3438%
48	32	6	0	Yes	16000	15625	2.3438%
48	16	8	1	Yes	11025	11029.4118	0.0400%
48	32	8	1	Yes	11025	11029.4118	0.0400%
48	16	11	1	Yes	8000	8152.17391	1.9022%
48	32	11	1	Yes	8000	8152.17391	1.9022%

1. This table gives only example values for different clock configurations. Other configurations allowing optimum clock precision are possible.

## 28.7.6 I<sup>2</sup>S master mode

The I2S can be configured in master mode. This means that the serial clock is generated on the CK pin as well as the Word Select signal WS. Master clock (MCK) may be output or not, controlled by the MCKOE bit in the SPIx\_I2SPR register.

### Procedure

1. Select the I2SDIV[7:0] bits in the SPIx\_I2SPR register to define the serial clock baud rate to reach the proper audio sample frequency. The ODD bit in the SPIx\_I2SPR register also has to be defined.
2. Select the CKPOL bit to define the steady level for the communication clock. Set the MCKOE bit in the SPIx\_I2SPR register if the master clock MCK needs to be provided to the external DAC/ADC audio component (the I2SDIV and ODD values should be computed depending on the state of the MCK output, for more details refer to [Section 28.7.5: Clock generator](#)).
3. Set the I2SMOD bit in the SPIx\_I2SCFGR register to activate the I2S functions and choose the I<sup>2</sup>S standard through the I2SSTD[1:0] and PCMSYNC bits, the data length through the DATLEN[1:0] bits and the number of bits per channel by configuring the CHLEN bit. Select also the I<sup>2</sup>S master mode and direction (Transmitter or Receiver) through the I2SCFG[1:0] bits in the SPIx\_I2SCFGR register.
4. If needed, select all the potential interrupt sources and the DMA capabilities by writing the SPIx\_CR2 register.
5. The I2SE bit in SPIx\_I2SCFGR register must be set.

WS and CK are configured in output mode. MCK is also an output, if the MCKOE bit in SPIx\_I2SPR is set.

### Transmission sequence

The transmission sequence begins when a half-word is written into the Tx buffer.

Lets assume the first data written into the Tx buffer corresponds to the left channel data. When data are transferred from the Tx buffer to the shift register, TXE is set and data corresponding to the right channel have to be written into the Tx buffer. The CHSIDE flag indicates which channel is to be transmitted. It has a meaning when the TXE flag is set because the CHSIDE flag is updated when TXE goes high.

A full frame has to be considered as a left channel data transmission followed by a right channel data transmission. It is not possible to have a partial frame where only the left channel is sent.

The data half-word is parallel loaded into the 16-bit shift register during the first bit transmission, and then shifted out, serially, to the MOSI/SD pin, MSB first. The TXE flag is set after each transfer from the Tx buffer to the shift register and an interrupt is generated if the TXIE bit in the SPIx\_CR2 register is set.

For more details about the write operations depending on the I<sup>2</sup>S standard mode selected, refer to [Section 28.7.3: Supported audio protocols](#)).

To ensure a continuous audio data transmission, it is mandatory to write the SPIx\_DR register with the next data to transmit before the end of the current transmission.

To switch off the I<sup>2</sup>S, by clearing I2SE, it is mandatory to wait for TXE = 1 and BSY = 0.

### Reception sequence

The operating mode is the same as for transmission mode except for the point 3 (refer to the procedure described in [Section 28.7.6: I<sup>2</sup>S master mode](#)), where the configuration should set the master reception mode through the I2SCFG[1:0] bits.

Whatever the data or channel length, the audio data are received by 16-bit packets. This means that each time the Rx buffer is full, the RXNE flag is set and an interrupt is generated if the RXNEIE bit is set in SPIx\_CR2 register. Depending on the data and channel length configuration, the audio value received for a right or left channel may result from one or two receptions into the Rx buffer.

Clearing the RXNE bit is performed by reading the SPIx\_DR register.

CHSIDE is updated after each reception. It is sensitive to the WS signal generated by the I<sup>2</sup>S cell.

For more details about the read operations depending on the I<sup>2</sup>S standard mode selected, refer to [Section 28.7.3: Supported audio protocols](#).

If data are received while the previously received data have not been read yet, an overrun is generated and the OVR flag is set. If the ERRIE bit is set in the SPIx\_CR2 register, an interrupt is generated to indicate the error.

To switch off the I<sup>2</sup>S, specific actions are required to ensure that the I<sup>2</sup>S completes the transfer cycle properly without initiating a new data transfer. The sequence depends on the configuration of the data and channel lengths, and on the audio protocol mode selected. In the case of:

- 16-bit data length extended on 32-bit channel length (DATLEN = 00 and CHLEN = 1) using the LSB justified mode (I2SSTD = 10)
  - a) Wait for the second to last RXNE = 1 ( $n - 1$ )
  - b) Then wait 17 I<sup>2</sup>S clock cycles (using a software loop)
  - c) Disable the I<sup>2</sup>S (I2SE = 0)
- 16-bit data length extended on 32-bit channel length (DATLEN = 00 and CHLEN = 1) in MSB justified, I<sup>2</sup>S or PCM modes (I2SSTD = 00, I2SSTD = 01 or I2SSTD = 11, respectively)
  - a) Wait for the last RXNE
  - b) Then wait 1 I<sup>2</sup>S clock cycle (using a software loop)
  - c) Disable the I<sup>2</sup>S (I2SE = 0)
- For all other combinations of DATLEN and CHLEN, whatever the audio mode selected through the I2SSTD bits, carry out the following sequence to switch off the I<sup>2</sup>S:
  - a) Wait for the second to last RXNE = 1 ( $n - 1$ )
  - b) Then wait one I<sup>2</sup>S clock cycle (using a software loop)
  - c) Disable the I<sup>2</sup>S (I2SE = 0)

*Note:* The BSY flag is kept low during transfers.

### 28.7.7 I<sup>2</sup>S slave mode

For the slave configuration, the I<sup>2</sup>S can be configured in transmission or reception mode. The operating mode is following mainly the same rules as described for the I<sup>2</sup>S master

configuration. In slave mode, there is no clock to be generated by the I2S interface. The clock and WS signals are input from the external master connected to the I2S interface. There is then no need, for the user, to configure the clock.

The configuration steps to follow are listed below:

1. Set the I2SMOD bit in the SPIx\_I2SCFGR register to select I<sup>2</sup>S mode and choose the I<sup>2</sup>S standard through the I2SSTD[1:0] bits, the data length through the DATLEN[1:0] bits and the number of bits per channel for the frame configuring the CHLEN bit. Select also the mode (transmission or reception) for the slave through the I2SCFG[1:0] bits in SPIx\_I2SCFGR register.
2. If needed, select all the potential interrupt sources and the DMA capabilities by writing the SPIx\_CR2 register.
3. The I2SE bit in SPIx\_I2SCFGR register must be set.

### Transmission sequence

The transmission sequence begins when the external master device sends the clock and when the NSS\_WS signal requests the transfer of data. The slave has to be enabled before the external master starts the communication. The I2S data register has to be loaded before the master initiates the communication.

For the I2S, MSB justified and LSB justified modes, the first data item to be written into the data register corresponds to the data for the left channel. When the communication starts, the data are transferred from the Tx buffer to the shift register. The TXE flag is then set in order to request the right channel data to be written into the I2S data register.

The CHSIDE flag indicates which channel is to be transmitted. Compared to the master transmission mode, in slave mode, CHSIDE is sensitive to the WS signal coming from the external master. This means that the slave needs to be ready to transmit the first data before the clock is generated by the master. WS assertion corresponds to left channel transmitted first.

*Note:* *The I2SE has to be written at least two PCLK cycles before the first clock of the master comes on the CK line.*

The data half-word is parallel-loaded into the 16-bit shift register (from the internal bus) during the first bit transmission, and then shifted out serially to the MOSI/SD pin MSB first. The TXE flag is set after each transfer from the Tx buffer to the shift register and an interrupt is generated if the TXEIE bit in the SPIx\_CR2 register is set.

Note that the TXE flag should be checked to be at 1 before attempting to write the Tx buffer.

For more details about the write operations depending on the I<sup>2</sup>S standard mode selected, refer to [Section 28.7.3: Supported audio protocols](#).

To secure a continuous audio data transmission, it is mandatory to write the SPIx\_DR register with the next data to transmit before the end of the current transmission. An underrun flag is set and an interrupt may be generated if the data are not written into the SPIx\_DR register before the first clock edge of the next data communication. This indicates to the software that the transferred data are wrong. If the ERRIE bit is set into the SPIx\_CR2 register, an interrupt is generated when the UDR flag in the SPIx\_SR register goes high. In this case, it is mandatory to switch off the I2S and to restart a data transfer starting from the left channel.

To switch off the I2S, by clearing the I2SE bit, it is mandatory to wait for TXE = 1 and BSY = 0.

## Reception sequence

The operating mode is the same as for the transmission mode except for the point 1 (refer to the procedure described in [Section 28.7.7: I<sup>2</sup>S slave mode](#)), where the configuration should set the master reception mode using the I2SCFG[1:0] bits in the SPIx\_I2SCFGR register.

Whatever the data length or the channel length, the audio data are received by 16-bit packets. This means that each time the RX buffer is full, the RXNE flag in the SPIx\_SR register is set and an interrupt is generated if the RXNEIE bit is set in the SPIx\_CR2 register. Depending on the data length and channel length configuration, the audio value received for a right or left channel may result from one or two receptions into the RX buffer.

The CHSIDE flag is updated each time data are received to be read from the SPIx\_DR register. It is sensitive to the external WS line managed by the external master component.

Clearing the RXNE bit is performed by reading the SPIx\_DR register.

For more details about the read operations depending the I<sup>2</sup>S standard mode selected, refer to [Section 28.7.3: Supported audio protocols](#).

If data are received while the preceding received data have not yet been read, an overrun is generated and the OVR flag is set. If the bit ERRIE is set in the SPIx\_CR2 register, an interrupt is generated to indicate the error.

To switch off the I<sup>2</sup>S in reception mode, I2SE has to be cleared immediately after receiving the last RXNE = 1.

*Note:* *The external master components should have the capability of sending/receiving data in 16-bit or 32-bit packets via an audio channel.*

## 28.7.8 I<sup>2</sup>S status flags

Three status flags are provided for the application to fully monitor the state of the I<sup>2</sup>S bus.

### Busy flag (BSY)

The BSY flag is set and cleared by hardware (writing to this flag has no effect). It indicates the state of the communication layer of the I<sup>2</sup>S.

When BSY is set, it indicates that the I<sup>2</sup>S is busy communicating. There is one exception in master receive mode (I2SCFG = 11) where the BSY flag is kept low during reception.

The BSY flag is useful to detect the end of a transfer if the software needs to disable the I<sup>2</sup>S. This avoids corrupting the last transfer. For this, the procedure described below must be strictly respected.

The BSY flag is set when a transfer starts, except when the I<sup>2</sup>S is in master receiver mode.

The BSY flag is cleared:

- When a transfer completes (except in master transmit mode, in which the communication is supposed to be continuous)
- When the I<sup>2</sup>S is disabled

When communication is continuous:

- In master transmit mode, the BSY flag is kept high during all the transfers
- In slave mode, the BSY flag goes low for one I<sup>2</sup>S clock cycle between each transfer

*Note:* *Do not use the BSY flag to handle each data transmission or reception. It is better to use the TXE and RXNE flags instead.*

### Tx buffer empty flag (TXE)

When set, this flag indicates that the Tx buffer is empty and the next data to be transmitted can then be loaded into it. The TXE flag is reset when the Tx buffer already contains data to be transmitted. It is also reset when the I2S is disabled (I2SE bit is reset).

### RX buffer not empty (RXNE)

When set, this flag indicates that there are valid received data in the RX Buffer. It is reset when SPIx\_DR register is read.

### Channel Side flag (CHSIDE)

In transmission mode, this flag is refreshed when TXE goes high. It indicates the channel side to which the data to transfer on SD has to belong. In case of an underrun error event in slave transmission mode, this flag is not reliable and I2S needs to be switched off and switched on before resuming the communication.

In reception mode, this flag is refreshed when data are received into SPIx\_DR. It indicates from which channel side data have been received. Note that in case of error (like OVR) this flag becomes meaningless and the I2S should be reset by disabling and then enabling it (with configuration if it needs changing).

This flag has no meaning in the PCM standard (for both Short and Long frame modes).

When the OVR or UDR flag in the SPIx\_SR is set and the ERRIE bit in SPIx\_CR2 is also set, an interrupt is generated. This interrupt can be cleared by reading the SPIx\_SR status register (once the interrupt source has been cleared).

## 28.7.9 I2S error flags

There are three error flags for the I2S cell.

### Underrun flag (UDR)

In slave transmission mode this flag is set when the first clock for data transmission appears while the software has not yet loaded any value into SPIx\_DR. It is available when the I2SMOD bit in the SPIx\_I2SCFGR register is set. An interrupt may be generated if the ERRIE bit in the SPIx\_CR2 register is set.

The UDR bit is cleared by a read operation on the SPIx\_SR register.

### Overrun flag (OVR)

This flag is set when data are received and the previous data have not yet been read from the SPIx\_DR register. As a result, the incoming data are lost. An interrupt may be generated if the ERRIE bit is set in the SPIx\_CR2 register.

In this case, the receive buffer contents are not updated with the newly received data from the transmitter device. A read operation to the SPIx\_DR register returns the previous correctly received data. All other subsequently transmitted half-words are lost.

Clearing the OVR bit is done by a read operation on the SPIx\_DR register followed by a read access to the SPIx\_SR register.

### Frame error flag (FRE)

This flag can be set by hardware only if the I2S is configured in Slave mode. It is set if the external master is changing the WS line while the slave is not expecting this change. If the

synchronization is lost, the following steps are required to recover from this state and resynchronize the external master device with the I<sup>2</sup>S slave device:

1. Disable the I<sup>2</sup>S.
2. Enable it again when the correct level is detected on the WS line (WS line is high in I<sup>2</sup>S mode or low for MSB- or LSB-justified or PCM modes).

Desynchronization between master and slave devices may be due to noisy environment on the CK communication clock or on the WS frame synchronization line. An error interrupt can be generated if the ERRIE bit is set. The desynchronization flag (FRE) is cleared by software when the status register is read.

### 28.7.10 DMA features

In I<sup>2</sup>S mode, the DMA works in exactly the same way as it does in SPI mode. There is no difference except that the CRC feature is not available in I<sup>2</sup>S mode since there is no data transfer protection system.

## 28.8 I<sup>2</sup>S interrupts

*Table 116* provides the list of I<sup>2</sup>S interrupts.

**Table 116. I<sup>2</sup>S interrupt requests**

Interrupt event	Event flag	Enable control bit
Transmit buffer empty flag	TXE	TXEIE
Receive buffer not empty flag	RXNE	RXNEIE
Overrun error	OVR	ERRIE
Underrun error	UDR	
Frame error flag	FRE	

## 28.9 SPI and I2S registers

The peripheral registers can be accessed by half-words (16-bit) or words (32-bit). SPI\_DR in addition can be accessed by 8-bit access.

### 28.9.1 SPI control register 1 (SPIx\_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIDI MODE	BIDIOE	CRC EN	CRCN EXT	CRCL	RX ONLY	SSM	SSI	LSB FIRST	SPE	BR[2:0]			MSTR	CPOL	CPHA
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 **BIDI MODE**: Bidirectional data mode enable.

This bit enables half-duplex communication using common single bidirectional data line.  
Keep RXONLY bit clear when bidirectional mode is active.

0: 2-line unidirectional data mode selected

1: 1-line bidirectional data mode selected

*Note: This bit is not used in I<sup>2</sup>S mode.*

Bit 14 **BIDIOE**: Output enable in bidirectional mode

This bit combined with the BIDI MODE bit selects the direction of transfer in bidirectional mode.  
0: Output disabled (receive-only mode)  
1: Output enabled (transmit-only mode)

*Note: In master mode, the MOSI pin is used and in slave mode, the MISO pin is used.  
This bit is not used in I<sup>2</sup>S mode.*

Bit 13 **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.  
This bit is not used in I<sup>2</sup>S mode.*

Bit 12 **CRCNEXT**: Transmit CRC next

0: Next transmit value is from Tx buffer.  
1: Next transmit value is from Tx CRC register.

*Note: This bit has to be written as soon as the last data is written in the SPIx\_DR register.  
This bit is not used in I<sup>2</sup>S mode.*

Bit 11 **CRCL**: CRC length

This bit is set and cleared by software to select the CRC length.  
0: 8-bit CRC length  
1: 16-bit CRC length

*Note: This bit should be written only when SPI is disabled (SPE = '0') for correct operation.  
This bit is not used in I<sup>2</sup>S mode.*

Bit 10 **RXONLY:** Receive only mode enabled.

This bit enables simplex communication using a single unidirectional line to receive data exclusively. Keep BIDIMODE bit clear when receive only mode is active. This bit is also useful in a multislave system in which this particular slave is not accessed, the output from the accessed slave is not corrupted.

- 0: Full-duplex (Transmit and receive)
- 1: Output disabled (Receive-only mode)

*Note: This bit is not used in I<sup>2</sup>S mode.*

Bit 9 **SSM:** Software slave management

When the SSM bit is set, the NSS pin input is replaced with the value from the SSI bit.

- 0: Software slave management disabled
- 1: Software slave management enabled

*Note: This bit is not used in I<sup>2</sup>S mode and SPI TI mode.*

Bit 8 **SSI:** Internal slave select

This bit has an effect only when the SSM bit is set. The value of this bit is forced onto the NSS pin and the I/O value of the NSS pin is ignored.

*Note: This bit is not used in I<sup>2</sup>S mode and SPI TI mode.*

Bit 7 **LSBFIRST:** Frame format

- 0: data is transmitted / received with the MSB first
- 1: data is transmitted / received with the LSB first

*Note: 1. This bit should not be changed when communication is ongoing.*

*2. This bit is not used in I<sup>2</sup>S mode and SPI TI mode.*

Bit 6 **SPE:** SPI enable

- 0: Peripheral disabled
- 1: Peripheral enabled

*Note: When disabling the SPI, follow the procedure described in [Procedure for disabling the SPI on page 779](#).*

*This bit is not used in I<sup>2</sup>S mode.*

Bits 5:3 **BR[2:0]:** Baud rate control

- 000: f<sub>PCLK</sub>/2
- 001: f<sub>PCLK</sub>/4
- 010: f<sub>PCLK</sub>/8
- 011: f<sub>PCLK</sub>/16
- 100: f<sub>PCLK</sub>/32
- 101: f<sub>PCLK</sub>/64
- 110: f<sub>PCLK</sub>/128
- 111: f<sub>PCLK</sub>/256

*Note: These bits should not be changed when communication is ongoing.*

*These bits are not used in I<sup>2</sup>S mode.*

Bit 2 **MSTR:** Master selection

- 0: Slave configuration
- 1: Master configuration

*Note: This bit should not be changed when communication is ongoing.*

*This bit is not used in I<sup>2</sup>S mode.*

Bit 1 **CPOL:** Clock polarity

- 0: CK to 0 when idle
- 1: CK to 1 when idle

*Note: This bit should not be changed when communication is ongoing.*

*This bit is not used in I<sup>2</sup>S mode and SPI TI mode except the case when CRC is applied at TI mode.*

Bit 0 **CPHA:** Clock phase

- 0: The first clock transition is the first data capture edge
- 1: The second clock transition is the first data capture edge

*Note: This bit should not be changed when communication is ongoing.*

*This bit is not used in I<sup>2</sup>S mode and SPI TI mode except the case when CRC is applied at TI mode.*

## 28.9.2 SPI control register 2 (SPIx\_CR2)

Address offset: 0x04

Reset value: 0x0700

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LDMA_TX	LDMA_RX	FRXTH	DS[3:0]				TXEIE	RXNEIE	ERRIE	FRF	NSSP	SSOE	TXDMAEN	RXDMAEN
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 Reserved, must be kept at reset value.

Bit 14 **LDMA\_TX:** Last DMA transfer for transmission

This bit is used in data packing mode, to define if the total number of data to transmit by DMA is odd or even. It has significance only if the TXDMAEN bit in the SPIx\_CR2 register is set and if packing mode is used (data length <= 8-bit and write access to SPIx\_DR is 16-bit wide). It has to be written when the SPI is disabled (SPE = 0 in the SPIx\_CR1 register).

- 0: Number of data to transfer is even
- 1: Number of data to transfer is odd

*Note: Refer to [Procedure for disabling the SPI on page 779](#) if the CRCEN bit is set.*

*This bit is not used in I<sup>2</sup>S mode.*

Bit 13 **LDMA\_RX:** Last DMA transfer for reception

This bit is used in data packing mode, to define if the total number of data to receive by DMA is odd or even. It has significance only if the RXDMAEN bit in the SPIx\_CR2 register is set and if packing mode is used (data length <= 8-bit and write access to SPIx\_DR is 16-bit wide). It has to be written when the SPI is disabled (SPE = 0 in the SPIx\_CR1 register).

- 0: Number of data to transfer is even
- 1: Number of data to transfer is odd

*Note: Refer to [Procedure for disabling the SPI on page 779](#) if the CRCEN bit is set.*

*This bit is not used in I<sup>2</sup>S mode.*

Bit 12 **FRXTH:** FIFO reception threshold

This bit is used to set the threshold of the RXFIFO that triggers an RXNE event

- 0: RXNE event is generated if the FIFO level is greater than or equal to 1/2 (16-bit)
- 1: RXNE event is generated if the FIFO level is greater than or equal to 1/4 (8-bit)

*Note: This bit is not used in I<sup>2</sup>S mode.*

Bits 11:8 **DS[3:0]**: Data size

These bits configure the data length for SPI transfers.

- 0000: Not used
- 0001: Not used
- 0010: Not used
- 0011: 4-bit
- 0100: 5-bit
- 0101: 6-bit
- 0110: 7-bit
- 0111: 8-bit
- 1000: 9-bit
- 1001: 10-bit
- 1010: 11-bit
- 1011: 12-bit
- 1100: 13-bit
- 1101: 14-bit
- 1110: 15-bit
- 1111: 16-bit

If software attempts to write one of the “Not used” values, they are forced to the value “0111” (8-bit)

*Note: These bits are not used in I<sup>2</sup>S mode.*

Bit 7 **TXEIE**: Tx buffer empty interrupt enable

0: TXE interrupt masked

1: TXE interrupt not masked. Used to generate an interrupt request when the TXE flag is set.

Bit 6 **RXNEIE**: RX buffer not empty interrupt enable

0: RXNE interrupt masked

1: RXNE interrupt not masked. Used to generate an interrupt request when the RXNE flag is set.

Bit 5 **ERRIE**: Error interrupt enable

This bit controls the generation of an interrupt when an error condition occurs (CRCERR, OVR, MODF in SPI mode, FRE at TI mode and UDR, OVR, and FRE in I<sup>2</sup>S mode).

0: Error interrupt is masked

1: Error interrupt is enabled

Bit 4 **FRF**: Frame format

0: SPI Motorola mode

1 SPI TI mode

*Note: This bit must be written only when the SPI is disabled (SPE=0).*

*This bit is not used in I<sup>2</sup>S mode.*

Bit 3 **NSSP**: NSS pulse management

This bit is used in master mode only. It allows the SPI to generate an NSS pulse between two consecutive data when doing continuous transfers. In the case of a single data transfer, it forces the NSS pin high level after the transfer.

It has no meaning if CPHA = '1', or FRF = '1'.

0: No NSS pulse

1: NSS pulse generated

*Note: 1. This bit must be written only when the SPI is disabled (SPE=0).*

*2. This bit is not used in I<sup>2</sup>S mode and SPI TI mode.*

Bit 2 **SSOE:** SS output enable

0: SS output is disabled in master mode and the SPI interface can work in multimaster configuration

1: SS output is enabled in master mode and when the SPI interface is enabled. The SPI interface cannot work in a multimaster environment.

*Note: This bit is not used in I<sup>2</sup>S mode and SPI TI mode.*

Bit 1 **TXDMAEN:** Tx buffer DMA enable

When this bit is set, a DMA request is generated whenever the TXE flag is set.

0: Tx buffer DMA disabled

1: Tx buffer DMA enabled

Bit 0 **RXDMAEN:** Rx buffer DMA enable

When this bit is set, a DMA request is generated whenever the RXNE flag is set.

0: Rx buffer DMA disabled

1: Rx buffer DMA enabled

### 28.9.3 SPI status register (SPIx\_SR)

Address offset: 0x08

Reset value: 0x0002

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	FTLVL[1:0]		FRLVL[1:0]		FRE	BSY	OVR	MODF	CRCE RR	UDR	CHSIDE	TXE	RXNE
			r	r	r	r	r	r	r	r	rc_w0	r	r	r	r

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

Bits 12:11 **FTLVL[1:0]:** FIFO transmission level

These bits are set and cleared by hardware.

00: FIFO empty

01: 1/4 FIFO

10: 1/2 FIFO

11: FIFO full (considered as FULL when the FIFO threshold is greater than 1/2)

*Note: This bit is not used in I<sup>2</sup>S mode.*

Bits 10:9 **FRLVL[1:0]:** FIFO reception level

These bits are set and cleared by hardware.

00: FIFO empty

01: 1/4 FIFO

10: 1/2 FIFO

11: FIFO full

*Note: These bits are not used in I<sup>2</sup>S mode and in SPI receive-only mode while CRC calculation is enabled.*

Bit 8 **FRE:** Frame format error

This flag is used for SPI in TI slave mode and I<sup>2</sup>S slave mode. Refer to [Section 28.5.11: SPI error flags](#) and [Section 28.7.9: I2S error flags](#).

This flag is set by hardware and reset when SPIx\_SR is read by software.

0: No frame format error

1: A frame format error occurred

**Bit 7 BSY:** Busy flag

- 0: SPI (or I2S) not busy
  - 1: SPI (or I2S) is busy in communication or Tx buffer is not empty
- This flag is set and cleared by hardware.

*Note: The BSY flag must be used with caution: refer to [Section 28.5.10: SPI status flags and Procedure for disabling the SPI](#) on page 779.*

**Bit 6 OVR:** Overrun flag

- 0: No overrun occurred
  - 1: Overrun occurred
- This flag is set by hardware and reset by a software sequence. Refer to [I2S error flags on page 812](#) for the software sequence.

**Bit 5 MODF:** Mode fault

- 0: No mode fault occurred
- 1: Mode fault occurred

This flag is set by hardware and reset by a software sequence. Refer to [Section : Mode fault \(MODF\) on page 789](#) for the software sequence.

*Note: This bit is not used in I<sup>2</sup>S mode.*

**Bit 4 CRCERR:** CRC error flag

- 0: CRC value received matches the SPIx\_RXCRCR value
- 1: CRC value received does not match the SPIx\_RXCRCR value

*Note: This flag is set by hardware and cleared by software writing 0.*

*This bit is not used in I<sup>2</sup>S mode.*

**Bit 3 UDR:** Underrun flag

- 0: No underrun occurred
- 1: Underrun occurred

This flag is set by hardware and reset by a software sequence. Refer to [I2S error flags on page 812](#) for the software sequence.

*Note: This bit is not used in SPI mode.*

**Bit 2 CHSIDE:** Channel side

- 0: Channel Left has to be transmitted or has been received
- 1: Channel Right has to be transmitted or has been received

*Note: This bit is not used in SPI mode. It has no significance in PCM mode.*

**Bit 1 TXE:** Transmit buffer empty

- 0: Tx buffer not empty
- 1: Tx buffer empty

**Bit 0 RXNE:** Receive buffer not empty

- 0: Rx buffer empty
- 1: Rx buffer not empty

### 28.9.4 SPI data register (SPIx\_DR)

Address offset: 0x0C

Reset value: 0x0000

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

Bits 15:0 **DR[15:0]**: Data register

Data received or to be transmitted

The data register serves as an interface between the Rx and Tx FIFOs. When the data register is read, RxFIFO is accessed while the write to data register accesses TxFIFO (See [Section 28.5.9: Data transmission and reception procedures](#)).

*Note:* *Data is always right-aligned. Unused bits are ignored when writing to the register, and read as zero when the register is read. The Rx threshold setting must always correspond with the read access currently used.*

### 28.9.5 SPI CRC polynomial register (SPIx\_CRCPR)

Address offset: 0x10

Reset value: 0x0007

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

Bits 15:0 **CRCPOLY[15:0]**: CRC polynomial register

This register contains the polynomial for the CRC calculation.

The CRC polynomial (0x0007) is the reset value of this register. Another polynomial can be configured as required.

*Note:* *The polynomial value should be odd only. No even value is supported.*

### 28.9.6 SPI Rx CRC register (SPIx\_RXCRCR)

Address offset: 0x14

Reset value: 0x0000

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

Bits 15:0 **RXCRC[15:0]**: Rx CRC register

When CRC calculation is enabled, the RXCRC[15:0] bits contain the computed CRC value of the subsequently received bytes. This register is reset when the CRCEN bit in SPIx\_CR1 register is written to 1. The CRC is calculated serially using the polynomial programmed in the SPIx\_CRCPR register.

Only the 8 LSB bits are considered when the CRC frame format is set to be 8-bit length (CRCL bit in the SPIx\_CR1 is cleared). CRC calculation is done based on any CRC8 standard.

The entire 16-bits of this register are considered when a 16-bit CRC frame format is selected (CRCL bit in the SPIx\_CR1 register is set). CRC calculation is done based on any CRC16 standard.

*Note: A read to this register when the BSY Flag is set could return an incorrect value.*

*These bits are not used in I<sup>2</sup>S mode.*

**28.9.7 SPI Tx CRC register (SPIx\_TXCRCR)**

Address offset: 0x18

Reset value: 0x0000

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

Bits 15:0 **TXCRC[15:0]**: Tx CRC register

When CRC calculation is enabled, the TXCRC[7:0] bits contain the computed CRC value of the subsequently transmitted bytes. This register is reset when the CRCEN bit of SPIx\_CR1 is written to 1. The CRC is calculated serially using the polynomial programmed in the SPIx\_CRCPR register.

Only the 8 LSB bits are considered when the CRC frame format is set to be 8-bit length (CRCL bit in the SPIx\_CR1 is cleared). CRC calculation is done based on any CRC8 standard.

The entire 16-bits of this register are considered when a 16-bit CRC frame format is selected (CRCL bit in the SPIx\_CR1 register is set). CRC calculation is done based on any CRC16 standard.

*Note: A read to this register when the BSY flag is set could return an incorrect value.*

*These bits are not used in I<sup>2</sup>S mode.*

**28.9.8 SPIx\_I2S configuration register (SPIx\_I2SCFGR)**

Address offset: 0x1C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.		I2SMOD	I2SE	I2SCFG[1:0]		PCMSYNC	Res.	I2SSSTD[1:0]		CKPOL	DATLEN[1:0]		CHLEN
				rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw

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

Bit 11 **I2SMOD**: I2S mode selection

- 0: SPI mode is selected
- 1: I2S mode is selected

*Note: This bit should be configured when the SPI or I2S is disabled.*

Bit 10 **I2SE**: I2S enable

- 0: I2S peripheral is disabled
- 1: I2S peripheral is enabled

*Note: This bit is not used in SPI mode.*

Bits 9:8 **I2SCFG[1:0]**: I2S configuration mode

- 00: Slave - transmit
- 01: Slave - receive
- 10: Master - transmit
- 11: Master - receive

*Note: These bits should be configured when the I2S is disabled.*

*They are not used in SPI mode.*

Bit 7 **PCMSYNC**: PCM frame synchronization

- 0: Short frame synchronization
- 1: Long frame synchronization

*Note: This bit has a meaning only if I2SSTD = 11 (PCM standard is used).*

*It is not used in SPI mode.*

Bit 6 Reserved, must be kept at reset value.

Bits 5:4 **I2SSTD[1:0]**: I2S standard selection

- 00: I<sup>2</sup>S Philips standard
- 01: MSB justified standard (left justified)
- 10: LSB justified standard (right justified)
- 11: PCM standard

For more details on I<sup>2</sup>S standards, refer to [Section 28.7.3 on page 796](#)

*Note: For correct operation, these bits should be configured when the I2S is disabled.*

*They are not used in SPI mode.*

Bit 3 **CKPOL**: Inactive state clock polarity

- 0: I2S clock inactive state is low level
- 1: I2S clock inactive state is high level

*Note: For correct operation, this bit should be configured when the I2S is disabled.*

*It is not used in SPI mode.*

*The bit CKPOL does not affect the CK edge sensitivity used to receive or transmit the SD and WS signals.*

Bits 2:1 **DATLEN[1:0]**: Data length to be transferred

- 00: 16-bit data length
- 01: 24-bit data length
- 10: 32-bit data length
- 11: Not allowed

*Note: For correct operation, these bits should be configured when the I2S is disabled.*

*They are not used in SPI mode.*

Bit 0 **CHLEN**: Channel length (number of bits per audio channel)

- 0: 16-bit wide
- 1: 32-bit wide

The bit write operation has a meaning only if DATLEN = 00 otherwise the channel length is fixed to 32-bit by hardware whatever the value filled in.

*Note: For correct operation, this bit should be configured when the I2S is disabled.*

*It is not used in SPI mode.*

### 28.9.9 SPIx\_I2S prescaler register (SPIx\_I2SPR)

Address offset: 0x20

Reset value: 0x0002

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
Res.	Res.	Res.	Res.	Res.	Res.	MCKOE	ODD	I2SDIV[7:0]													
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw						

Bits 15:10 Reserved, must be kept at reset value.

Bit 9 **MCKOE**: Master clock output enable

- 0: Master clock output is disabled
- 1: Master clock output is enabled

*Note: This bit should be configured when the I2S is disabled. It is used only when the I2S is in master mode.*

*It is not used in SPI mode.*

Bit 8 **ODD**: Odd factor for the prescaler

- 0: Real divider value is = I2SDIV \*2
- 1: Real divider value is = (I2SDIV \* 2) + 1

Refer to [Section 28.7.4 on page 803](#).

*Note: This bit should be configured when the I2S is disabled. It is used only when the I2S is in master mode.*

*It is not used in SPI mode.*

Bits 7:0 **I2SDIV[7:0]**: I2S linear prescaler

I2SDIV [7:0] = 0 or I2SDIV [7:0] = 1 are forbidden values.

Refer to [Section 28.7.4 on page 803](#).

*Note: These bits should be configured when the I2S is disabled. They are used only when the I2S is in master mode.*

*They are not used in SPI mode.*

### 28.9.10 SPI/I2S register map

*Table 117* shows the SPI/I2S register map and reset values.

**Table 117. SPI/I2S register map and reset values**

Offset	Register	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	SPIx_CR1	BIDIMODE	0	BIDIOE	0	CRCEN	0	CRCNEXT	0	CRCL	0	RXONLY	0	SSM	0	SSI	0
	Reset value	0	0	LDMA_RX	0	FRXTH	0	DS[3:0]	0	TXEIE	0	LSBFIRST	0	BR [2:0]	0	0	0
0x04	SPIx_CR2	Res.	0	LDMA_TX	0	FTLVL[1:0]	0	FRLVL[1:0]	1	BSY	0	RXNEIE	0	SPE	0	0	0
	Reset value	0	0	0	0	0	0	0	0	0	0	MODF	0	ERRIE	0	0	0
0x08	SPIx_SR	Res.	0	FTLVL[1:0]	0	FRLVL[1:0]	0	FRE	0	QVR	0	CRCEFR	0	FRF	0	0	0
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	SPIx_DR	DR[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	SPIx_CRCPR	CRCPOLY[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
0x14	SPIx_RXCRCR	RXCRC[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	SPIx_TXCRCR	TXCRC[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1C	SPIx_I2SCFGR	Res.	Res.	I2SMOD	0	I2SE	0	I2SCFG[1:0]	0	PCMSYNC	0	I2STD	0	CKPOL	0	DATLEN[1:0]	0
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x20	SPIx_I2SPR	Res.	Res.	MCKOE	0	ODD	0	0	0	0	0	0	0	TXE	0	TXDMAEN	0
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	RXNE	0	RXDMAEN	0

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 29 Controller area network (bxCAN)

### 29.1 Introduction

The **Basic Extended CAN** peripheral, named **bxCAN**, interfaces the CAN network. It supports the CAN protocols version 2.0A and B. It has been designed to manage a high number of incoming messages efficiently with a minimum CPU load. It also meets the priority requirements for transmit messages.

For safety-critical applications, the CAN controller provides all hardware functions for supporting the CAN Time Triggered Communication option.

### 29.2 bxCAN main features

- Supports CAN protocol version 2.0 A, B Active
- Bit rates up to 1 Mbit/s
- Supports the Time Triggered Communication option

Transmission

- Three transmit mailboxes
- Configurable transmit priority
- Time stamp on SOF transmission

Reception

- Two receive FIFOs with three stages
- Scalable filter banks:
  - 14 filter banks for single CAN
- Identifier list feature
- Configurable FIFO overrun
- Time stamp on SOF reception

Time-triggered communication option

- Disable automatic retransmission mode
- 16-bit free running timer
- Time stamp sent in last two data bytes

Management

- Maskable interrupts
- Software-efficient mailbox mapping at a unique address space

### 29.3 bxCAN general description

In today 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 (to be handled by each node) has significantly increased. In addition to the application messages, network management and diagnostic messages have been introduced.

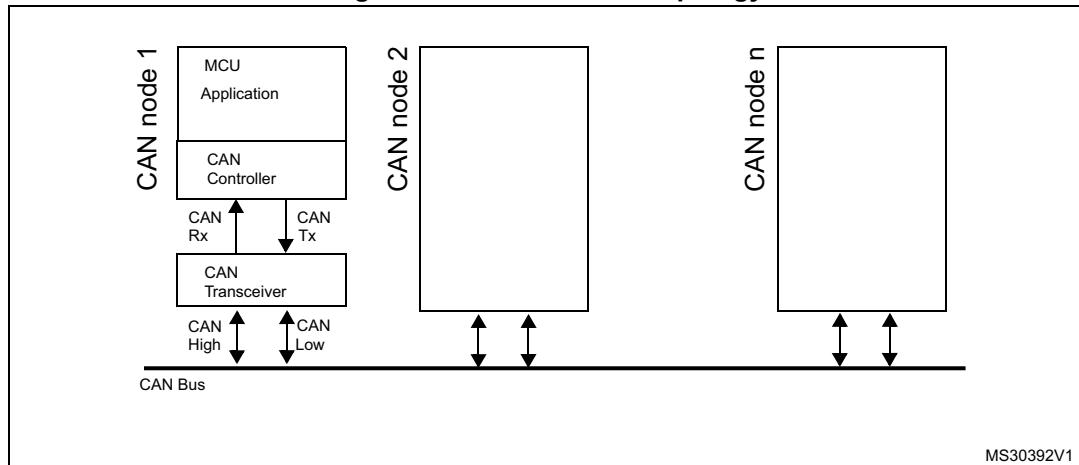
- An enhanced filtering mechanism is required to handle each type of message.

Furthermore, application tasks require more CPU time, therefore real-time constraints caused by message reception have to be reduced.

- A receive FIFO scheme allows the CPU to be dedicated to application tasks for a long time period without losing messages.

The standard HLP (Higher Layer Protocol) based on standard CAN drivers requires an efficient interface to the CAN controller.

**Figure 311. CAN network topology**



### 29.3.1 CAN 2.0B active core

The bxCAN module handles the transmission and the reception of CAN messages fully autonomously. Standard identifiers (11-bit) and extended identifiers (29-bit) are fully supported by hardware.

### 29.3.2 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

### 29.3.3 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.

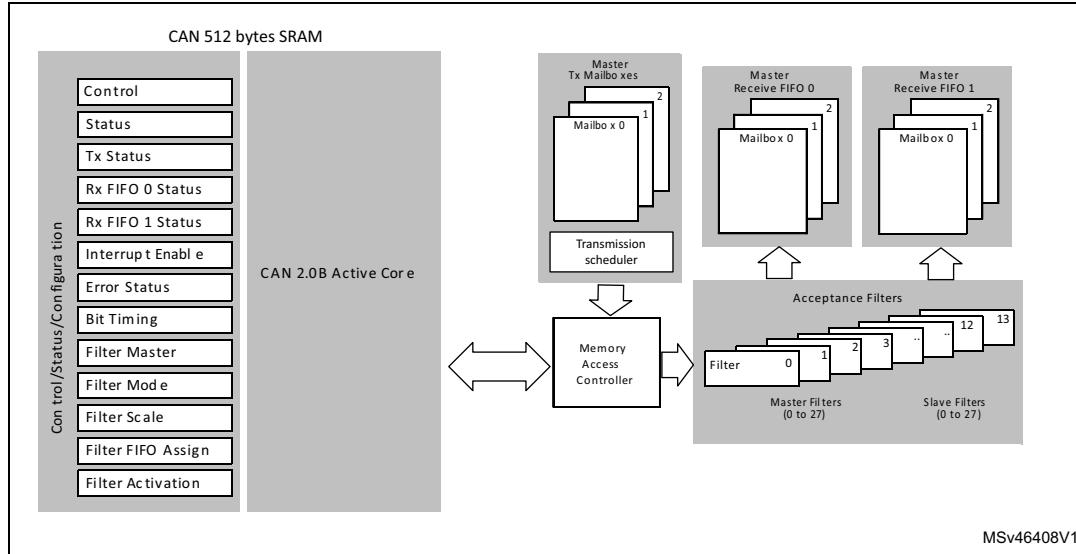
### 29.3.4 Acceptance filters

The bxCAN provides up to 14 scalable/configurable identifier filter banks in single CAN configuration, for selecting the incoming messages, that the software needs and discarding the others.

## Receive FIFO

Two receive FIFOs are used by hardware to store the incoming messages. Three complete messages can be stored in each FIFO. The FIFOs are managed completely by hardware.

**Figure 312. Single-CAN block diagram**



## 29.4 bxCAN operating modes

bxCAN has three main operating modes: **initialization**, **normal** and **Sleep**. After a hardware reset, bxCAN is in Sleep mode to reduce power consumption and an internal pull-up is active on CANTX. The software requests bxCAN to enter **initialization** or **Sleep** mode by setting the INRQ or SLEEP bits in the CAN\_MCR register. Once the mode has been entered, bxCAN confirms it by setting the INAK or SLAK bits in the CAN\_MSR register and the internal pull-up is disabled. When neither INAK nor SLAK are set, bxCAN is in **normal** mode. Before entering **normal** mode bxCAN always has to **synchronize** on the CAN bus. To synchronize, bxCAN waits until the CAN bus is idle, this means 11 consecutive recessive bits have been monitored on CANRX.

### 29.4.1 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. bxCAN has left Initialization mode once the INAK bit has been cleared by hardware.

While in Initialization Mode, all message transfers to and from the CAN bus are stopped and the status of the CAN bus output CANTX is recessive (high).

Entering Initialization Mode does not change any of the configuration registers.

To initialize the CAN Controller, software has to set up the Bit Timing (CAN\_BTR) and CAN options (CAN\_MCR) registers.

To initialize the registers associated with the CAN filter banks (mode, scale, FIFO assignment, activation and filter values), software has to set the FINIT bit (CAN\_FMR). Filter initialization also can be done outside the initialization mode.

**Note:** When FINIT=1, CAN reception is deactivated.

The filter values also can be modified by deactivating the associated filter activation bits (in the CAN\_FA1R register).

If a filter bank is not used, it is recommended to leave it non active (leave the corresponding FACT bit cleared).

For code example refer to the Appendix section [A.11.1: bxCAN initialization mode code example](#).

#### 29.4.2 Normal mode

Once the initialization is complete, the software must request the hardware to enter Normal mode to be able to synchronize on the CAN bus and start reception and transmission.

The request to enter Normal mode is issued by clearing the INRQ bit in the CAN\_MCR register. The bxCAN enters Normal mode and is ready to take part in bus activities when it has synchronized with the data transfer on the CAN bus. This is done by waiting for the occurrence of a sequence of 11 consecutive recessive bits (Bus Idle state). The switch to Normal mode is confirmed by the 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 is not active (corresponding FACTx bit cleared). The filter scale and mode configuration must be configured before entering Normal Mode.

#### 29.4.3 Sleep mode (low-power)

To reduce power consumption, bxCAN has a low-power mode called Sleep mode. This mode is entered on software request by setting the SLEEP bit in the CAN\_MCR register. In this mode, the bxCAN clock is stopped, however software can still access the bxCAN mailboxes.

If software requests entry to **initialization** mode by setting the INRQ bit while bxCAN is in **Sleep** mode, it must also clear the SLEEP bit.

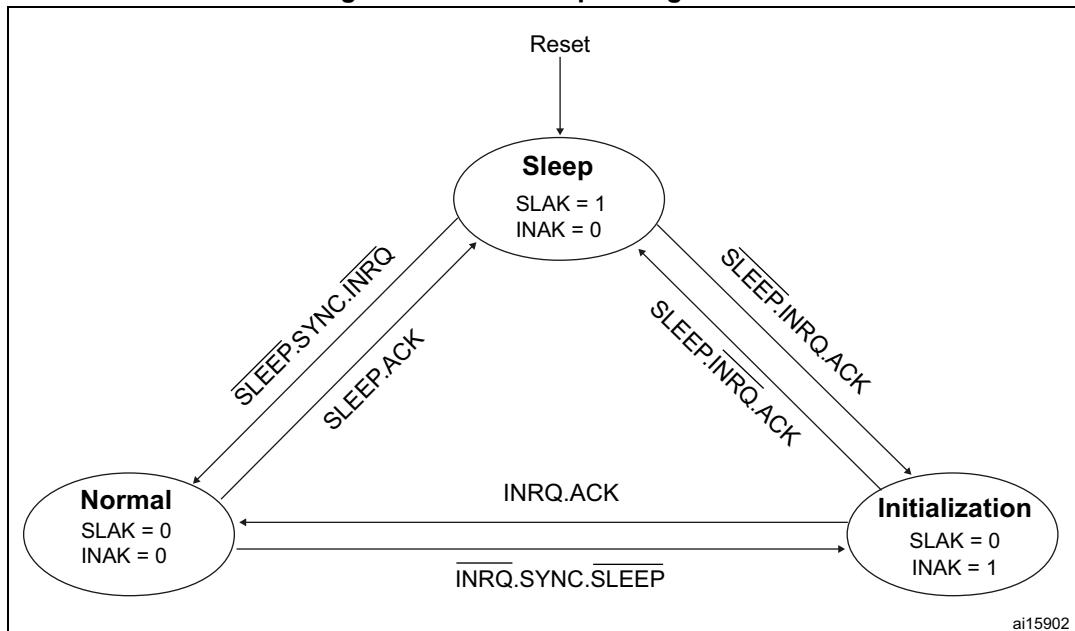
bxCAN can be woken up (exit Sleep mode) either by software clearing the SLEEP bit or on detection of CAN bus activity.

On CAN bus activity detection, hardware automatically performs the wake-up 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 wake-up interrupt occurs, in order to exit from Sleep mode.

**Note:** If the wake-up interrupt is enabled (WKUIE bit set in CAN\_IER register) a wake-up interrupt is generated on detection of CAN bus activity, even if the bxCAN automatically performs the wake-up sequence.

After the SLEEP bit has been cleared, Sleep mode is exited once bxCAN has synchronized with the CAN bus, refer to [Figure 313](#). The Sleep mode is exited once the SLAK bit has been cleared by hardware.

Figure 313. bxCAN operating modes



1. ACK = The wait state during which hardware confirms a request by setting the INAK or SLAK bits in the CAN\_MSR register.
2. SYNC = The state during which bxCAN waits until the CAN bus is idle, meaning 11 consecutive recessive bits have been monitored on CANRX.

## 29.5 Test mode

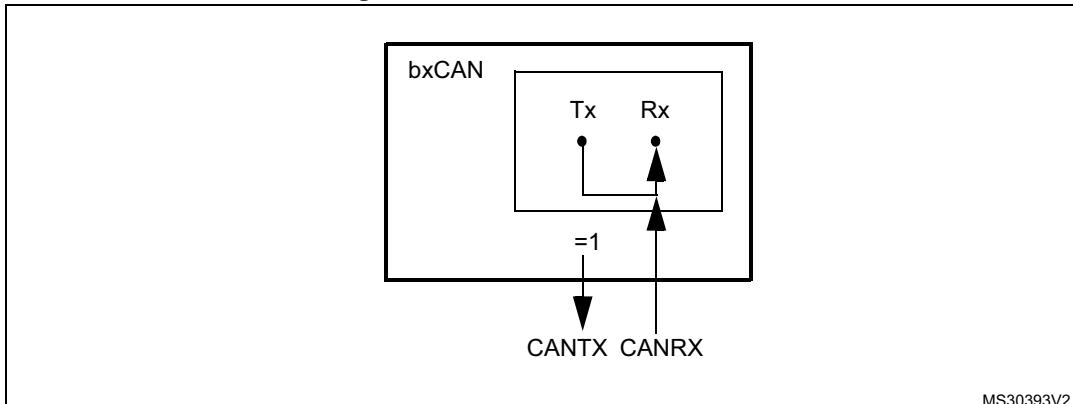
Test mode can be selected by the SILM and LBKM bits in the CAN\_BTR register. These bits must be configured while bxCAN is in Initialization mode. Once test mode has been selected, the INRQ bit in the CAN\_MCR register must be reset to enter Normal mode.

### 29.5.1 Silent mode

The bxCAN can be put in Silent mode by setting the SILM bit in the CAN\_BTR register.

In Silent mode, the bxCAN is able to receive valid data frames and valid remote frames, but sends only recessive bits on the CAN bus and cannot start a transmission. If the bxCAN has to send a dominant bit (ACK bit, overload flag, active error flag), the bit is rerouted internally so that the CAN Core monitors this dominant bit, although the CAN bus may remain in recessive state. Silent mode can be used to analyze the traffic on a CAN bus without affecting it by the transmission of dominant bits (Acknowledge bits, Error frames).

Figure 314. bxCAN in silent mode

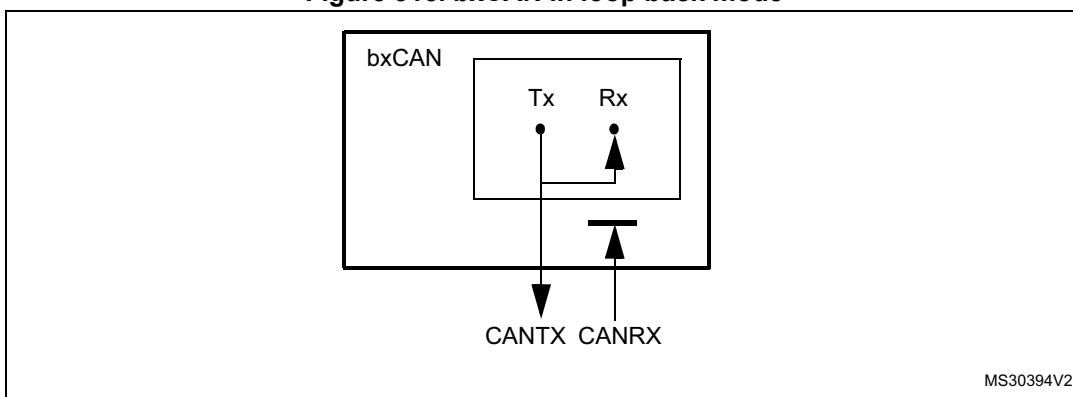


MS30393V2

### 29.5.2 Loop back mode

The bxCAN can be set in Loop Back Mode by setting the LBKM bit in the CAN\_BTR register. In Loop Back Mode, the bxCAN treats its own transmitted messages as received messages and stores them (if they pass acceptance filtering) in a Receive mailbox.

Figure 315. bxCAN in loop back mode



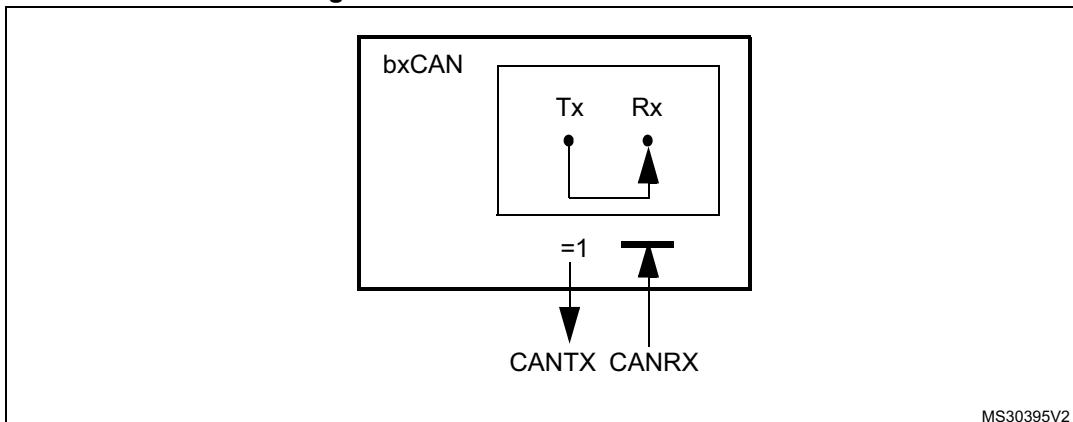
MS30394V2

This mode is provided for self-test functions. To be independent of external events, the CAN Core ignores acknowledge errors (no dominant bit sampled in the acknowledge slot of a data / remote frame) in Loop Back Mode. In this mode, the bxCAN performs an internal feedback from its Tx output to its Rx input. The actual value of the CANRX input pin is disregarded by the bxCAN. The transmitted messages can be monitored on the CANTX pin.

### 29.5.3 Loop back combined with silent mode

It is also possible to combine Loop back mode and Silent mode by setting the LBKM and SILM bits in the CAN\_BTR register. This mode can be used for a "Hot Selftest", meaning the bxCAN can be tested like in Loop back mode but without affecting a running CAN system connected to the CANTX and CANRX pins. In this mode, the CANRX pin is disconnected from the bxCAN and the CANTX pin is held recessive.

Figure 316. bxCAN in combined mode



## 29.6 Behavior in debug mode

When the microcontroller enters the debug mode (Cortex<sup>®</sup>-M0 core halted), the bxCAN continues to work normally or stops, depending on:

- the DBG\_CAN1\_STOP bit in the DBG module for the single mode.
- the DBF bit in CAN\_MCR. For more details, refer to [Section 29.9.2: CAN control and status registers](#).

## 29.7 bxCAN functional description

### 29.7.1 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\_TlxR register. Once the mailbox has left **empty** state, the software no longer has write access to the mailbox registers. Immediately after the TXRQ bit has been set, the mailbox enters **Pending** state and waits to become the highest priority mailbox, see *Transmit Priority*. As soon as the mailbox has the highest priority it is **Scheduled** for transmission. The transmission of the message of the scheduled mailbox starts (enter **transmit** state) when the CAN bus becomes idle. Once the mailbox has been successfully transmitted, it becomes **empty** again. The hardware indicates a successful transmission by setting the RQCP and TXOK bits in the CAN\_TSR register.

If the transmission fails, the cause is indicated by the ALST bit in the CAN\_TSR register in case of an Arbitration Lost, and/or the TERR bit, in case of transmission error detection.

For code example refer to the Appendix section [A.11.2: bxCAN transmit code example](#).

#### 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 is scheduled first.

By transmit request order

The transmit mailboxes can be configured as a transmit FIFO by setting the TXFP bit in the CAN\_MCR register. In this mode the priority order is given by the transmit request order.

This mode is very useful for segmented transmission.

### Abort

A transmission request can be aborted by the user setting the ABRQ bit in the CAN\_TSR register. In **pending** or **scheduled** state, the mailbox is aborted immediately. An abort request while the mailbox is in **transmit** state can have two results. If the mailbox is transmitted successfully the mailbox becomes **empty** with the TXOK bit set in the CAN\_TSR register. If the transmission fails, the mailbox becomes **scheduled**, the transmission is aborted and becomes **empty** with TXOK cleared. In all cases the mailbox becomes **empty** again at least at the end of the current transmission.

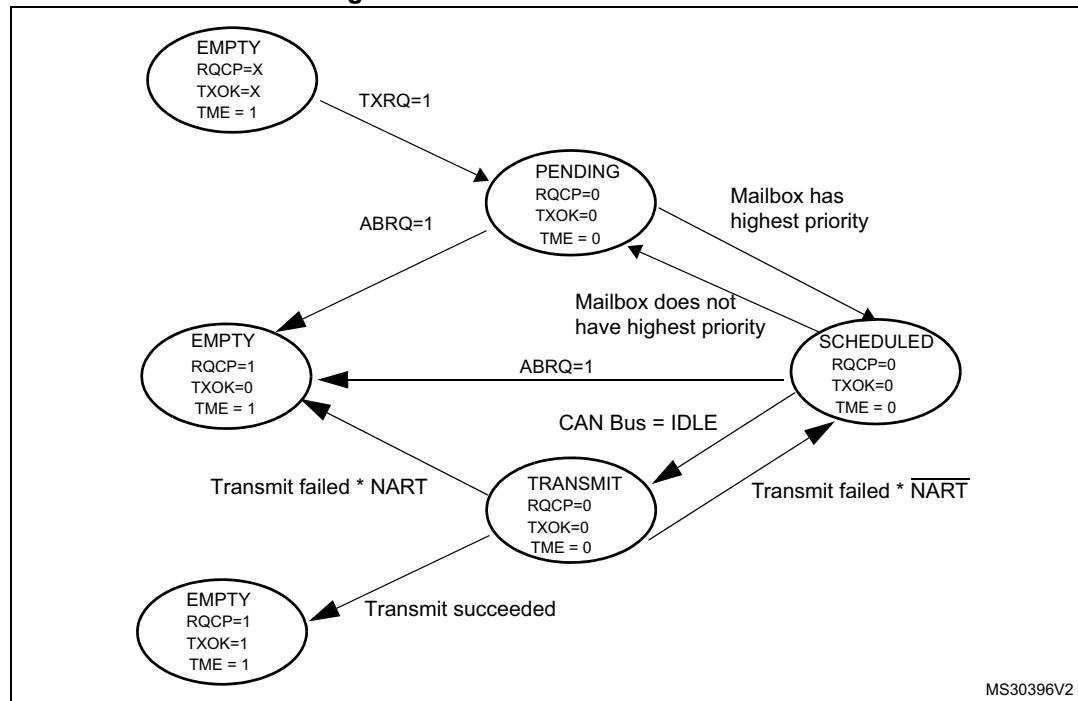
### Non automatic retransmission mode

This mode has been implemented in order to fulfill 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 does not automatically restart the message transmission.

At the end of the first transmission attempt, the hardware considers the request as completed and sets the RQCP bit in the CAN\_TSR register. The result of the transmission is indicated in the CAN\_TSR register by the TXOK, ALST and TERR bits.

**Figure 317. Transmit mailbox states**



### 29.7.2 Time triggered communication mode

In this mode, the internal counter of the CAN hardware is activated and used to generate the time stamp value stored in the CAN\_RDTxR/CAN\_TDTxR registers, respectively (for Rx and Tx mailboxes). The internal counter is incremented each CAN bit time (refer to [Section 29.7.7](#)). The internal counter is captured on the sample point of the Start Of Frame bit in both reception and transmission.

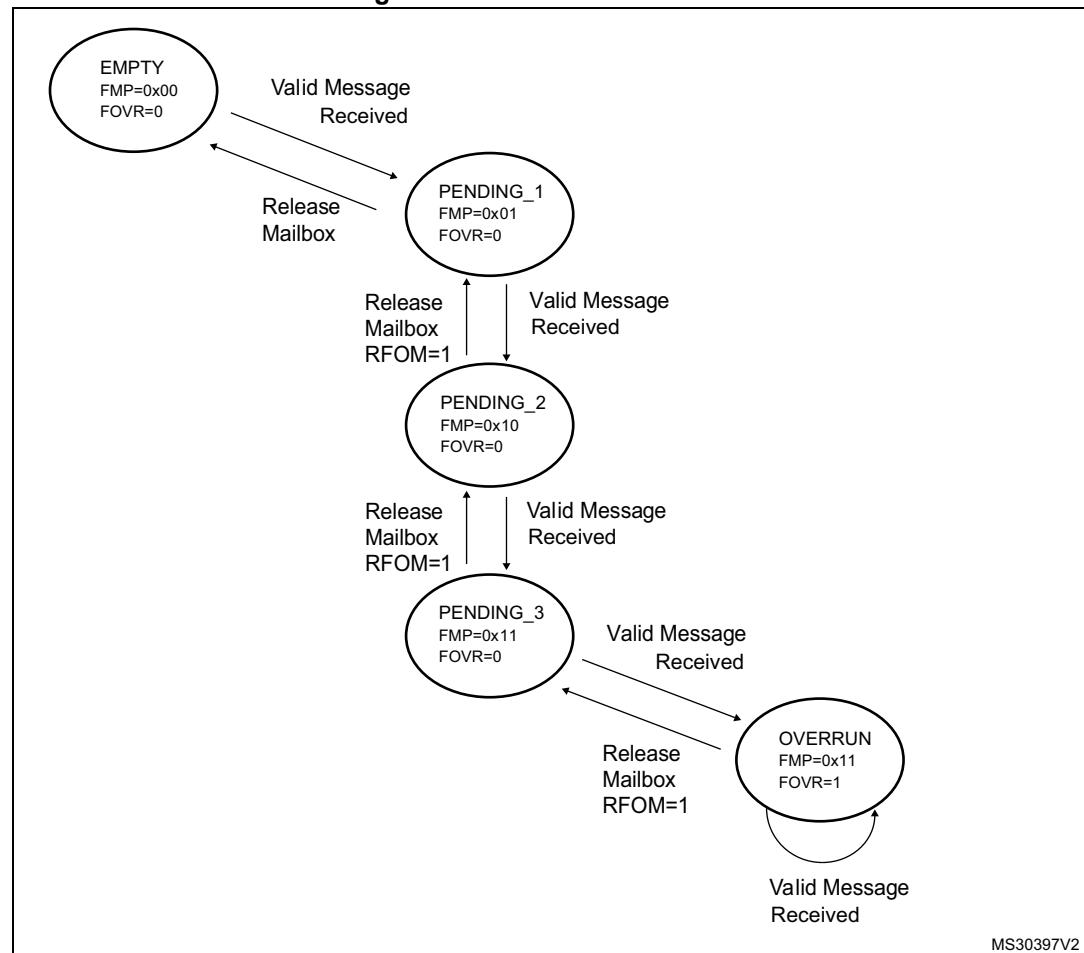
### 29.7.3 Reception handling

For the reception of CAN messages, three mailboxes organized as a FIFO are provided. In order to save CPU load, simplify the software and guarantee data consistency, the FIFO is managed completely by hardware. The application accesses the messages stored in the FIFO through the FIFO output mailbox.

#### Valid message

A received message is considered as valid **when** it has been received correctly according to the CAN protocol (no error until the last but one bit of the EOF field) **and** It passed through the identifier filtering successfully, see [Section 29.7.4](#).

**Figure 318. Receive FIFO states**



## FIFO management

Starting from the **empty** state, the first valid message received is stored in the FIFO which becomes **pending\_1**. The hardware signals the event setting the FMP[1:0] bits in the CAN\_RFR register to the value 01b. The message is available in the FIFO output mailbox. The software reads out the mailbox content and releases it by setting the RFOM bit in the CAN\_RFR register. The FIFO becomes **empty** again. If a new valid message has been received in the meantime, the FIFO stays in **pending\_1** state and the new message is available in the output mailbox.

For code example refer to the Appendix section [A.11.3: bxCAN receive code example](#).

If the application does not release the mailbox, the next valid message is stored in the FIFO which enters **pending\_2** state (FMP[1:0] = 10b). The storage process is repeated for the next valid message putting the FIFO into **pending\_3** state (FMP[1:0] = 11b). At this point, the software must release the output mailbox by setting the RFOM bit, so that a mailbox is free to store the next valid message. Otherwise the next valid message received causes a loss of message. Refer also to [Section 29.7.5](#).

## Overrun

Once the FIFO is in **pending\_3** state (i.e. the three mailboxes are full) the next valid message reception leads to an **overrun** and a message is 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 is overwritten by the new incoming message. In this case the latest messages are always available to the application.
- If the FIFO lock function is enabled (RFLM bit in the CAN\_MCR register set) the most recent message is discarded and the software has the three oldest messages in the FIFO available.

## Reception related interrupts

Once a message has been stored in the FIFO, the FMP[1:0] bits are updated and an interrupt request is generated if the FMPIE bit in the CAN\_IER register is set.

When the FIFO becomes full (i.e. a third message is stored) the FULL bit in the CAN\_RFR register is set and an interrupt is generated if the FFIE bit in the CAN\_IER register is set.

On overrun condition, the FOVR bit is set and an interrupt is generated if the FOVIE bit in the CAN\_IER register is set.

## 29.7.4 Identifier filtering

In the CAN protocol the identifier of a message is not associated with the address of a node but related to the content of the message. Consequently a transmitter broadcasts its message to all receivers. On message reception a receiver node decides - depending on the identifier value - whether the software needs the message or not. If the message is needed, it is copied into the SRAM. If not, the message must be discarded without intervention by the software.

To fulfill this requirement the bxCAN Controller provides 14 configurable and scalable filter banks (13-0) to the application, in order to receive only the messages the software needs.

This hardware filtering saves CPU resources which would be otherwise needed to perform filtering by software. Each filter bank x consists of two 32-bit registers, CAN\_FxR0 and CAN\_FxR1.

### Scalable width

To optimize and adapt the filters to the application needs, each filter bank can be scaled independently. Depending on the filter scale a filter bank provides:

- One 32-bit filter for the STDID[10:0], EXTID[17:0], IDE and RTR bits.
- Two 16-bit filters for the STDID[10:0], RTR, IDE and EXTID[17:15] bits.

Refer to [Figure 319](#).

Furthermore, the filters can be configured in mask mode or in identifier list mode.

### Mask mode

In **mask** mode the identifier registers are associated with mask registers specifying which bits of the identifier are handled as “must match” or as “don’t care”.

### Identifier list mode

In **identifier list** mode, the mask registers are used as identifier registers. Thus instead of defining an identifier and a mask, two identifiers are specified, doubling the number of single identifiers. All bits of the incoming identifier must match the bits specified in the filter registers.

### Filter bank scale and mode configuration

The filter banks are configured by means of the corresponding CAN\_FMR register. To configure a filter bank it must be deactivated by clearing the FACT bit in the CAN\_FAR register. The filter scale is configured by means of the corresponding FSCx bit in the CAN\_FS1R register, refer to [Figure 319](#). The **identifier list** or **identifier mask** mode for the corresponding Mask/Identifier registers is configured by means of the FBMx bits in the CAN\_FMR register.

To filter a group of identifiers, configure the Mask/Identifier registers in mask mode.

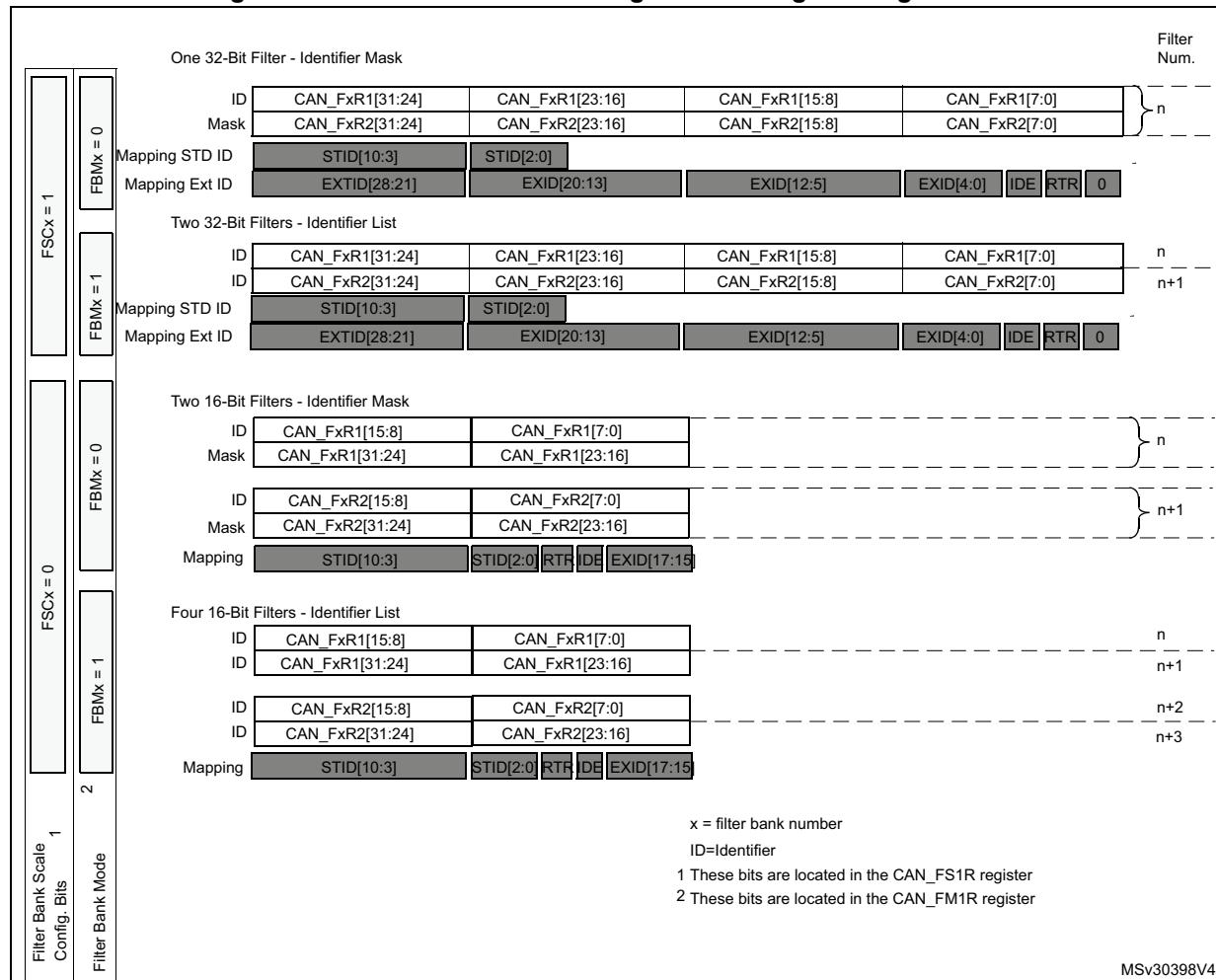
To select single identifiers, configure the Mask/Identifier registers in identifier list mode.

Filters not used by the application should be left deactivated.

Each filter within a filter bank is numbered (called the *Filter Number*) from 0 to a maximum dependent on the mode and the scale of each of the filter banks.

Concerning the filter configuration, refer to [Figure 319](#).

Figure 319. Filter bank scale configuration - Register organization



## Filter match index

Once a message has been received in the FIFO it is available to the application. Typically, application data is copied into SRAM locations. To copy the data to the right location the application has to identify the data by means of the identifier. To avoid this, and to ease the access to the SRAM locations, the CAN controller provides a filter match index.

This index is stored in the mailbox together with the message according to the filter priority rules. Thus each received message has its associated filter match index.

The filter match index can be used in two ways:

- Compare the filter match index with a list of expected values.
- Use the filter match index as an index on an array to access the data destination location.

For non masked filters, the software no longer has to compare the identifier.

If the filter is masked the software reduces the comparison to the masked bits only.

The index value of the filter number does not take into account the activation state of the filter banks. In addition, two independent numbering schemes are used, one for each FIFO. Refer to [Figure 320](#) for an example.

**Figure 320. Example of filter numbering**

Filter Bank	FIFO0	Filter Num.	Filter Bank	FIFO1	Filter Num.
0	ID List (32-bit)	0 1	2	ID Mask (16-bit)	0 1
1	ID Mask (32-bit)	2	4	ID List (32-bit)	2 3
3	ID List (16-bit)	3 4 5 6	7	Deactivated ID Mask (16-bit)	4 5
5	Deactivated ID List (32-bit)	7 8	8	ID Mask (16-bit)	6 7
6	ID Mask (16-bit)	9 10	10	Deactivated ID List (16-bit)	8 9 10 11
9	ID List (32-bit)	11 12	11	ID List (32-bit)	12 13
13	ID Mask (32-bit)	13	12	ID Mask (32-bit)	14

ID=Identifier

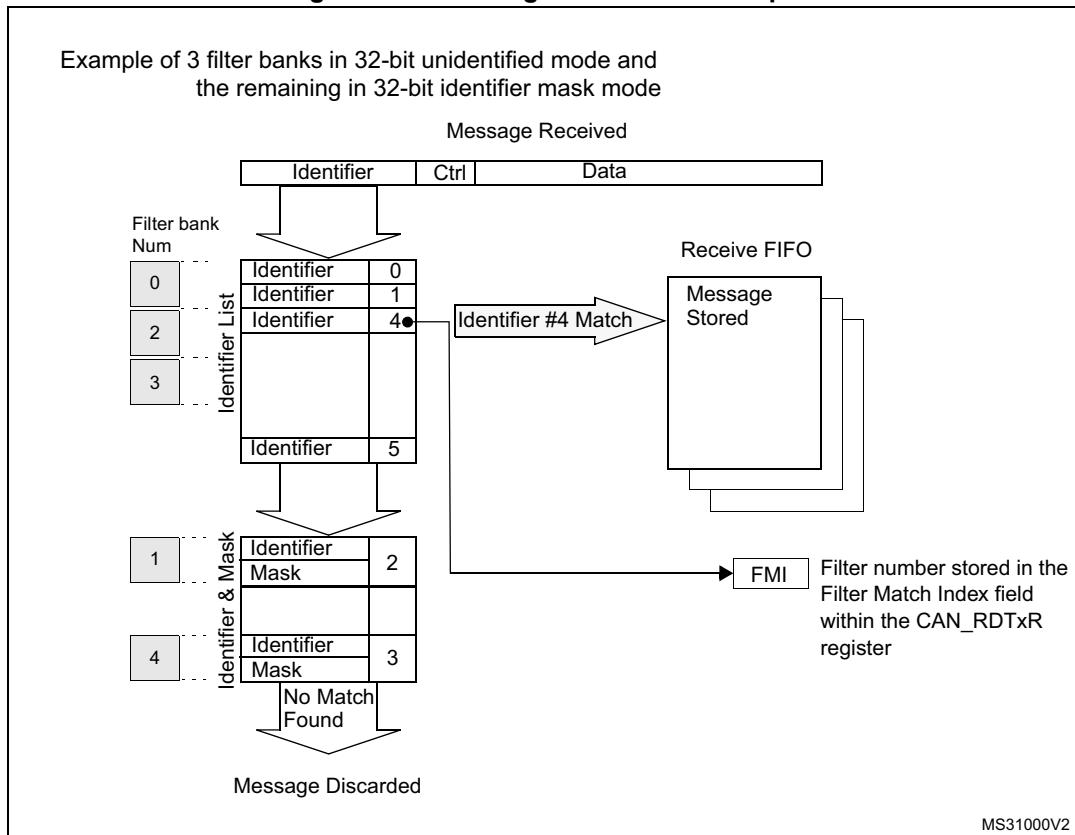
MS30399V2

### Filter priority rules

Depending on the filter combination it may occur that an identifier passes successfully through several filters. In this case the filter match value stored in the receive mailbox is chosen according to the following priority rules:

- A 32-bit filter takes priority over a 16-bit filter.
- For filters of equal scale, priority is given to the Identifier List mode over the Identifier Mask mode
- For filters of equal scale and mode, priority is given by the filter number (the lower the number, the higher the priority).

Figure 321. Filtering mechanism example



The example above shows the filtering principle of the bxCAN. On reception of a message, the identifier is compared first with the filters configured in identifier list mode. If there is a match, the message is stored in the associated FIFO and the index of the matching filter is stored in the filter match index. As shown in the example, the identifier matches with Identifier #2 thus the message content and FMI 2 is stored in the FIFO.

If there is no match, the incoming identifier is then compared with the filters configured in mask mode.

If the identifier does not match any of the identifiers configured in the filters, the message is discarded by hardware without disturbing the software.

### 29.7.5 Message storage

The interface between the software and the hardware for the CAN messages is implemented by means of mailboxes. A mailbox contains all information related to a message; identifier, data, control, status and time stamp information.

#### Transmit mailbox

The software sets up the message to be transmitted in an empty transmit mailbox. The status of the transmission is indicated by hardware in the CAN\_TSR register.

**Table 118. Transmit mailbox mapping**

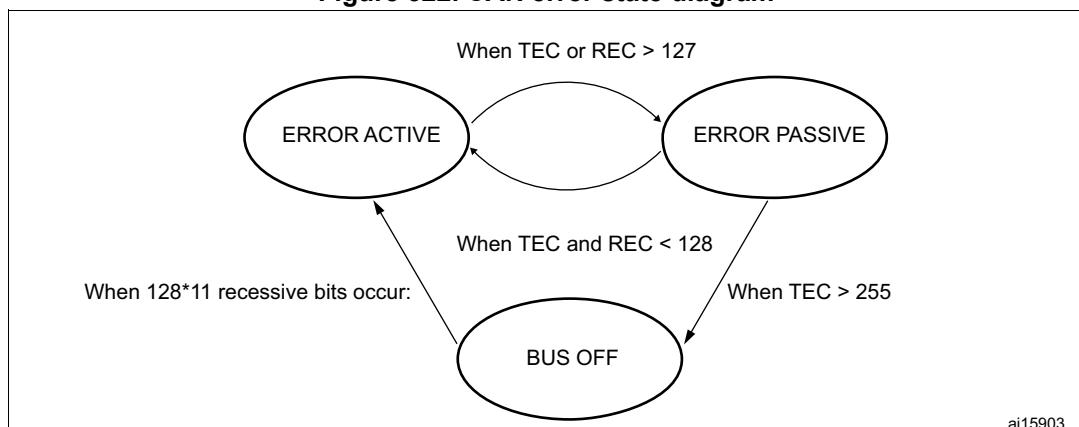
Offset to transmit mailbox base address	Register name
0	CAN_TIxR
4	CAN_TDTxR
8	CAN_TDLxR
12	CAN_TDhxR

### Receive mailbox

When a message has been received, it is available to the software in the FIFO output mailbox. Once the software has handled the message (e.g. read it) the software must release the FIFO output mailbox by means of the RFOM bit in the CAN\_RFR register to make the next incoming message available. The filter match index is stored in the MFMI field of the CAN\_RDTxR register. The 16-bit time stamp value is stored in the TIME[15:0] field of CAN\_RDTxR.

**Table 119. Receive mailbox mapping**

Offset to receive mailbox base address (bytes)	Register name
0	CAN_RIxR
4	CAN_RDTxR
8	CAN_RDLxR
12	CAN_RDhxR

**Figure 322. CAN error state diagram**

### 29.7.6 Error management

The error management as described in the CAN protocol is handled entirely by hardware using a Transmit Error Counter (TEC value, in CAN\_ESR register) and a Receive Error Counter (REC value, in the CAN\_ESR register), which get incremented or decremented according to the error condition. For detailed information about TEC and REC management, refer to the CAN standard.

Both of them may be read by software to determine the stability of the network. Furthermore, the CAN hardware provides detailed information on the current error status in CAN\_ESR register. By means of the CAN\_IER register (ERRIE bit, etc.), the software can configure the interrupt generation on error detection in a very flexible way.

### Bus-Off recovery

The Bus-Off state is reached when TEC is greater than 255, this state is indicated by BOFF bit in CAN\_ESR register. In Bus-Off state, the bxCAN is no longer able to transmit and receive messages.

Depending on the ABOM bit in the CAN\_MCR register, bxCAN recovers from Bus-Off (become error active again) either automatically or on software request. But in both cases the bxCAN has to wait at least for the recovery sequence specified in the CAN standard (128 occurrences of 11 consecutive recessive bits monitored on CANRX).

If ABOM is set, the bxCAN starts the recovering sequence automatically after it has entered Bus-Off state.

If ABOM is cleared, the software must initiate the recovering sequence by requesting bxCAN to enter and to leave initialization mode.

*Note:* *In initialization mode, bxCAN does not monitor the CANRX signal, therefore it cannot complete the recovery sequence. To recover, bxCAN must be in normal mode.*

## 29.7.7 Bit timing

The bit timing logic monitors the serial bus-line and performs sampling and adjustment of the sample point by synchronizing on the start-bit edge and resynchronizing on the following edges.

Its operation may be explained simply by splitting nominal bit time into three segments as follows:

- **Synchronization segment (SYNC\_SEG):** a bit change is expected to occur within this time segment. It has a fixed length of one time quantum ( $1 \times t_q$ ).
- **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.

A valid edge is defined as the first transition in a bit time from dominant to recessive bus level provided the controller itself does not send a recessive bit.

If a valid edge is detected in BS1 instead of SYNC\_SEG, BS1 is extended by up to SJW so that the sample point is delayed.

Conversely, if a valid edge is detected in BS2 instead of SYNC\_SEG, BS2 is shortened by up to SJW so that the transmit point is moved earlier.

As a safeguard against programming errors, the configuration of the Bit timing register (CAN\_BTR) is only possible while the device is in Standby mode.

**Note:** For a detailed description of the CAN bit timing and resynchronization mechanism, refer to the ISO 11898 standard.

**Figure 323. Bit timing**

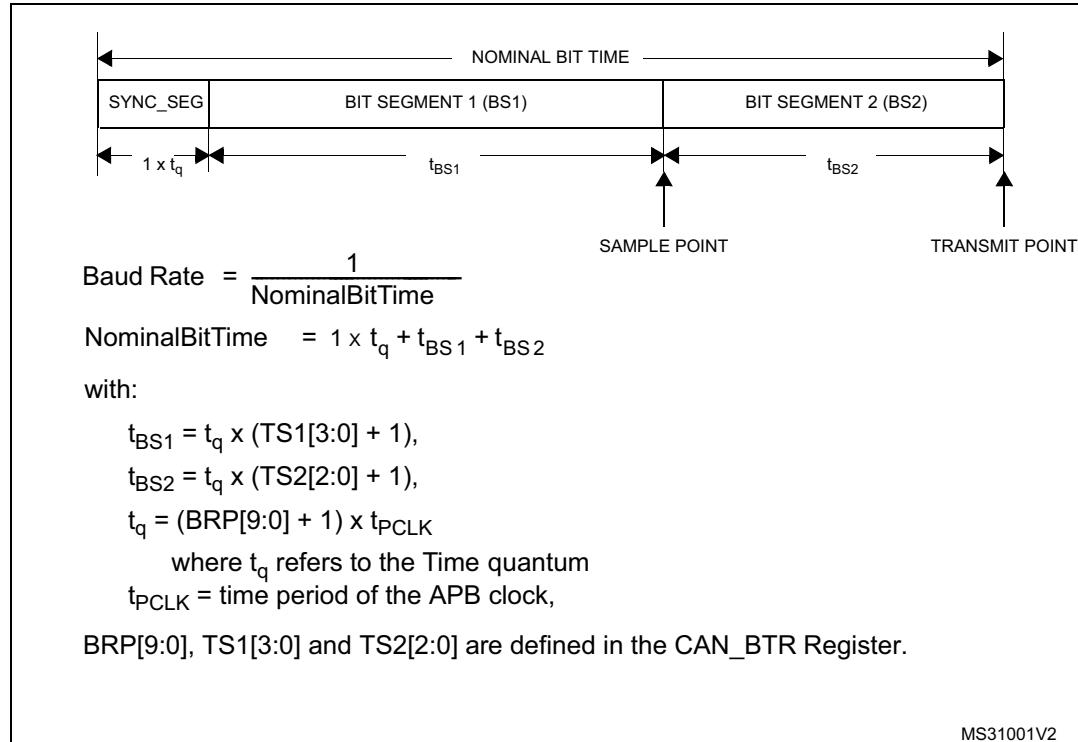
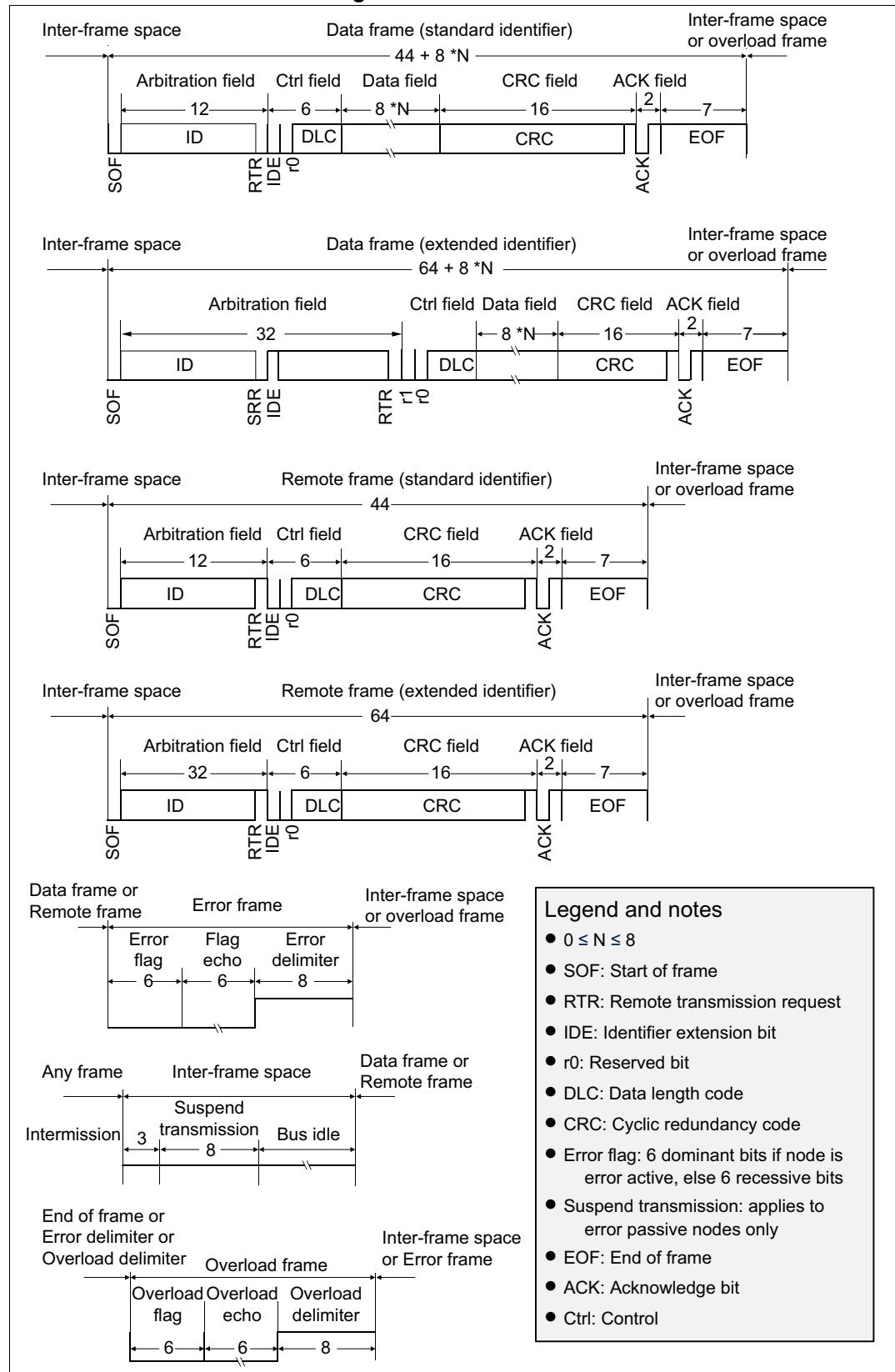


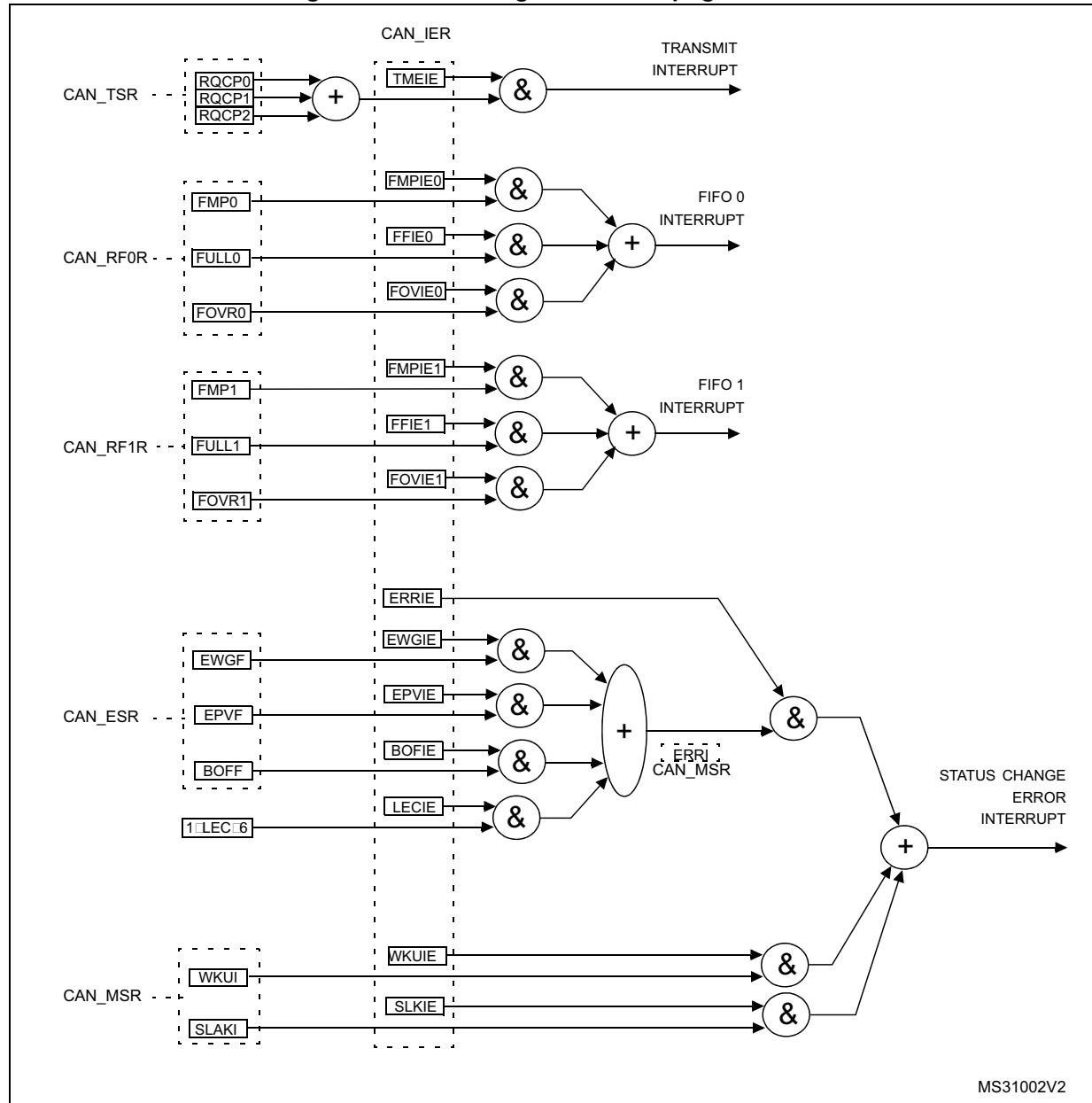
Figure 324. CAN frames



## 29.8 bxCAN interrupts

Four interrupt vectors are dedicated to bxCAN. Each interrupt source can be independently enabled or disabled by means of the CAN interrupt enable register (CAN\_IER).

**Figure 325. Event flags and interrupt generation**



- The **transmit interrupt** can be generated by the following events:
  - Transmit mailbox 0 becomes empty, RQCP0 bit in the CAN\_TSR register set.
  - Transmit mailbox 1 becomes empty, RQCP1 bit in the CAN\_TSR register set.
  - Transmit mailbox 2 becomes empty, RQCP2 bit in the CAN\_TSR register set.
- The **FIFO 0 interrupt** can be generated by the following events:
  - Reception of a new message, FMP0 bits in the CAN\_RF0R register are not '00'.
  - FIFO0 full condition, FULL0 bit in the CAN\_RF0R register set.
  - FIFO0 overrun condition, FOVR0 bit in the CAN\_RF0R register set.
- The **FIFO 1 interrupt** can be generated by the following events:
  - Reception of a new message, FMP1 bits in the CAN\_RF1R register are not '00'.
  - FIFO1 full condition, FULL1 bit in the CAN\_RF1R register set.
  - FIFO1 overrun condition, FOVR1 bit in the CAN\_RF1R register set.
- The **error and status change interrupt** can be generated by the following events:
  - Error condition, for more details on error conditions refer to the CAN Error Status register (CAN\_ESR).
  - Wake-up condition, SOF monitored on the CAN Rx signal.
  - Entry into Sleep mode.

## 29.9 CAN registers

The peripheral registers have to be accessed by words (32 bits).

### 29.9.1 Register access protection

Erroneous access to certain configuration registers can cause the hardware to temporarily disturb the whole CAN network. Therefore the CAN\_BTR register can be modified by software only while the CAN hardware is in initialization mode.

Although the transmission of incorrect data does 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 317: Transmit mailbox states](#).

The filter values can be modified either deactivating the associated filter banks or by setting the FINIT bit. Moreover, the modification of the filter configuration (scale, mode and FIFO assignment) in CAN\_FMxR, CAN\_FSxR and CAN\_FFAR registers can only be done when the filter initialization mode is set (FINIT=1) in the CAN\_FMR register.

### 29.9.2 CAN control and status registers

Refer to [Section 1.2](#) for a list of abbreviations used in register descriptions.

#### CAN master control register (CAN\_MCR)

Address offset: 0x00

Reset value: 0x0001 0002

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DBF
															rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESET	Res.	TTCM	ABOM	AWUM	NART	RFLM	TXFP	SLEEP	INRQ						
rs								rw	rw						

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

Bit 16 **DBF:** Debug freeze

- 0: CAN working during debug
- 1: CAN reception/transmission frozen during debug. Reception FIFOs can still be accessed/controlled normally.

Bit 15 **RESET:** bxCAN software master reset

- 0: Normal operation.
- 1: Force a master reset of the bxCAN -> Sleep mode activated after reset (FMP bits and CAN\_MCR register are initialized to the reset values). This bit is automatically reset to 0.

Bits 14:8 Reserved, must be kept at reset value.

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, refer to Section 29.7.2: Time triggered communication mode.*

Bit 6 **ABOM:** Automatic bus-off management

- This bit controls the behavior of the CAN hardware on leaving the Bus-Off state.
- 0: The Bus-Off state is left on software request, once 128 occurrences of 11 recessive bits have been monitored and the software has first set and cleared the INRQ bit of the CAN\_MCR register.
- 1: The Bus-Off state is left automatically by hardware once 128 occurrences of 11 recessive bits have been monitored.

For detailed information on the Bus-Off state refer to [Section 29.7.6: Error management](#).

Bit 5 **AWUM:** Automatic wake-up mode

- This bit controls the behavior of the CAN hardware on message reception during Sleep mode.
  - 0: The Sleep mode is left on software request by clearing the SLEEP bit of the CAN\_MCR register.
  - 1: The Sleep mode is left automatically by hardware on CAN message detection.
- The SLEEP bit of the CAN\_MCR register and the SLAK bit of the CAN\_MSR register are cleared by hardware.

Bit 4 **NART:** No automatic retransmission

- 0: The CAN hardware automatically retransmits the message until it has been successfully transmitted according to the CAN standard.
- 1: A message is 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 overwrites the previous one.

1: Receive FIFO locked against overrun. Once a receive FIFO is full the next incoming message is discarded.

Bit 2 **TXFP**: Transmit FIFO priority

This bit controls the transmission order when several mailboxes are pending at the same time.

0: Priority driven by the identifier of the message

1: Priority driven by the request order (chronologically)

Bit 1 **SLEEP**: Sleep mode request

This bit is set by software to request the CAN hardware to enter the Sleep mode. Sleep mode is entered as soon as the current CAN activity (transmission or reception of a CAN frame) has been completed.

This bit is cleared by software to exit Sleep mode.

This bit is cleared by hardware when the AWUM bit is set and a SOF bit is detected on the CAN Rx signal.

This bit is set after reset - CAN starts in Sleep mode.

Bit 0 **INRQ**: Initialization request

The software clears this bit to switch the hardware into normal mode. Once 11 consecutive recessive bits have been monitored on the Rx signal the CAN hardware is synchronized and ready for transmission and reception. Hardware signals this event by clearing the INAK bit in the CAN\_MSR register.

Software sets this bit to request the CAN hardware to enter initialization mode. Once software has set the INRQ bit, the CAN hardware waits until the current CAN activity (transmission or reception) is completed before entering the initialization mode. Hardware signals this event by setting the INAK bit in the CAN\_MSR register.

**CAN master status register (CAN\_MSR)**

Address offset: 0x04

Reset value: 0x0000 0C02

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	RX	SAMP	RXM	TXM	Res.	Res.	Res.	SLAKI	WKUI	ERRI	SLAK	INAK
				r	r	r	r				rc_w1	rc_w1	rc_w1	r	r

Bits 31:12 Reserved, must be kept at reset value.

Bit 11 **RX**: CAN Rx signal

Monitors the actual value of the **CAN\_RX** Pin.

Bit 10 **SAMP**: Last sample point

The value of RX on the last sample point (current received bit value).

Bit 9 **RXM**: Receive mode

The CAN hardware is currently receiver.

Bit 8 **TXM**: Transmit mode

The CAN hardware is currently transmitter.

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

Bit 4 **SLAKI**: Sleep acknowledge interrupt

When SLKIE=1, this bit is set by hardware to signal that the bxCAN has entered Sleep Mode. When set, this bit generates a status change interrupt if the SLKIE bit in the CAN\_IER register is set.

This bit is cleared by software or by hardware, when SLAK is cleared.

*Note: When SLKIE=0, no polling on SLAKI is possible. In this case the SLAK bit can be polled.*

Bit 3 **WKUI**: Wake-up interrupt

This bit is set by hardware to signal that a SOF bit has been detected while the CAN hardware was in Sleep mode. Setting this bit generates a status change interrupt if the WKUIE bit in the CAN\_IER register is set.

This bit is cleared by software.

Bit 2 **ERRI**: Error interrupt

This bit is set by hardware when a bit of the CAN\_ESR has been set on error detection and the corresponding interrupt in the CAN\_IER is enabled. Setting this bit generates a status change interrupt if the ERRIE bit in the CAN\_IER register is set.

This bit is cleared by software.

Bit 1 **SLAK**: Sleep acknowledge

This bit is set by hardware and indicates to the software that the CAN hardware is now in Sleep mode. This bit acknowledges the Sleep mode request from the software (set SLEEP bit in CAN\_MCR register).

This bit is cleared by hardware when the CAN hardware has left Sleep mode (to be synchronized on the CAN bus). To be synchronized the hardware has to monitor a sequence of 11 consecutive recessive bits on the CAN RX signal.

*Note: The process of leaving Sleep mode is triggered when the SLEEP bit in the CAN\_MCR register is cleared. Refer to the AWUM bit of the CAN\_MCR register description for detailed information for clearing SLEEP bit*

Bit 0 **INAK**: Initialization acknowledge

This bit is set by hardware and indicates to the software that the CAN hardware is now in initialization mode. This bit acknowledges the initialization request from the software (set INRQ bit in CAN\_MCR register).

This bit is cleared by hardware when the CAN hardware has left the initialization mode (to be synchronized on the CAN bus). To be synchronized the hardware has to monitor a sequence of 11 consecutive recessive bits on the CAN RX signal.

**CAN transmit status register (CAN\_TSR)**

Address offset: 0x08

Reset value: 0x1C00 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LOW2	LOW1	LOW0	TME2	TME1	TME0	CODE[1:0]		ABRQ2	Res.	Res.	Res.	TERR2	ALST2	TXOK2	RQCP2
r	r	r	r	r	r	r	r	rs				rc_w1	rc_w1	rc_w1	rc_w1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABRQ1	Res.	Res.	Res.	TERR1	ALST1	TXOK1	RQCP1	ABRQ0	Res.	Res.	Res.	TERR0	ALST0	TXOK0	RQCP0
rs				rc_w1	rc_w1	rc_w1	rc_w1	rs				rc_w1	rc_w1	rc_w1	rc_w1

Bit 31 **LOW2**: Lowest priority flag for mailbox 2

This bit is set by hardware when more than one mailbox are pending for transmission and mailbox 2 has the lowest priority.

Bit 30 **LOW1**: Lowest priority flag for mailbox 1

This bit is set by hardware when more than one mailbox are pending for transmission and mailbox 1 has the lowest priority.

Bit 29 **LOW0**: Lowest priority flag for mailbox 0

This bit is set by hardware when more than one mailbox are pending for transmission and mailbox 0 has the lowest priority.

*Note: The LOW[2:0] bits are set to 0 when only one mailbox is pending.*

Bit 28 **TME2**: Transmit mailbox 2 empty

This bit is set by hardware when no transmit request is pending for mailbox 2.

Bit 27 **TME1**: Transmit mailbox 1 empty

This bit is set by hardware when no transmit request is pending for mailbox 1.

Bit 26 **TME0**: Transmit mailbox 0 empty

This bit is set by hardware when no transmit request is pending for mailbox 0.

Bits 25:24 **CODE[1:0]**: Mailbox code

In case at least one transmit mailbox is free, the code value is equal to the number of the next transmit mailbox free.

In case all transmit mailboxes are pending, the code value is equal to the number of the transmit mailbox with the lowest priority.

Bit 23 **ABRQ2**: Abort request for mailbox 2

Set by software to abort the transmission request for the corresponding mailbox.

Cleared by hardware when the mailbox becomes empty.

Setting this bit has no effect when the mailbox is not pending for transmission.

Bits 22:20 Reserved, must be kept at reset value.

Bit 19 **TERR2**: Transmission error of mailbox 2

This bit is set when the previous TX failed due to an error.

Bit 18 **ALST2**: Arbitration lost for mailbox 2

This bit is set when the previous TX failed due to an arbitration lost.

Bit 17 **TXOK2**: Transmission OK of mailbox 2

The hardware updates this bit after each transmission attempt.

0: The previous transmission failed

1: The previous transmission was successful

This bit is set by hardware when the transmission request on mailbox 2 has been completed successfully. Refer to [Figure 317](#).

- Bit 16 **RQCP2**: Request completed mailbox2  
Set by hardware when the last request (transmit or abort) has been performed.  
Cleared by software writing a “1” or by hardware on transmission request (TXRQ2 set in CAN\_TMRD2R register).  
Clearing this bit clears all the status bits (TXOK2, ALST2 and TERR2) for Mailbox 2.
- Bit 15 **ABRQ1**: Abort request for mailbox 1  
Set by software to abort the transmission request for the corresponding mailbox.  
Cleared by hardware when the mailbox becomes empty.  
Setting this bit has no effect when the mailbox is not pending for transmission.
- Bits 14:12 Reserved, must be kept at reset value.
- Bit 11 **TERR1**: Transmission error of mailbox1  
This bit is set when the previous TX failed due to an error.
- Bit 10 **ALST1**: Arbitration lost for mailbox1  
This bit is set when the previous TX failed due to an arbitration lost.
- Bit 9 **TXOK1**: Transmission OK of mailbox1  
The hardware updates this bit after each transmission attempt.  
0: The previous transmission failed  
1: The previous transmission was successful  
This bit is set by hardware when the transmission request on mailbox 1 has been completed successfully. Refer to [Figure 317](#).
- Bit 8 **RQCP1**: Request completed mailbox1  
Set by hardware when the last request (transmit or abort) has been performed.  
Cleared by software writing a “1” or by hardware on transmission request (TXRQ1 set in CAN\_TMRD1R register).  
Clearing this bit clears all the status bits (TXOK1, ALST1 and TERR1) for Mailbox 1.
- Bit 7 **ABRQ0**: Abort request for mailbox0  
Set by software to abort the transmission request for the corresponding mailbox.  
Cleared by hardware when the mailbox becomes empty.  
Setting this bit has no effect when the mailbox is not pending for transmission.
- Bits 6:4 Reserved, must be kept at reset value.
- Bit 3 **TERR0**: Transmission error of mailbox0  
This bit is set when the previous TX failed due to an error.
- Bit 2 **ALST0**: Arbitration lost for mailbox0  
This bit is set when the previous TX failed due to an arbitration lost.
- Bit 1 **TXOK0**: Transmission OK of mailbox0  
The hardware updates this bit after each transmission attempt.  
0: The previous transmission failed  
1: The previous transmission was successful  
This bit is set by hardware when the transmission request on mailbox 0 has been completed successfully. Refer to [Figure 317](#).
- Bit 0 **RQCP0**: Request completed mailbox0  
Set by hardware when the last request (transmit or abort) has been performed.  
Cleared by software writing a “1” or by hardware on transmission request (TXRQ0 set in CAN\_TMRD0R register).  
Clearing this bit clears all the status bits (TXOK0, ALST0 and TERR0) for Mailbox 0.

**CAN receive FIFO 0 register (CAN\_RF0R)**

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	RFOM0	FOVR0	FULL0	Res.		FMP0[1:0]									
										rs	rc_w1	rc_w1		r	r

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

Bit 5 **RFOM0**: Release FIFO 0 output mailbox

Set by software to release the output mailbox of the FIFO. The output mailbox can only be released when at least one message is pending in the FIFO. Setting this bit when the FIFO is empty has no effect. If at least two messages are pending in the FIFO, the software has to release the output mailbox to access the next message.

Cleared by hardware when the output mailbox has been released.

Bit 4 **FOVR0**: FIFO 0 overrun

This bit is set by hardware when a new message has been received and passed the filter while the FIFO was full.

This bit is cleared by software.

Bit 3 **FULL0**: FIFO 0 full

Set by hardware when three messages are stored in the FIFO.

This bit is cleared by software.

## Bit 2 Reserved, must be kept at reset value.

Bits 1:0 **FMP0[1:0]**: FIFO 0 message pending

These bits indicate how many messages are pending in the receive FIFO.

FMP is increased each time the hardware stores a new message in to the FIFO. FMP is decreased each time the software releases the output mailbox by setting the RFOM0 bit.

**CAN receive FIFO 1 register (CAN\_RF1R)**

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	RFOM1	FOVR1	FULL1	Res.		FMP1[1:0]									
										rs	rc_w1	rc_w1		r	r

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

**Bit 5 RFOM1:** Release FIFO 1 output mailbox

Set by software to release the output mailbox of the FIFO. The output mailbox can only be released when at least one message is pending in the FIFO. Setting this bit when the FIFO is empty has no effect. If at least two messages are pending in the FIFO, the software has to release the output mailbox to access the next message.

Cleared by hardware when the output mailbox has been released.

**Bit 4 FOVR1:** FIFO 1 overrun

This bit is set by hardware when a new message has been received and passed the filter while the FIFO was full.

This bit is cleared by software.

**Bit 3 FULL1:** FIFO 1 full

Set by hardware when three messages are stored in the FIFO.

This bit is cleared by software.

Bit 2 Reserved, must be kept at reset value.

**Bits 1:0 FMP1[1:0]:** FIFO 1 message pending

These bits indicate how many messages are pending in the receive FIFO1.

FMP1 is increased each time the hardware stores a new message in to the FIFO1. FMP is decreased each time the software releases the output mailbox by setting the RFOM1 bit.

### CAN interrupt enable register (CAN\_IER)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SLKIE	WKUIE
														rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERRIE	Res.	Res.	Res.	LEC IE	BOF IE	EPV IE	EWG IE	Res.	FOV IE1	FF IE1	FMP IE1	FOV IE0	FF IE0	FMP IE0	TME IE
rw				rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

Bits 31:18 Reserved, must be kept at reset value.

**Bit 17 SLKIE:** Sleep interrupt enable

0: No interrupt when SLAKI bit is set.

1: Interrupt generated when SLAKI bit is set.

**Bit 16 WKUIE:** Wake-up interrupt enable

0: No interrupt when WKUI is set.

1: Interrupt generated when WKUI bit is set.

**Bit 15 ERRIE:** Error interrupt enable

0: No interrupt is generated when an error condition is pending in the CAN\_ESR.

1: An interrupt is generation when an error condition is pending in the CAN\_ESR.

Bits 14:12 Reserved, must be kept at reset value.

- Bit 11 **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 10 **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 9 **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 8 **EWGIE**: Error warning interrupt enable  
0: ERRI bit is not set when EWGF is set.  
1: ERRI bit is set when EWGF is set.
- Bit 7 Reserved, must be kept at reset value.
- Bit 6 **FOVIE1**: FIFO overrun interrupt enable  
0: No interrupt when FOVR is set.  
1: Interrupt generation when FOVR is set.
- Bit 5 **FFIE1**: FIFO full interrupt enable  
0: No interrupt when FULL bit is set.  
1: Interrupt generated when FULL bit is set.
- Bit 4 **FMPIE1**: FIFO message pending interrupt enable  
0: No interrupt generated when state of FMP[1:0] bits are not 00b.  
1: Interrupt generated when state of FMP[1:0] bits are not 00b.
- Bit 3 **FOVIE0**: FIFO overrun interrupt enable  
0: No interrupt when FOVR bit is set.  
1: Interrupt generated when FOVR bit is set.
- Bit 2 **FFIE0**: FIFO full interrupt enable  
0: No interrupt when FULL bit is set.  
1: Interrupt generated when FULL bit is set.
- Bit 1 **FMPIE0**: FIFO message pending interrupt enable  
0: No interrupt generated when state of FMP[1:0] bits are not 00b.  
1: Interrupt generated when state of FMP[1:0] bits are not 00b.
- Bit 0 **TMEIE**: Transmit mailbox empty interrupt enable  
0: No interrupt when RQCPx bit is set.  
1: Interrupt generated when RQCPx bit is set.

*Note:* Refer to [Section 29.8: bxCAN interrupts](#).

**CAN error status register (CAN\_ESR)**

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
REC[7:0]								TEC[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	LEC[2:0]				Res.	BOFF	EPVF	EWGF
								rw	rw	rw		r	r	r	

Bits 31:24 **REC[7:0]**: Receive error counter

The implementing part of the fault confinement mechanism of the CAN protocol. In case of an error during reception, this counter is incremented by 1 or by 8 depending on the error condition as defined by the CAN standard. After every successful reception the counter is decremented by 1 or reset to 120 if its value was higher than 128. When the counter value exceeds 127, the CAN controller enters the error passive state.

Bits 23:16 **TEC[7:0]**: Least significant byte of the 9-bit transmit error counter

The implementing part of the fault confinement mechanism of the CAN protocol.

Bits 15:7 Reserved, must be kept at reset value.

Bits 6:4 **LEC[2:0]**: Last error code

This field is set by hardware and holds a code which indicates the error condition of the last error detected on the CAN bus. If a message has been transferred (reception or transmission) without error, this field is cleared to 0.

The LEC[2:0] bits can be set to value 0b111 by software. They are updated by hardware to indicate the current communication status.

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, must be kept at reset value.

Bit 2 **BOFF**: Bus-off flag

This bit is set by hardware when it enters the bus-off state. The bus-off state is entered on TEC overflow, greater than 255, refer to [Section 29.7.6: Error management](#).

Bit 1 **EPVF**: Error passive flag

This bit is set by hardware when the Error passive limit has been reached (Receive Error Counter or Transmit Error Counter > 127).

Bit 0 **EWGF**: Error warning flag

This bit is set by hardware when the warning limit has been reached (Receive Error Counter or Transmit Error Counter ≥ 96).

### CAN bit timing register (CAN\_BTR)

Address offset: 0x1C

Reset value: 0x0123 0000

This register can only be accessed by the software when the CAN hardware is in initialization mode.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SILM	LBKM	Res.	Res.	Res.	Res.	SJW[1:0]		Res.	TS2[2:0]			TS1[3:0]			
rw	rw					rw	rw		rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	BRP[9:0]									
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **SILM**: Silent mode (debug)

0: Normal operation

1: Silent Mode

Bit 30 **LBKM**: Loop back mode (debug)

0: Loop Back Mode disabled

1: Loop Back Mode enabled

Bits 29:26 Reserved, must be kept at reset value.

Bits 25:24 **SJW[1:0]**: Resynchronization jump width

These bits define the maximum number of time quanta the CAN hardware is allowed to lengthen or shorten a bit to perform the resynchronization.

$$t_{RJW} = t_q \times (SJW[1:0] + 1)$$

Bit 23 Reserved, must be kept at reset value.

Bits 22:20 **TS2[2:0]**: Time segment 2

These bits define the number of time quanta in Time Segment 2.

$$t_{BS2} = t_q \times (TS2[2:0] + 1)$$

Bits 19:16 **TS1[3:0]**: Time segment 1

These bits define the number of time quanta in Time Segment 1

$$t_{BS1} = t_q \times (TS1[3:0] + 1)$$

For more information on bit timing, refer to [Section 29.7.7: Bit timing](#).

Bits 15:10 Reserved, must be kept at reset value.

Bits 9:0 **BRP[9:0]**: Baud rate prescaler

These bits define the length of a time quanta.

$$t_q = (BRP[9:0]+1) \times t_{PCLK}$$

### 29.9.3 CAN mailbox registers

This chapter describes the registers of the transmit and receive mailboxes. Refer to [Section 29.7.5: Message storage](#) for detailed register mapping.

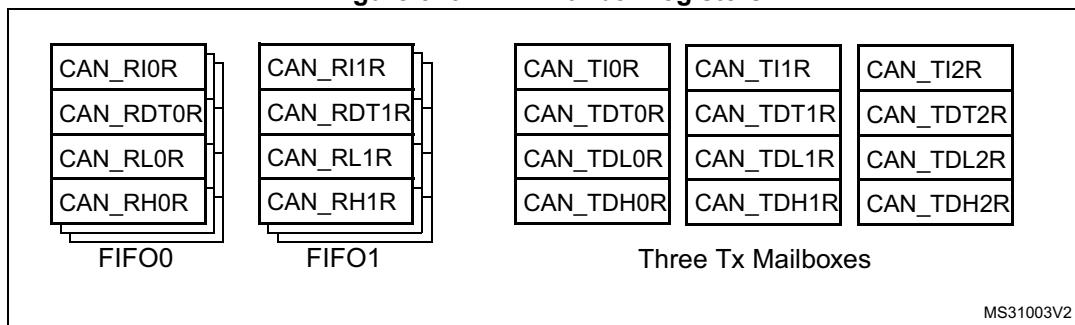
Transmit and receive mailboxes have the same registers except:

- The FMI field in the CAN\_RDTxR register.
- A receive mailbox is always write protected.
- A transmit mailbox is write-enabled only while empty, corresponding TME bit in the CAN\_TSR register set.

There are three TX mailboxes and two RX mailboxes. Each RX Mailbox allows access to a 3-level depth FIFO, the access being offered only to the oldest received message in the FIFO.

Each mailbox consist of four registers.

**Figure 326. CAN mailbox registers**



#### CAN TX mailbox identifier register (CAN\_TIxR) ( $x = 0..2$ )

Address offsets: 0x180, 0x190, 0x1A0

Reset value: 0xFFFF XXXX (except bit 0, TXRQ = 0)

All TX registers are write protected when the mailbox is pending transmission (TME<sub>Ex</sub> reset).

This register also implements the TX request control (bit 0) - reset value 0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
STID[10:0]/EXID[28:18]												EXID[17:13]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXID[12:0]												IDE	RTR	TXRQ	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:21 **STID[10:0]/EXID[28:18]**: Standard identifier or extended identifier

The standard identifier or the MSBs of the extended identifier (depending on the IDE bit value).

Bit 20:3 **EXID[17:0]**: Extended identifier

The LSBs of the extended identifier.

Bit 2 **IDE**: Identifier extension

This bit defines the identifier type of message in the mailbox.

- 0: Standard identifier.
- 1: Extended identifier.

Bit 1 **RTR**: Remote transmission request

- 0: Data frame
- 1: Remote frame

Bit 0 **TXRQ**: Transmit mailbox request

Set by software to request the transmission for the corresponding mailbox.

Cleared by hardware when the mailbox becomes empty.

### CAN mailbox data length control and time stamp register (CAN\_TDTxR) (x = 0..2)

All bits of this register are write protected when the mailbox is not in empty state.

Address offsets: 0x184, 0x194, 0x1A4

Reset value: 0xFFFF XXXX

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

Bits 31:16 **TIME[15:0]**: Message time stamp

This field contains the 16-bit timer value captured at the SOF transmission.

Bits 15:9 Reserved, must be kept at reset value.

Bit 8 **TGT**: Transmit global time

This bit is active only when the hardware is in the Time Trigger Communication mode, TTTCM bit of the CAN\_MCR register is set.

0: Time stamp TIME[15:0] is not sent.

1: Time stamp TIME[15:0] value is sent in the last two data bytes of the 8-byte message: TIME[7:0] in data byte 7 and TIME[15:8] in data byte 6, replacing the data written in CAN\_TDHR[x][31:16] register (DATA6[7:0] and DATA7[7:0]). DLC must be programmed as 8 in order these two bytes to be sent over the CAN bus.

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

Bits 3:0 **DLC[3:0]**: Data length code

This field defines the number of data bytes a data frame contains or a remote frame request. A message can contain from 0 to 8 data bytes, depending on the value in the DLC field.

### CAN mailbox data low register (CAN\_TDLxR) (x = 0..2)

All bits of this register are write protected when the mailbox is not in empty state.

Address offsets: 0x188, 0x198, 0x1A8

Reset value: 0xFFFF XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA3[7:0]								DATA2[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA1[7:0]								DATA0[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **DATA3[7:0]**: Data byte 3

Data byte 3 of the message.

Bits 23:16 **DATA2[7:0]**: Data byte 2

Data byte 2 of the message.

Bits 15:8 **DATA1[7:0]**: Data byte 1

Data byte 1 of the message.

Bits 7:0 **DATA0[7:0]**: Data byte 0

Data byte 0 of the message.

A message can contain from 0 to 8 data bytes and starts with byte 0.

### CAN mailbox data high register (CAN\_TDhxR) (x = 0..2)

All bits of this register are write protected when the mailbox is not in empty state.

Address offsets: 0x18C, 0x19C, 0x1AC

Reset value: 0xFFFF XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA7[7:0]								DATA6[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA5[7:0]								DATA4[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **DATA7[7:0]**: Data byte 7

Data byte 7 of the message.

Note: If TGT of this message and TTCM are active, DATA7 and DATA6 are replaced by the TIME stamp value.

Bits 23:16 **DATA6[7:0]**: Data byte 6

Data byte 6 of the message.

Bits 15:8 **DATA5[7:0]**: Data byte 5

Data byte 5 of the message.

Bits 7:0 **DATA4[7:0]**: Data byte 4

Data byte 4 of the message.

**CAN receive FIFO mailbox identifier register (CAN\_RIxR) (x = 0..1)**

Address offsets: 0x1B0, 0x1C0

Reset value: 0XXXXX XXXX

All RX registers are write protected.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
STID[10:0]/EXID[28:18]														EXID[17:13]	
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXID[12:0]														IDE	RTR
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	Res

Bits 31:21 **STID[10:0]/EXID[28:18]**: Standard identifier or extended identifier

The standard identifier or the MSBs of the extended identifier (depending on the IDE bit value).

Bits 20:3 **EXID[17:0]**: Extended identifier

The LSBs of the extended identifier.

Bit 2 **IDE**: Identifier extension

This bit defines the identifier type of message in the mailbox.

0: Standard identifier.

1: Extended identifier.

Bit 1 **RTR**: Remote transmission request

0: Data frame

1: Remote frame

Bit 0 Reserved, must be kept at reset value.

**CAN receive FIFO mailbox data length control and time stamp register (CAN\_RDTxR) (x = 0..1)**

Address offsets: 0x1B4, 0x1C4

Reset value: 0XXXXX XXXX

All RX registers are write protected.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TIME[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FMI[7:0]								Res.	Res.	Res.	Res.	DLC[3:0]			
r	r	r	r	r	r	r	r					r	r	r	r

Bits 31:16 **TIME[15:0]**: Message time stamp

This field contains the 16-bit timer value captured at the SOF detection.

Bits 15:8 **FMI[7:0]**: Filter match index

This register contains the index of the filter the message stored in the mailbox passed through. For more details on identifier filtering, refer to [Section 29.7.4: Identifier filtering](#).

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

Bits 3:0 **DLC[3:0]**: Data length code

This field defines the number of data bytes a data frame contains (0 to 8). It is 0 in the case of a remote frame request.

### CAN receive FIFO mailbox data low register (CAN\_RDLxR) (x = 0..1)

All bits of this register are write protected when the mailbox is not in empty state.

Address offsets: 0x1B8, 0x1C8

Reset value: 0XXXX XXXX

All RX registers are write protected.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA3[7:0]								DATA2[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA1[7:0]								DATA0[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:24 **DATA3[7:0]**: Data byte 3

Data byte 3 of the message.

Bits 23:16 **DATA2[7:0]**: Data byte 2

Data byte 2 of the message.

Bits 15:8 **DATA1[7:0]**: Data byte 1

Data byte 2 of the message.

Bits 7:0 **DATA0[7:0]**: Data byte 0

Data byte 0 of the message.

A message can contain from 0 to 8 data bytes and starts with byte 0.

### CAN receive FIFO mailbox data high register (CAN\_RDHxR) (x = 0..1)

Address offsets: 0x1BC, 0x1CC

Reset value: 0XXXX XXXX

All RX registers are write protected.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA7[7:0]								DATA6[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA5[7:0]								DATA4[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:24 **DATA7[7:0]**: Data byte 7  
Data byte 3 of the message.

Bits 23:16 **DATA6[7:0]**: Data byte 6  
Data byte 2 of the message.

Bits 15:8 **DATA5[7:0]**: Data byte 5  
Data byte 1 of the message.

Bits 7:0 **DATA4[7:0]**: Data byte 4  
Data byte 0 of the message.

#### 29.9.4 CAN filter registers

##### CAN filter master register (CAN\_FMR)

Address offset: 0x200

Reset value: 0x2A1C 0E01

All bits of this register are set and cleared by software.

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

Bits 31:1 Reserved, must be kept at reset value.

Bit 0 **FINIT**: Filter initialization mode

Initialization mode for filter banks

0: Active filters mode.

1: Initialization mode for the filters.

##### CAN filter mode register (CAN\_FM1R)

Address offset: 0x204

Reset value: 0x0000 0000

This register can be written only when the filter initialization mode is set (FINIT=1) in the CAN\_FMR register.

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

Res.	Res.	FBM13	FBM12	FBM11	FBM10	FBM9	FBM8	FBM7	FBM6	FBM5	FBM4	FBM3	FBM2	FBM1	FBM0
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Note: Refer to [Figure 319: Filter bank scale configuration - Register organization](#).

Bits 31:14 Reserved, must be kept at reset value.

Bits 13:0 **FBMx**: Filter mode

Mode of the registers of Filter x.

0: Two 32-bit registers of filter bank x are in Identifier Mask mode.

1: Two 32-bit registers of filter bank x are in Identifier List mode.

### CAN filter scale register (CAN\_FS1R)

Address offset: 0x20C

Reset value: 0x0000 0000

This register can be written only when the filter initialization mode is set (FINIT=1) in the CAN\_FMR register.

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

Res.	Res.	FSC13	FSC12	FSC11	FSC10	FSC9	FSC8	FSC7	FSC6	FSC5	FSC4	FSC3	FSC2	FSC1	FSC0
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:14 Reserved, must be kept at reset value.

Bits 13:0 **FSCx**: Filter scale configuration

These bits define the scale configuration of Filters 13-0.

0: Dual 16-bit scale configuration

1: Single 32-bit scale configuration

Note: Refer to [Figure 319: Filter bank scale configuration - Register organization](#).

### CAN filter FIFO assignment register (CAN\_FFA1R)

Address offset: 0x214

Reset value: 0x0000 0000

This register can be written only when the filter initialization mode is set (FINIT=1) in the CAN\_FMR register.

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

Res.	Res.	FFA13	FFA12	FFA11	FFA10	FFA9	FFA8	FFA7	FFA6	FFA5	FFA4	FFA3	FFA2	FFA1	FFA0
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:14 Reserved, must be kept at reset value.

Bits 13:0 **FFAx**: Filter FIFO assignment for filter x

The message passing through this filter is stored in the specified FIFO.

0: Filter assigned to FIFO 0

1: Filter assigned to FIFO 1

### CAN filter activation register (CAN\_FA1R)

Address offset: 0x21C

Reset value: 0x0000 0000

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

Res.	Res.	FACT1 3	FACT1 2	FACT1 1	FACT1 0	FACT9	FACT8	FACT7	FACT6	FACT5	FACT4	FACT3	FACT2	FACT1	FACT0
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:14 Reserved, must be kept at reset value.

Bits 13:0 **FACTx**: Filter active

The software sets this bit to activate Filter x. To modify the Filter x registers (CAN\_FxR[0:7]), the FACTx bit must be cleared or the FINIT bit of the CAN\_FMR register must be set.

0: Filter x is not active

1: Filter x is active

### Filter bank i register x (CAN\_FiRx) (i = 0..13, x = 1, 2)

Address offsets: 0x240 to 0x2AC

Reset value: 0XXXXX XXXX

There are 14 filter banks, i= 0 to 13. Each filter bank i is composed of two 32-bit registers, CAN\_FiR[2:1].

This register can only be modified when the FACTx bit of the CAN\_FAxR register is cleared or when the FINIT bit of the CAN\_FMR register is set.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FB31	FB30	FB29	FB28	FB27	FB26	FB25	FB24	FB23	FB22	FB21	FB20	FB19	FB18	FB17	FB16
rw															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FB15	FB14	FB13	FB12	FB11	FB10	FB9	FB8	FB7	FB6	FB5	FB4	FB3	FB2	FB1	FB0
rw															

In all configurations:

Bits 31:0 **FB[31:0]**: Filter bits

#### Identifier

Each bit of the register specifies the level of the corresponding bit of the expected identifier.  
0: Dominant bit is expected

1: Recessive bit is expected

#### Mask

Each bit of the register specifies whether the bit of the associated identifier register must match with the corresponding bit of the expected identifier or not.

0: Do not care, the bit is not used for the comparison

1: Must match, the bit of the incoming identifier must have the same level has specified in the corresponding identifier register of the filter.

**Note:** Depending on the scale and mode configuration of the filter, the function of each register can differ. For the filter mapping, functions description and mask registers association, refer to [Section 29.7.4: Identifier filtering](#).

A Mask/Identifier register in **mask mode** has the same bit mapping as in **identifier list mode**.

For the register mapping/addresses of the filter banks refer to [Table 120](#).

## 29.9.5 bxCAN register map

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

**Table 120. bxCAN register map and reset values**

Offset	Register	Reset value	31
0x000	<b>CAN_MCR</b>	Res.	Res.
0x004	<b>CAN_MSR</b>	Res.	Res.
0x008	<b>CAN_TSR</b>	0 LOW[2:0]	30
0x00C	<b>CAN_RF0R</b>	Res.	Res.
0x010	<b>CAN_RF1R</b>	Res.	Res.
0x014	<b>CAN_IER</b>	Res.	Res.
0x018	<b>CAN_ESR</b>	REC[7:0]	29
0x01C	<b>CAN_BTR</b>	TEC[7:0]	28
0x020-0x17F	-	TS2[2:0]	27
0x180	<b>CAN_TI0R</b>	TS1[3:0]	26
		BRP[9:0]	25
		EXID[17:0]	24
		STID[10:0]/EXID[28:18]	23
		EWGIE	22
		ERRIE	21
		SLKIE	20
		WRUIE	19
		RES	18
		ALST2	17
		TXOK2	16
		RCACP2	15
		ABRQ1	14
		DBF	13
		RESET	12
		ABOM	11
		TERR1	10
		RX	9
		SAMP	8
		TXM	7
		TTCM	6
		RES	5
		AWUM	4
		SLAKI	3
		NART	2
		RFLM	1
		ERRI	1
		TXFP	0
		ALST0	0
		TXOK0	0
		SLEEP	0
		INAK	0
		INRQ	0

Table 120. bxCAN register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x184	CAN_TDT0R	TIME[15:0]															Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TGT	Res.	Res.	Res.	Res.	DLC[3:0]			
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x			
0x188	CAN_TDL0R	DATA3[7:0]							DATA2[7:0]							DATA1[7:0]							DATA0[7:0]										
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x				
0x18C	CAN_TDH0R	DATA7[7:0]							DATA6[7:0]							DATA5[7:0]							DATA4[7:0]										
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x				
0x190	CAN_TI1R	STID[10:0]/EXID[28:18]															EXID[17:0]													IDE			
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0				
0x194	CAN_TDT1R	TIME[15:0]															Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DLC[3:0]				
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x					
0x198	CAN_TDL1R	DATA3[7:0]							DATA2[7:0]							DATA1[7:0]							DATA0[7:0]										
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x					
0x19C	CAN_TDH1R	DATA7[7:0]							DATA6[7:0]							DATA5[7:0]							DATA4[7:0]										
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x					
0x1A0	CAN_TI2R	STID[10:0]/EXID[28:18]															EXID[17:0]													IDE			
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0					
0x1A4	CAN_TDT2R	TIME[15:0]															Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DLC[3:0]					
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x						
0x1A8	CAN_TDL2R	DATA3[7:0]							DATA2[7:0]							DATA1[7:0]							DATA0[7:0]										
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x					
0x1AC	CAN_TDH2R	DATA7[7:0]							DATA6[7:0]							DATA5[7:0]							DATA4[7:0]										
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x					
0x1B0	CAN_RI0R	STID[10:0]/EXID[28:18]															EXID[17:0]													IDE			
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	-					

**Table 120. bxCAN register map and reset values (continued)**

**Table 120. bxCAN register map and reset values (continued)**

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 30 Universal serial bus full-speed device interface (USB)

This section applies to STM32F04x, STM32F072 and STM32F078 devices only.

### 30.1 Introduction

The USB peripheral implements an interface between a full-speed USB 2.0 bus and the APB bus.

USB suspend/resume are supported, which allows to stop the device clocks for low-power consumption.

### 30.2 USB main features

- USB specification version 2.0 full-speed compliant
- Configurable number of endpoints from 1 to 8
- Up to 1024 bytes of dedicated packet buffer memory SRAM (last 256 Bytes are exclusively shared with CAN peripheral)
- Cyclic redundancy check (CRC) generation/checking, Non-return-to-zero Inverted (NRZI) encoding/decoding and bit-stuffing
- Isochronous transfers support
- Double-buffered bulk/isochronous endpoint support
- USB Suspend/Resume operations
- Frame locked clock pulse generation
- USB 2.0 Link Power Management support
- Battery Charging Specification Revision 1.2 support
- USB connect / disconnect capability (controllable embedded pull-up resistor on USB\_DP line)

### 30.3 USB implementation

*Table 121* describes the USB implementation in the devices.

**Table 121. STM32F0xx USB implementation**

USB features <sup>(1)</sup>	STM32F04x, STM32F072, STM32F078
USB	
Number of endpoints	8
Size of dedicated packet buffer memory SRAM	1024 bytes <sup>(2)</sup>
Dedicated packet buffer memory SRAM access scheme	2 x 16 bits / word
USB 2.0 Link Power Management (LPM) support	X

**Table 121. STM32F0xx USB implementation (continued)**

<b>USB features<sup>(1)</sup></b>	<b>STM32F04x, STM32F072, STM32F078</b>
	<b>USB</b>
Battery Charging Detection (BCD) support	X
Embedded pull-up resistor on USB_DP line	X

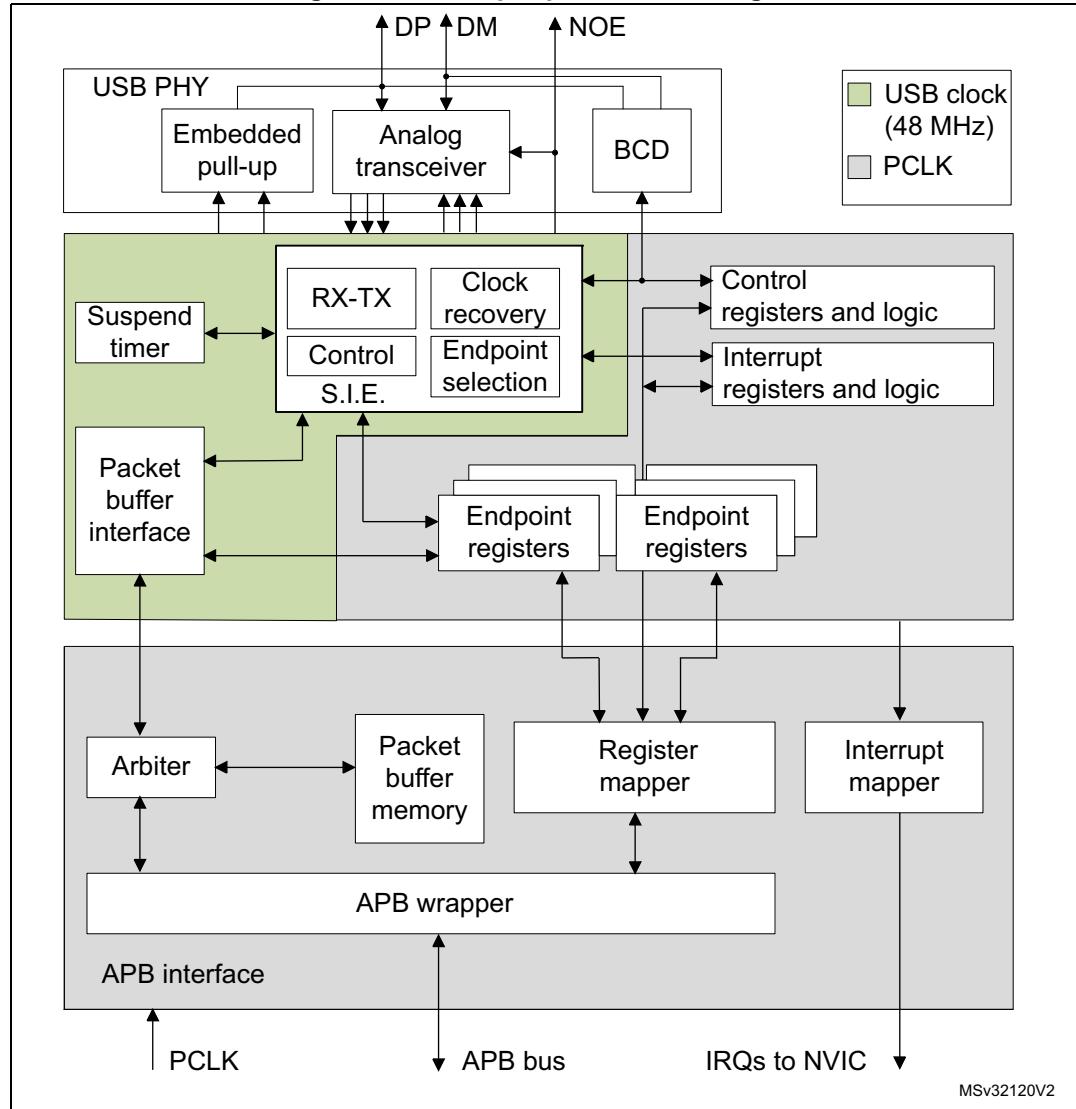
1. X= supported

2. When the CAN peripheral clock is enabled in the RCC\_APB1ENR register, only the first 768 Bytes are available to USB while the last 256 Bytes are used by CAN.

## 30.4 USB functional description

*Figure 327* shows the block diagram of the USB peripheral.

**Figure 327. USB peripheral block diagram**



The USB peripheral provides an USB-compliant connection between the host PC and the function implemented by the microcontroller. Data transfer between the host PC and the system memory occurs through a dedicated packet buffer memory accessed directly by the USB peripheral. This dedicated memory size is up to 1024 bytes, and up to 16 mono-directional or 8 bidirectional endpoints can be used. The USB peripheral interfaces with the USB host, detecting token packets, handling data transmission/reception, and processing handshake packets as required by the USB standard. Transaction formatting is performed by the hardware, including CRC generation and checking.

Each endpoint is associated with a buffer description block indicating where the endpoint-related memory area is located, how large it is or how many bytes must be transmitted. When a token for a valid function/endpoint pair is recognized by the USB peripheral, the related data transfer (if required and if the endpoint is configured) takes

place. The data buffered by the USB peripheral is loaded in an internal 16-bit register and memory access to the dedicated buffer is performed. When all the data has been transferred, if needed, the proper handshake packet over the USB is generated or expected according to the direction of the transfer.

At the end of the transaction, an endpoint-specific interrupt is generated, reading status registers and/or using different interrupt response routines. The microcontroller can determine:

- which endpoint has to be served,
- which type of transaction took place, if errors occurred (bit stuffing, format, CRC, protocol, missing ACK, over/underrun, etc.).

Special support is offered to isochronous transfers and high throughput bulk transfers, implementing a double buffer usage, which allows to always have an available buffer for the USB peripheral while the microcontroller uses the other one.

The unit can be placed in low-power mode (SUSPEND mode), by writing in the control register, whenever required. At this time, all static power dissipation is avoided, and the USB clock can be slowed down or stopped. The detection of activity at the USB inputs, while in low-power mode, wakes the device up asynchronously. A special interrupt source can be connected directly to a wake-up line to allow the system to immediately restart the normal clock generation and/or support direct clock start/stop.

### 30.4.1 Description of USB blocks

The USB peripheral implements all the features related to USB interfacing, which include the following blocks:

- USB Physical Interface (USB PHY): This block is maintaining the electrical interface to an external USB host. It contains the differential analog transceiver itself, controllable embedded pull-up resistor (connected to USB\_DP line) and support for Battery Charging Detection (BCD), multiplexed on same USB\_DP and USB\_DM lines. The output enable control signal of the analog transceiver (active low) is provided externally on USB\_NOE. It can be used to drive some activity LED or to provide information about the actual communication direction to some other circuitry.
- Serial Interface Engine (SIE): The functions of this block include: synchronization pattern recognition, bit-stuffing, CRC generation and checking, PID verification/generation, and handshake evaluation. It must interface with the USB transceivers and uses the virtual buffers provided by the packet buffer interface for local data storage. This unit also generates signals according to USB peripheral events, such as Start of Frame (SOF), USB\_Reset, Data errors etc. and to Endpoint related events like end of transmission or correct reception of a packet; these signals are then used to generate interrupts.
- Timer: This block generates a start-of-frame locked clock pulse and detects a global suspend (from the host) when no traffic has been received for 3 ms.
- Packet Buffer Interface: This block manages the local memory implementing a set of buffers in a flexible way, both for transmission and reception. It can choose the proper buffer according to requests coming from the SIE and locate them in the memory addresses pointed by the Endpoint registers. It increments the address after each exchanged byte until the end of packet, keeping track of the number of exchanged bytes and preventing the buffer to overrun the maximum capacity.

- Endpoint-Related Registers: Each endpoint has an associated register containing the endpoint type and its current status. For mono-directional/single-buffer endpoints, a single register can be used to implement two distinct endpoints. The number of registers is 8, allowing up to 16 mono-directional/single-buffer or up to 7 double-buffer endpoints in any combination. For example the USB peripheral can be programmed to have 4 double buffer endpoints and 8 single-buffer/mono-directional endpoints.
- Control Registers: These are the registers containing information about the status of the whole USB peripheral and used to force some USB events, such as resume and power-down.
- Interrupt Registers: These contain the Interrupt masks and a record of the events. They can be used to inquire an interrupt reason, the interrupt status or to clear the status of a pending interrupt.

Note:

\* *Endpoint 0 is always used for control transfer in single-buffer mode.*

The USB peripheral is connected to the APB bus through an APB interface, containing the following blocks:

- Packet Memory: This is the local memory that physically contains the Packet Buffers. It can be used by the Packet Buffer interface, which creates the data structure and can be accessed directly by the application software. The size of the Packet Memory is up to 1024 bytes, structured as 512 half-words of 16 bits.
- Arbiter: This block accepts memory requests coming from the APB bus and from the USB interface. It resolves the conflicts by giving priority to APB accesses, while always reserving half of the memory bandwidth to complete all USB transfers. This time-duplex scheme implements a virtual dual-port SRAM that allows memory access, while an USB transaction is happening. Multiword APB transfers of any length are also allowed by this scheme.
- Register Mapper: This block collects the various byte-wide and bit-wide registers of the USB peripheral in a structured 16-bit wide half-word set addressed by the APB.
- APB Wrapper: This provides an interface to the APB for the memory and register. It also maps the whole USB peripheral in the APB address space.
- Interrupt Mapper: This block is used to select how the possible USB events can generate interrupts and map them to the NVIC.

## 30.5 Programming considerations

In the following sections, the expected interactions between the USB peripheral and the application program are described, in order to ease application software development.

### 30.5.1 Generic USB device programming

This part describes the main tasks required of the application software in order to obtain USB compliant behavior. The actions related to the most general USB events are taken into account and paragraphs are dedicated to the special cases of double-buffered endpoints and Isochronous transfers. Apart from system reset, action is always initiated by the USB peripheral, driven by one of the USB events described below.

### 30.5.2 System and power-on reset

Upon system and power-on reset, the first operation the application software should perform is to provide all required clock signals to the USB peripheral and subsequently de-assert its reset signal so to be able to access its registers. The whole initialization sequence is hereafter described.

As a first step application software needs to activate register macrocell clock and de-assert macrocell specific reset signal using related control bits provided by device clock management logic.

After that, the analog part of the device related to the USB transceiver must be switched on using the PDWN bit in CNTR register, which requires a special handling. This bit is intended to switch on the internal voltage references that supply the port transceiver. This circuit has a defined startup time ( $t_{STARTUP}$  specified in the datasheet) during which the behavior of the USB transceiver is not defined. It is thus necessary to wait this time, after setting the PDWN bit in the CNTR register, before removing the reset condition on the USB part (by clearing the FRES bit in the CNTR register). Clearing the ISTR register then removes any spurious pending interrupt before any other macrocell operation is enabled.

At system reset, the microcontroller must initialize all required registers and the packet buffer description table, to make the USB peripheral able to properly generate interrupts and data transfers. All registers not specific to any endpoint must be initialized according to the needs of application software (choice of enabled interrupts, chosen address of packet buffers, etc.). Then the process continues as for the USB reset case (see further paragraph).

#### USB reset (RESET interrupt)

When this event occurs, the USB peripheral is put in the same conditions it is left by the system reset after the initialization described in the previous paragraph: communication is disabled in all endpoint registers (the USB peripheral will not respond to any packet). As a response to the USB reset event, the USB function must be enabled, having as USB address 0, implementing only the default control endpoint (endpoint address is 0 too). This is accomplished by setting the Enable Function (EF) bit of the USB\_DADDR register and initializing the EP0R register and its related packet buffers accordingly. During USB enumeration process, the host assigns a unique address to this device, which must be written in the ADD[6:0] bits of the USB\_DADDR register, and configures any other necessary endpoint.

When a RESET interrupt is received, the application software is responsible to enable again the default endpoint of USB function 0 within 10 ms from the end of reset sequence which triggered the interrupt.

#### Structure and usage of packet buffers

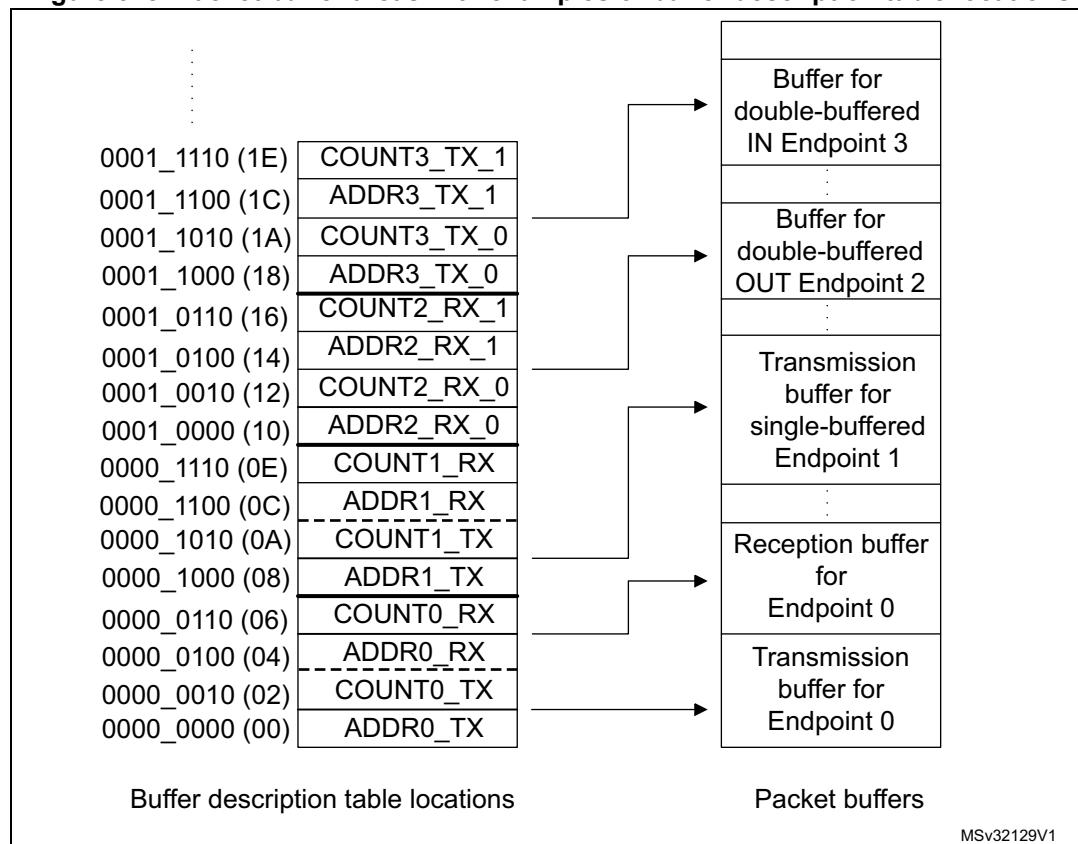
Each bidirectional endpoint may receive or transmit data from/to the host. The received data is stored in a dedicated memory buffer reserved for that endpoint, while another memory buffer contains the data to be transmitted by the endpoint. Access to this memory is performed by the packet buffer interface block, which delivers a memory access request and waits for its acknowledgment. Since the packet buffer memory has to be accessed by the microcontroller also, an arbitration logic takes care of the access conflicts, using half APB cycle for microcontroller access and the remaining half for the USB peripheral access. In this way, both the agents can operate as if the packet memory is a dual-port SRAM, without being aware of any conflict even when the microcontroller is performing back-to-

back accesses. The USB peripheral logic uses a dedicated clock. The frequency of this dedicated clock is fixed by the requirements of the USB standard at 48 MHz, and this can be different from the clock used for the interface to the APB bus. Different clock configurations are possible where the APB clock frequency can be higher or lower than the USB peripheral one.

**Note:** *Due to USB data rate and packet memory interface requirements, the APB clock must have a minimum frequency of 10 MHz to avoid data overrun/underrun problems.*

Each endpoint is associated with two packet buffers (usually one for transmission and the other one for reception). Buffers can be placed anywhere inside the packet memory because their location and size is specified in a buffer description table, which is also located in the packet memory at the address indicated by the USB\_BTABLE register. Each table entry is associated to an endpoint register and it is composed of four 16-bit half-words so that table start address must always be aligned to an 8-byte boundary (the lowest three bits of USB\_BTABLE register are always "000"). Buffer descriptor table entries are described in the [Section 30.6.2: Buffer descriptor table](#). If an endpoint is unidirectional and it is neither an Isochronous nor a double-buffered bulk, only one packet buffer is required (the one related to the supported transfer direction). Other table locations related to unsupported transfer directions or unused endpoints, are available to the user. Isochronous and double-buffered bulk endpoints have special handling of packet buffers (Refer to [Section 30.5.4: Isochronous transfers](#) and [Section 30.5.3: Double-buffered endpoints](#) respectively). The relationship between buffer description table entries and packet buffer areas is depicted in [Figure 328](#).

**Figure 328. Packet buffer areas with examples of buffer description table locations**



Each packet buffer is used either during reception or transmission starting from the bottom. The USB peripheral will never change the contents of memory locations adjacent to the allocated memory buffers; if a packet bigger than the allocated buffer length is received (buffer overrun condition) the data are copied to the memory only up to the last available location.

### Endpoint initialization

The first step to initialize an endpoint is to write appropriate values to the ADDRn\_TX/ADDRn\_RX registers so that the USB peripheral finds the data to be transmitted already available and the data to be received can be buffered. The EP\_TYPE bits in the USB\_EPnR register must be set according to the endpoint type, eventually using the EP\_KIND bit to enable any special required feature. On the transmit side, the endpoint must be enabled using the STAT\_TX bits in the USB\_EPnR register and COUNTn\_TX must be initialized. For reception, STAT\_RX bits must be set to enable reception and COUNTn\_RX must be written with the allocated buffer size using the BL\_SIZE and NUM\_BLOCK fields. Unidirectional endpoints, except Isochronous and double-buffered bulk endpoints, need to initialize only bits and registers related to the supported direction. Once the transmission and/or reception are enabled, register USB\_EPnR and locations ADDRn\_TX/ADDRn\_RX, COUNTn\_TX/COUNTn\_RX (respectively), should not be modified by the application software, as the hardware can change their value on the fly. When the data transfer operation is completed, notified by a CTR interrupt event, they can be accessed again to re-enable a new operation.

### IN packets (data transmission)

When receiving an IN token packet, if the received address matches a configured and valid endpoint, the USB peripheral accesses the contents of ADDRn\_TX and COUNTn\_TX locations inside the buffer descriptor table entry related to the addressed endpoint. The content of these locations is stored in its internal 16 bit registers ADDR and COUNT (not accessible by software). The packet memory is accessed again to read the first byte to be transmitted (Refer to [Structure and usage of packet buffers on page 873](#)) and starts sending a DATA0 or DATA1 PID according to USB\_EPnR bit DTOG\_TX. When the PID is completed, the first byte, read from buffer memory, is loaded into the output shift register to be transmitted on the USB bus. After the last data byte is transmitted, the computed CRC is sent. If the addressed endpoint is not valid, a NAK or STALL handshake packet is sent instead of the data packet, according to STAT\_TX bits in the USB\_EPnR register.

The ADDR internal register is used as a pointer to the current buffer memory location while COUNT is used to count the number of remaining bytes to be transmitted. Each half-word read from the packet buffer memory is transmitted over the USB bus starting from the least significant byte. Transmission buffer memory is read starting from the address pointed by ADDRn\_TX for COUNTn\_TX/2 half-words. If a transmitted packet is composed of an odd number of bytes, only the lower half of the last half-word accessed is used.

On receiving the ACK receipt by the host, the USB\_EPnR register is updated in the following way: DTOG\_TX bit is toggled, the endpoint is made invalid by setting STAT\_TX=10 (NAK) and bit CTR\_TX is set. The application software must first identify the endpoint, which is requesting microcontroller attention by examining the EP\_ID and DIR bits in the USB\_ISTR register. Servicing of the CTR\_TX event starts clearing the interrupt bit; the application software then prepares another buffer full of data to be sent, updates the COUNTn\_TX table location with the number of byte to be transmitted during the next transfer, and finally sets STAT\_TX to '11 (VALID) to re-enable transmissions. While the STAT\_TX bits are equal to '10 (NAK), any IN request addressed to that endpoint is NAKed,

indicating a flow control condition: the USB host will retry the transaction until it succeeds. It is mandatory to execute the sequence of operations in the above mentioned order to avoid losing the notification of a second IN transaction addressed to the same endpoint immediately following the one which triggered the CTR interrupt.

### OUT and SETUP packets (data reception)

These two tokens are handled by the USB peripheral more or less in the same way; the differences in the handling of SETUP packets are detailed in the following paragraph about control transfers. When receiving an OUT/SETUP PID, if the address matches a valid endpoint, the USB peripheral accesses the contents of the ADDRn\_RX and COUNTn\_RX locations inside the buffer descriptor table entry related to the addressed endpoint. The content of the ADDRn\_RX is stored directly in its internal register ADDR. While COUNT is now reset and the values of BL\_SIZE and NUM\_BLOCK bit fields, which are read within COUNTn\_RX content are used to initialize BUF\_COUNT, an internal 16 bit counter, which is used to check the buffer overrun condition (all these internal registers are not accessible by software). Data bytes subsequently received by the USB peripheral are packed in half-words (the first byte received is stored as least significant byte) and then transferred to the packet buffer starting from the address contained in the internal ADDR register while BUF\_COUNT is decremented and COUNT is incremented at each byte transfer. When the end of DATA packet is detected, the correctness of the received CRC is tested and only if no errors occurred during the reception, an ACK handshake packet is sent back to the transmitting host.

In case of wrong CRC or other kinds of errors (bit-stuff violations, frame errors, etc.), data bytes are still copied in the packet memory buffer, at least until the error detection point, but ACK packet is not sent and the ERR bit in USB\_ISTR register is set. However, there is usually no software action required in this case: the USB peripheral recovers from reception errors and remains ready for the next transaction to come. If the addressed endpoint is not valid, a NAK or STALL handshake packet is sent instead of the ACK, according to bits STAT\_RX in the USB\_EPnR register and no data is written in the reception memory buffers.

Reception memory buffer locations are written starting from the address contained in the ADDRn\_RX for a number of bytes corresponding to the received data packet length, CRC included (i.e. data payload length + 2), or up to the last allocated memory location, as defined by BL\_SIZE and NUM\_BLOCK, whichever comes first. In this way, the USB peripheral never writes beyond the end of the allocated reception memory buffer area. If the length of the data packet payload (actual number of bytes used by the application) is greater than the allocated buffer, the USB peripheral detects a buffer overrun condition. in this case, a STALL handshake is sent instead of the usual ACK to notify the problem to the host, no interrupt is generated and the transaction is considered failed.

When the transaction is completed correctly, by sending the ACK handshake packet, the internal COUNT register is copied back in the COUNTn\_RX location inside the buffer description table entry, leaving unaffected BL\_SIZE and NUM\_BLOCK fields, which normally do not require to be re-written, and the USB\_EPnR register is updated in the following way: DTOG\_RX bit is toggled, the endpoint is made invalid by setting STAT\_RX = '10 (NAK) and bit CTR\_RX is set. If the transaction has failed due to errors or buffer overrun condition, none of the previously listed actions take place. The application software must first identify the endpoint, which is requesting microcontroller attention by examining the EP\_ID and DIR bits in the USB\_ISTR register. The CTR\_RX event is serviced by first determining the transaction type (SETUP bit in the USB\_EPnR register); the application software must clear the interrupt flag bit and get the number of received bytes reading the COUNTn\_RX location inside the buffer description table entry related to the endpoint being

processed. After the received data is processed, the application software should set the STAT\_RX bits to '11 (Valid) in the USB\_EPnR, enabling further transactions. While the STAT\_RX bits are equal to '10 (NAK), any OUT request addressed to that endpoint is NAKed, indicating a flow control condition: the USB host will retry the transaction until it succeeds. It is mandatory to execute the sequence of operations in the above mentioned order to avoid losing the notification of a second OUT transaction addressed to the same endpoint following immediately the one which triggered the CTR interrupt.

### Control transfers

Control transfers are made of a SETUP transaction, followed by zero or more data stages, all of the same direction, followed by a status stage (a zero-byte transfer in the opposite direction). SETUP transactions are handled by control endpoints only and are very similar to OUT ones (data reception) except that the values of DTOG\_TX and DTOG\_RX bits of the addressed endpoint registers are set to 1 and 0 respectively, to initialize the control transfer, and both STAT\_TX and STAT\_RX are set to '10 (NAK) to let software decide if subsequent transactions must be IN or OUT depending on the SETUP contents. A control endpoint must check SETUP bit in the USB\_EPnR register at each CTR\_RX event to distinguish normal OUT transactions from SETUP ones. A USB device can determine the number and direction of data stages by interpreting the data transferred in the SETUP stage, and is required to STALL the transaction in the case of errors. To do so, at all data stages before the last, the unused direction should be set to STALL, so that, if the host reverses the transfer direction too soon, it gets a STALL as a status stage.

While enabling the last data stage, the opposite direction should be set to NAK, so that, if the host reverses the transfer direction (to perform the status stage) immediately, it is kept waiting for the completion of the control operation. If the control operation completes successfully, the software will change NAK to VALID, otherwise to STALL. At the same time, if the status stage is an OUT, the STATUS\_OUT (EP\_KIND in the USB\_EPnR register) bit should be set, so that an error is generated if a status transaction is performed with not-zero data. When the status transaction is serviced, the application clears the STATUS\_OUT bit and sets STAT\_RX to VALID (to accept a new command) and STAT\_TX to NAK (to delay a possible status stage immediately following the next setup).

Since the USB specification states that a SETUP packet cannot be answered with a handshake different from ACK, eventually aborting a previously issued command to start the new one, the USB logic doesn't allow a control endpoint to answer with a NAK or STALL packet to a SETUP token received from the host.

When the STAT\_RX bits are set to '01 (STALL) or '10 (NAK) and a SETUP token is received, the USB accepts the data, performing the required data transfers and sends back an ACK handshake. If that endpoint has a previously issued CTR\_RX request not yet acknowledged by the application (i.e. CTR\_RX bit is still set from a previously completed reception), the USB discards the SETUP transaction and does not answer with any handshake packet regardless of its state, simulating a reception error and forcing the host to send the SETUP token again. This is done to avoid losing the notification of a SETUP transaction addressed to the same endpoint immediately following the transaction, which triggered the CTR\_RX interrupt.

### 30.5.3 Double-buffered endpoints

All different endpoint types defined by the USB standard represent different traffic models, and describe the typical requirements of different kind of data transfer operations. When large portions of data are to be transferred between the host PC and the USB function, the bulk endpoint type is the most suited model. This is because the host schedules bulk transactions so as to fill all the available bandwidth in the frame, maximizing the actual transfer rate as long as the USB function is ready to handle a bulk transaction addressed to it. If the USB function is still busy with the previous transaction when the next one arrives, it will answer with a NAK handshake and the host PC will issue the same transaction again until the USB function is ready to handle it, reducing the actual transfer rate due to the bandwidth occupied by re-transmissions. For this reason, a dedicated feature called 'double-buffering' can be used with bulk endpoints.

When 'double-buffering' is activated, data toggle sequencing is used to select, which buffer is to be used by the USB peripheral to perform the required data transfers, using both 'transmission' and 'reception' packet memory areas to manage buffer swapping on each successful transaction in order to always have a complete buffer to be used by the application, while the USB peripheral fills the other one. For example, during an OUT transaction directed to a 'reception' double-buffered bulk endpoint, while one buffer is being filled with new data coming from the USB host, the other one is available for the microcontroller software usage (the same would happen with a 'transmission' double-buffered bulk endpoint and an IN transaction).

Since the swapped buffer management requires the usage of all 4 buffer description table locations hosting the address pointer and the length of the allocated memory buffers, the USB\_EPnR registers used to implement double-buffered bulk endpoints are forced to be used as unidirectional ones. Therefore, only one STAT bit pair must be set at a value different from '00 (Disabled): STAT\_RX if the double-buffered bulk endpoint is enabled for reception, STAT\_TX if the double-buffered bulk endpoint is enabled for transmission. In case it is required to have double-buffered bulk endpoints enabled both for reception and transmission, two USB\_EPnR registers must be used.

To exploit the double-buffering feature and reach the highest possible transfer rate, the endpoint flow control structure, described in previous chapters, has to be modified, in order to switch the endpoint status to NAK only when a buffer conflict occurs between the USB peripheral and application software, instead of doing it at the end of each successful transaction. The memory buffer which is currently being used by the USB peripheral is defined by the DTOG bit related to the endpoint direction: DTOG\_RX (bit 14 of USB\_EPnR register) for 'reception' double-buffered bulk endpoints or DTOG\_TX (bit 6 of USB\_EPnR register) for 'transmission' double-buffered bulk endpoints. To implement the new flow control scheme, the USB peripheral should know which packet buffer is currently in use by the application software, so to be aware of any conflict. Since in the USB\_EPnR register, there are two DTOG bits but only one is used by USB peripheral for data and buffer sequencing (due to the unidirectional constraint required by double-buffering feature) the other one can be used by the application software to show which buffer it is currently using. This new buffer flag is called SW\_BUF. In the following table the correspondence between USB\_EPnR register bits and DTOG/SW\_BUF definition is explained, for the cases of 'transmission' and 'reception' double-buffered bulk endpoints.

**Table 122. Double-buffering buffer flag definition**

Buffer flag	'Transmission' endpoint	'Reception' endpoint
DTOG	DTOG_TX (USB_EPnR bit 6)	DTOG_RX (USB_EPnR bit 14)
SW_BUF	USB_EPnR bit 14	USB_EPnR bit 6

The memory buffer which is currently being used by the USB peripheral is defined by DTOG buffer flag, while the buffer currently in use by application software is identified by SW\_BUF buffer flag. The relationship between the buffer flag value and the used packet buffer is the same in both cases, and it is listed in the following table.

**Table 123. Bulk double-buffering memory buffers usage**

Endpoint type	DTOG	SW_BUF	Packet buffer used by USB peripheral	Packet buffer used by Application Software
IN	0	1	ADDRn_TX_0 / COUNTn_TX_0 Buffer description table locations.	ADDRn_TX_1 / COUNTn_TX_1 Buffer description table locations.
	1	0	ADDRn_TX_1 / COUNTn_TX_1 Buffer description table locations	ADDRn_TX_0 / COUNTn_TX_0 Buffer description table locations.
	0	0	None <sup>(1)</sup>	ADDRn_TX_0 / COUNTn_TX_0 Buffer description table locations.
	1	1	None <sup>(1)</sup>	ADDRn_TX_1 / COUNTn_TX_1 Buffer description table locations.
OUT	0	1	ADDRn_RX_0 / COUNTn_RX_0 Buffer description table locations.	ADDRn_RX_1 / COUNTn_RX_1 Buffer description table locations.
	1	0	ADDRn_RX_1 / COUNTn_RX_1 Buffer description table locations.	ADDRn_RX_0 / COUNTn_RX_0 Buffer description table locations.
	0	0	None <sup>(1)</sup>	ADDRn_RX_0 / COUNTn_RX_0 Buffer description table locations.
	1	1	None <sup>(1)</sup>	ADDRn_RX_1 / COUNTn_RX_1 Buffer description table locations.

1. Endpoint in NAK Status.

Double-buffering feature for a bulk endpoint is activated by:

- Writing EP\_TYPE bit field at '00 in its USB\_EPnR register, to define the endpoint as a bulk, and
- Setting EP\_KIND bit at '1 (DBL\_BUF), in the same register.

The application software is responsible for DTOG and SW\_BUF bits initialization according to the first buffer to be used; this has to be done considering the special toggle-only property that these two bits have. The end of the first transaction occurring after having set DBL\_BUF, triggers the special flow control of double-buffered bulk endpoints, which is used for all other transactions addressed to this endpoint until DBL\_BUF remain set. At the end of each transaction the CTR\_RX or CTR\_TX bit of the addressed endpoint USB\_EPnR register is set, depending on the enabled direction. At the same time, the affected DTOG bit in the USB\_EPnR register is hardware toggled making the USB peripheral buffer swapping completely software independent. Unlike common transactions, and the first one after

DBL\_BUF setting, STAT bit pair is not affected by the transaction termination and its value remains '11 (Valid). However, as the token packet of a new transaction is received, the actual endpoint status is masked as '10 (NAK) when a buffer conflict between the USB peripheral and the application software is detected (this condition is identified by DTOG and SW\_BUF having the same value, see [Table 123 on page 879](#)). The application software responds to the CTR event notification by clearing the interrupt flag and starting any required handling of the completed transaction. When the application packet buffer usage is over, the software toggles the SW\_BUF bit, writing '1 to it, to notify the USB peripheral about the availability of that buffer. In this way, the number of NAKed transactions is limited only by the application elaboration time of a transaction data: if the elaboration time is shorter than the time required to complete a transaction on the USB bus, no re-transmissions due to flow control will take place and the actual transfer rate is limited only by the host PC.

The application software can always override the special flow control implemented for double-buffered bulk endpoints, writing an explicit status different from '11 (Valid) into the STAT bit pair of the related USB\_EPnR register. In this case, the USB peripheral will always use the programmed endpoint status, regardless of the buffer usage condition.

### 30.5.4 Isochronous transfers

The USB standard supports full speed peripherals requiring a fixed and accurate data production/consume frequency, defining this kind of traffic as 'Isochronous'. Typical examples of this data are: audio samples, compressed video streams, and in general any sort of sampled data having strict requirements for the accuracy of delivered frequency. When an endpoint is defined to be 'isochronous' during the enumeration phase, the host allocates in the frame the required bandwidth and delivers exactly one IN or OUT packet each frame, depending on endpoint direction. To limit the bandwidth requirements, no re-transmission of failed transactions is possible for Isochronous traffic; this leads to the fact that an isochronous transaction does not have a handshake phase and no ACK packet is expected or sent after the data packet. For the same reason, Isochronous transfers do not support data toggle sequencing and always use DATA0 PID to start any data packet.

The Isochronous behavior for an endpoint is selected by setting the EP\_TYPE bits at '10 in its USB\_EPnR register; since there is no handshake phase the only legal values for the STAT\_RX/STAT\_TX bit pairs are '00 (Disabled) and '11 (Valid), any other value will produce results not compliant to USB standard. Isochronous endpoints implement double-buffering to ease application software development, using both 'transmission' and 'reception' packet memory areas to manage buffer swapping on each successful transaction in order to have always a complete buffer to be used by the application, while the USB peripheral fills the other.

The memory buffer which is currently used by the USB peripheral is defined by the DTOG bit related to the endpoint direction (DTOG\_RX for 'reception' isochronous endpoints, DTOG\_TX for 'transmission' isochronous endpoints, both in the related USB\_EPnR register) according to [Table 124](#).

**Table 124. Isochronous memory buffers usage**

<b>Endpoint Type</b>	<b>DTOG bit value</b>	<b>Packet buffer used by the USB peripheral</b>	<b>Packet buffer used by the application software</b>
IN	0	ADDRn_TX_0 / COUNTn_TX_0 buffer description table locations.	ADDRn_TX_1 / COUNTn_TX_1 buffer description table locations.
	1	ADDRn_TX_1 / COUNTn_TX_1 buffer description table locations.	ADDRn_TX_0 / COUNTn_TX_0 buffer description table locations.
OUT	0	ADDRn_RX_0 / COUNTn_RX_0 buffer description table locations.	ADDRn_RX_1 / COUNTn_RX_1 buffer description table locations.
	1	ADDRn_RX_1 / COUNTn_RX_1 buffer description table locations.	ADDRn_RX_0 / COUNTn_RX_0 buffer description table locations.

As it happens with double-buffered bulk endpoints, the USB\_EPnR registers used to implement Isochronous endpoints are forced to be used as unidirectional ones. In case it is required to have Isochronous endpoints enabled both for reception and transmission, two USB\_EPnR registers must be used.

The application software is responsible for the DTOG bit initialization according to the first buffer to be used; this has to be done considering the special toggle-only property that these two bits have. At the end of each transaction, the CTR\_RX or CTR\_TX bit of the addressed endpoint USB\_EPnR register is set, depending on the enabled direction. At the same time, the affected DTOG bit in the USB\_EPnR register is hardware toggled making buffer swapping completely software independent. STAT bit pair is not affected by transaction completion; since no flow control is possible for Isochronous transfers due to the lack of handshake phase, the endpoint remains always '11 (Valid). CRC errors or buffer-overrun conditions occurring during Isochronous OUT transfers are anyway considered as correct transactions and they always trigger an CTR\_RX event. However, CRC errors will anyway set the ERR bit in the USB\_ISTR register to notify the software of the possible data corruption.

### 30.5.5 Suspend/Resume events

The USB standard defines a special peripheral state, called SUSPEND, in which the average current drawn from the USB bus must not be greater than 2.5 mA. This requirement is of fundamental importance for bus-powered devices, while self-powered devices are not required to comply to this strict power consumption constraint. In suspend mode, the host PC sends the notification by not sending any traffic on the USB bus for more than 3 ms: since a SOF packet must be sent every 1 ms during normal operations, the USB peripheral detects the lack of 3 consecutive SOF packets as a suspend request from the host PC and set the SUSP bit to '1' in USB\_ISTR register, causing an interrupt if enabled. Once the device is suspended, its normal operation can be restored by a so called RESUME sequence, which can be started from the host PC or directly from the peripheral itself, but it is always terminated by the host PC. The suspended USB peripheral must be anyway able to detect a RESET sequence, reacting to this event as a normal USB reset event.

The actual procedure used to suspend the USB peripheral is device dependent since according to the device composition, different actions may be required to reduce the total consumption.

A brief description of a typical suspend procedure is provided below, focused on the USB-related aspects of the application software routine responding to the SUSP notification of the USB peripheral:

1. Set the FSUSP bit in the USB\_CNTR register to 1. This action activates the suspend mode within the USB peripheral. As soon as the suspend mode is activated, the check on SOF reception is disabled to avoid any further SUSP interrupts being issued while the USB is suspended.
2. Remove or reduce any static power consumption in blocks different from the USB peripheral.
3. Set LP\_MODE bit in USB\_CNTR register to 1 to remove static power consumption in the analog USB transceivers but keeping them able to detect resume activity.
4. Optionally turn off external oscillator and device PLL to stop any activity inside the device.

When an USB event occurs while the device is in SUSPEND mode, the RESUME procedure must be invoked to restore nominal clocks and regain normal USB behavior. Particular care must be taken to insure that this process does not take more than 10 ms when the wakening event is an USB reset sequence (See “Universal Serial Bus Specification” for more details). The start of a resume or reset sequence, while the USB peripheral is suspended, clears the LP\_MODE bit in USB\_CNTR register asynchronously. Even if this event can trigger an WKUP interrupt if enabled, the use of an interrupt response routine must be carefully evaluated because of the long latency due to system clock restart; to have the shorter latency before re-activating the nominal clock it is suggested to put the resume procedure just after the end of the suspend one, so its code is immediately executed as soon as the system clock restarts. To prevent ESD discharges or any other kind of noise from waking-up the system (the exit from suspend mode is an asynchronous event), a suitable analog filter on data line status is activated during suspend; the filter width is about 70 ns.

The following is a list of actions a resume procedure should address:

1. Optionally turn on external oscillator and/or device PLL.
2. Clear FSUSP bit of USB\_CNTR register.
3. If the resume triggering event has to be identified, bits RXDP and RXDM in the USB\_FNR register can be used according to [Table 125](#), which also lists the intended software action in all the cases. If required, the end of resume or reset sequence can be detected monitoring the status of the above mentioned bits by checking when they reach the “10” configuration, which represent the Idle bus state; moreover at the end of a reset sequence the RESET bit in USB\_ISTR register is set to 1, issuing an interrupt if enabled, which should be handled as usual.

**Table 125. Resume event detection**

[RXDP,RXDM] status	Wake-up event	Required resume software action
“00”	Root reset	None
“10”	None (noise on bus)	Go back in Suspend mode

**Table 125. Resume event detection (continued)**

[RXDP,RXDM] status	Wake-up event	Required resume software action
"01"	Root resume	None
"11"	Not allowed (noise on bus)	Go back in Suspend mode

A device may require to exit from suspend mode as an answer to particular events not directly related to the USB protocol (e.g. a mouse movement wakes up the whole system). In this case, the resume sequence can be started by setting the RESUME bit in the USB\_CNTR register to '1 and resetting it to 0 after an interval between 1 ms and 15 ms (this interval can be timed using ESOF interrupts, occurring with a 1 ms period when the system clock is running at nominal frequency). Once the RESUME bit is clear, the resume sequence is completed by the host PC and its end can be monitored again using the RXDP and RXDM bits in the USB\_FNR register.

*Note:* *The RESUME bit must be anyway used only after the USB peripheral has been put in suspend mode, setting the FSUSP bit in USB\_CNTR register to 1.*

## 30.6 USB and USB SRAM registers

The USB peripheral registers can be divided into the following groups:

- Common Registers: Interrupt and Control registers
- Endpoint Registers: Endpoint configuration and status

The USB SRAM registers cover:

- Buffer Descriptor Table: Location of packet memory used to locate data buffers (see [Section 2.2: Memory organization](#) to find USB SRAM base address).

All register addresses are expressed as offsets with respect to the USB peripheral registers base address, except the buffer descriptor table locations, which starts at the USB SRAM base address offset by the value specified in the USB\_BTABLE register.

Refer to [Section 1.2 on page 42](#) for a list of abbreviations used in register descriptions.

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

### 30.6.1 Common registers

These registers affect the general behavior of the USB peripheral defining operating mode, interrupt handling, device address and giving access to the current frame number updated by the host PC.

#### USB control register (USB\_CNTR)

Address offset: 0x40

Reset value: 0x0003

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTR M	PMAOVR M	ERR M	WKUP M	SUSP M	RESET M	SOF M	ESOF M	L1REQ M	Res	L1RESU ME	RE SUME	F SUSP	LP MODE	PDW N	F RES
rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw

Bit 15 **CTRM**: Correct transfer interrupt mask

0: Correct Transfer (CTR) Interrupt disabled.

1: CTR Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB\_ISTR register is set.

Bit 14 **PMAOVRM**: Packet memory area over / underrun interrupt mask

0: PMAOVR Interrupt disabled.

1: PMAOVR Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB\_ISTR register is set.

Bit 13 **ERRM**: Error interrupt mask

0: ERR Interrupt disabled.

1: ERR Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB\_ISTR register is set.

Bit 12 **WKUPM**: Wake-up interrupt mask

0: WKUP Interrupt disabled.

1: WKUP Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB\_ISTR register is set.

- Bit 11 **SUSPM:** Suspend mode interrupt mask  
 0: Suspend Mode Request (SUSP) Interrupt disabled.  
 1: SUSP Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB\_ISTR register is set.
- Bit 10 **RESETM:** USB reset interrupt mask  
 0: RESET Interrupt disabled.  
 1: RESET Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB\_ISTR register is set.
- Bit 9 **SOFM:** Start of frame interrupt mask  
 0: SOF Interrupt disabled.  
 1: SOF Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB\_ISTR register is set.
- Bit 8 **ESOFM:** Expected start of frame interrupt mask  
 0: Expected Start of Frame (ESOF) Interrupt disabled.  
 1: ESOF Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB\_ISTR register is set.
- Bit 7 **L1REQM:** LPM L1 state request interrupt mask  
 0: LPM L1 state request (L1REQ) Interrupt disabled.  
 1: L1REQ Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB\_ISTR register is set.
- Bit 6 Reserved, must be kept at reset value.
- Bit 5 **L1RESUME:** LPM L1 Resume request  
 The microcontroller can set this bit to send a LPM L1 Resume signal to the host. After the signaling ends, this bit is cleared by hardware.
- Bit 4 **RESUME:** Resume request  
 The microcontroller can set this bit to send a Resume signal to the host. It must be activated, according to USB specifications, for no less than 1 ms and no more than 15 ms after which the Host PC is ready to drive the resume sequence up to its end.
- Bit 3 **FSUSP:** Force suspend  
 Software must set this bit when the SUSP interrupt is received, which is issued when no traffic is received by the USB peripheral for 3 ms.  
 0: No effect.  
 1: Enter suspend mode. Clocks and static power dissipation in the analog transceiver are left unaffected. If suspend power consumption is a requirement (bus-powered device), the application software should set the LP\_MODE bit after FSUSP as explained below.
- Bit 2 **LP\_MODE:** Low-power mode  
 This mode is used when the suspend-mode power constraints require that all static power dissipation is avoided, except the one required to supply the external pull-up resistor. This condition should be entered when the application is ready to stop all system clocks, or reduce their frequency in order to meet the power consumption requirements of the USB suspend condition. The USB activity during the suspend mode (WKUP event) asynchronously resets this bit (it can also be reset by software).  
 0: No Low-power mode.  
 1: Enter Low-power mode.

**Bit 1 PDWN:** Power down

This bit is used to completely switch off all USB-related analog parts if it is required to completely disable the USB peripheral for any reason. When this bit is set, the USB peripheral is disconnected from the transceivers and it cannot be used.

- 0: Exit Power Down.
- 1: Enter Power down mode.

**Bit 0 FRES:** Force USB Reset

- 0: Clear USB reset.

1: Force a reset of the USB peripheral, exactly like a RESET signaling on the USB. The USB peripheral is held in RESET state until software clears this bit. A “USB-RESET” interrupt is generated, if enabled.

**USB interrupt status register (USB\_ISTR)**

Address offset: 0x44

Reset value: 0x0000 0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTR	PMA OVR	ERR	WKUP	SUSP	RESET	SOF	ESOF	L1REQ	Res.	Res.	DIR	EP_ID[3:0]			
r	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0			r	r	r	r	r

This register contains the status of all the interrupt sources allowing application software to determine, which events caused an interrupt request.

The upper part of this register contains single bits, each of them representing a specific event. These bits are set by the hardware when the related event occurs; if the corresponding bit in the USB\_CNTR register is set, a generic interrupt request is generated. The interrupt routine, examining each bit, will perform all necessary actions, and finally it will clear the serviced bits. If any of them is not cleared, the interrupt is considered to be still pending, and the interrupt line is kept high again. If several bits are set simultaneously, only a single interrupt is generated.

Endpoint transaction completion can be handled in a different way to reduce interrupt response latency. The CTR bit is set by the hardware as soon as an endpoint successfully completes a transaction, generating a generic interrupt request if the corresponding bit in USB\_CNTR is set. An endpoint dedicated interrupt condition is activated independently from the CTRM bit in the USB\_CNTR register. Both interrupt conditions remain active until software clears the pending bit in the corresponding USB\_EPnR register (the CTR bit is actually a read only bit). For endpoint-related interrupts, the software can use the Direction of Transaction (DIR) and EP\_ID read-only bits to identify, which endpoint made the last interrupt request and called the corresponding interrupt service routine.

The user can choose the relative priority of simultaneously pending USB\_ISTR events by specifying the order in which software checks USB\_ISTR bits in an interrupt service routine. Only the bits related to events, which are serviced, are cleared. At the end of the service routine, another interrupt is requested, to service the remaining conditions.

To avoid spurious clearing of some bits, it is recommended to clear them with a load instruction where all bits which must not be altered are written with 1, and all bits to be cleared are written with '0 (these bits can only be cleared by software). Read-modify-write cycles should be avoided because between the read and the write operations another bit

could be set by the hardware and the next write will clear it before the microprocessor has the time to serve the event.

The following describes each bit in detail:

Bit 15 **CTR:** Correct transfer

This bit is set by the hardware to indicate that an endpoint has successfully completed a transaction; using DIR and EP\_ID bits software can determine which endpoint requested the interrupt. This bit is read-only.

Bit 14 **PMAOVR:** Packet memory area over / underrun

This bit is set if the microcontroller has not been able to respond in time to an USB memory request. The USB peripheral handles this event in the following way: During reception an ACK handshake packet is not sent, during transmission a bit-stuff error is forced on the transmitted stream; in both cases the host will retry the transaction. The PMAOVR interrupt should never occur during normal operations. Since the failed transaction is retried by the host, the application software has the chance to speed-up device operations during this interrupt handling, to be ready for the next transaction retry; however this does not happen during Isochronous transfers (no Isochronous transaction is anyway retried) leading to a loss of data in this case. This bit is read/write but only '0 can be written and writing '1 has no effect.

Bit 13 **ERR:** Error

This flag is set whenever one of the errors listed below has occurred:

NANS: No ANSwer. The timeout for a host response has expired.

CRC: Cyclic Redundancy Check error. One of the received CRCs, either in the token or in the data, was wrong.

BST: Bit Stuffing error. A bit stuffing error was detected anywhere in the PID, data, and/or CRC.

FVIO: Framing format Violation. A non-standard frame was received (EOP not in the right place, wrong token sequence, etc.).

The USB software can usually ignore errors, since the USB peripheral and the PC host manage retransmission in case of errors in a fully transparent way. This interrupt can be useful during the software development phase, or to monitor the quality of transmission over the USB bus, to flag possible problems to the user (e.g. loose connector, too noisy environment, broken conductor in the USB cable and so on). This bit is read/write but only '0 can be written and writing '1 has no effect.

Bit 12 **WKUP:** Wake-up

This bit is set to 1 by the hardware when, during suspend mode, activity is detected that wakes up the USB peripheral. This event asynchronously clears the LP\_MODE bit in the CTLR register and activates the USB\_WAKEUP line, which can be used to notify the rest of the device (e.g. wake-up unit) about the start of the resume process. This bit is read/write but only '0 can be written and writing '1 has no effect.

Bit 11 **SUSP:** Suspend mode request

This bit is set by the hardware when no traffic has been received for 3 ms, indicating a suspend mode request from the USB bus. The suspend condition check is enabled immediately after any USB reset and it is disabled by the hardware when the suspend mode is active (FSUSP=1) until the end of resume sequence. This bit is read/write but only '0 can be written and writing '1 has no effect.

**Bit 10 RESET:** USB reset request

Set when the USB peripheral detects an active USB RESET signal at its inputs. The USB peripheral, in response to a RESET, just resets its internal protocol state machine, generating an interrupt if RESETM enable bit in the USB\_CNTR register is set. Reception and transmission are disabled until the RESET bit is cleared. All configuration registers do not reset: the microcontroller must explicitly clear these registers (this is to ensure that the RESET interrupt can be safely delivered, and any transaction immediately followed by a RESET can be completed). The function address and endpoint registers are reset by an USB reset event.

This bit is read/write but only '0 can be written and writing '1 has no effect.

**Bit 9 SOF:** Start of frame

This bit signals the beginning of a new USB frame and it is set when a SOF packet arrives through the USB bus. The interrupt service routine may monitor the SOF events to have a 1 ms synchronization event to the USB host and to safely read the USB\_FNR register which is updated at the SOF packet reception (this could be useful for isochronous applications). This bit is read/write but only '0 can be written and writing '1 has no effect.

**Bit 8 ESOF:** Expected start of frame

This bit is set by the hardware when an SOF packet is expected but not received. The host sends an SOF packet each 1 ms, but if the device does not receive it properly, the Suspend Timer issues this interrupt. If three consecutive ESOF interrupts are generated (i.e. three SOF packets are lost) without any traffic occurring in between, a SUSP interrupt is generated. This bit is set even when the missing SOF packets occur while the Suspend Timer is not yet locked. This bit is read/write but only '0 can be written and writing '1 has no effect.

**Bit 7 L1REQ:** LPM L1 state request

This bit is set by the hardware when LPM command to enter the L1 state is successfully received and acknowledged. This bit is read/write but only '0 can be written and writing '1 has no effect.

Bits 6:5 Reserved, must be kept at reset value.

**Bit 4 DIR:** Direction of transaction

This bit is written by the hardware according to the direction of the successful transaction, which generated the interrupt request.

If DIR bit=0, CTR\_TX bit is set in the USB\_EPnR register related to the interrupting endpoint. The interrupting transaction is of IN type (data transmitted by the USB peripheral to the host PC).

If DIR bit=1, CTR\_RX bit or both CTR\_TX/CTR\_RX are set in the USB\_EPnR register related to the interrupting endpoint. The interrupting transaction is of OUT type (data received by the USB peripheral from the host PC) or two pending transactions are waiting to be processed.

This information can be used by the application software to access the USB\_EPnR bits related to the triggering transaction since it represents the direction having the interrupt pending. This bit is read-only.

### Bits 3:0 EP\_ID[3:0]: Endpoint Identifier

These bits are written by the hardware according to the endpoint number, which generated the interrupt request. If several endpoint transactions are pending, the hardware writes the endpoint identifier related to the endpoint having the highest priority defined in the following way: Two endpoint sets are defined, in order of priority: Isochronous and double-buffered bulk endpoints are considered first and then the other endpoints are examined. If more than one endpoint from the same set is requesting an interrupt, the EP\_ID bits in USB\_ISTR register are assigned according to the lowest requesting endpoint register, EP0R having the highest priority followed by EP1R and so on. The application software can assign a register to each endpoint according to this priority scheme, so as to order the concurring endpoint requests in a suitable way. These bits are read only.

## USB frame number register (USB\_FNR)

Address offset: 0x48

Reset value: 0x0XXX where X is undefined

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
	RXDP	RXDM	LCK	LSOF[1:0]		FN[10:0]												
	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

### Bit 15 RXDP: Receive data + line status

This bit can be used to observe the status of received data plus upstream port data line. It can be used during end-of-suspend routines to help determining the wake-up event.

### Bit 14 RXDM: Receive data - line status

This bit can be used to observe the status of received data minus upstream port data line. It can be used during end-of-suspend routines to help determining the wake-up event.

### Bit 13 LCK: Locked

This bit is set by the hardware when at least two consecutive SOF packets have been received after the end of an USB reset condition or after the end of an USB resume sequence. Once locked, the frame timer remains in this state until an USB reset or USB suspend event occurs.

### Bits 12:11 LSOF[1:0]: Lost SOF

These bits are written by the hardware when an ESOF interrupt is generated, counting the number of consecutive SOF packets lost. At the reception of an SOF packet, these bits are cleared.

### Bits 10:0 FN[10:0]: Frame number

This bit field contains the 11-bits frame number contained in the last received SOF packet. The frame number is incremented for every frame sent by the host and it is useful for Isochronous transfers. This bit field is updated on the generation of an SOF interrupt.

## USB device address (USB\_DADDR)

Address offset: 0x4C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	EF	ADD6	ADD5	ADD4	ADD3	ADD2	ADD1	ADD0							
								rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:8 Reserved

#### Bit 7 **EF:** Enable function

This bit is set by the software to enable the USB device. The address of this device is contained in the following ADD[6:0] bits. If this bit is at '0' no transactions are handled, irrespective of the settings of USB\_EPnR registers.

#### Bits 6:0 **ADD[6:0]:** Device address

These bits contain the USB function address assigned by the host PC during the enumeration process. Both this field and the Endpoint Address (EA) field in the associated USB\_EPnR register must match with the information contained in a USB token in order to handle a transaction to the required endpoint.

### Buffer table address (USB\_BTABLE)

Address offset: 0x50

Reset value: 0x0000

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

Bits 15:3 **BTABLE[15:3]:** Buffer table

These bits contain the start address of the buffer allocation table inside the dedicated packet memory. This table describes each endpoint buffer location and size and it must be aligned to an 8 byte boundary (the 3 least significant bits are always '0'). At the beginning of every transaction addressed to this device, the USB peripheral reads the element of this table related to the addressed endpoint, to get its buffer start location and the buffer size (Refer to [Structure and usage of packet buffers on page 873](#)).

Bits 2:0 Reserved, forced by hardware to 0.

### LPM control and status register (USB\_LPMCSR)

Address offset: 0x54

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	BESL[3:0]				REM WAKE	Res.	LPM ACK	LPM EN							
								r	r	r	r	r		rw	rw

Bits 15:8 Reserved, must be kept at reset value.

**Bits 7:4 BESL[3:0]: BESL value**

These bits contain the BESL value received with last ACKed LPM Token

**Bit 3 REMWAKE: bRemoteWake value**

This bit contains the bRemoteWake value received with last ACKed LPM Token

**Bit 2 Reserved**

**Bit 1 LPMACK: LPM Token acknowledge enable**

0: the valid LPM Token is NYET.

1: the valid LPM Token is ACK.

The NYET/ACK is returned only on a successful LPM transaction:

No errors in both the EXT token and the LPM token (else ERROR)

A valid bLinkState = 0001B (L1) is received (else STALL)

**Bit 0 LPMEN: LPM support enable**

This bit is set by the software to enable the LPM support within the USB device. If this bit is at '0' no LPM transactions are handled.

## Battery charging detector (USB\_BCDR)

Address offset: 0x58

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DPPU	Res.	PS2 DET	SDET	PDET	DC DET	SDEN	PDEN	DCD EN	BCD EN						
rw								r	r	r	r	rw	rw	rw	rw

**Bit 15 DPPU: DP pull-up control**

This bit is set by software to enable the embedded pull-up on the DP line. Clearing it to '0' can be used to signalize disconnect to the host when needed by the user software.

Bits 14:8 Reserved, must be kept at reset value.

**Bit 7 PS2DET: DM pull-up detection status**

This bit is active only during PD and gives the result of comparison between DM voltage level and  $V_{LGC}$  threshold. In normal situation, the DM level should be below this threshold. If it is above, it means that the DM is externally pulled high. This can be caused by connection to a PS2 port (which pulls-up both DP and DM lines) or to some proprietary charger not following the BCD specification.

0: Normal port detected (connected to SDP, ACA, CDP or DCP).

1: PS2 port or proprietary charger detected.

**Bit 6 SDET: Secondary detection (SD) status**

This bit gives the result of SD.

0: CDP detected.

1: DCP detected.

**Bit 5 PDET: Primary detection (PD) status**

This bit gives the result of PD.

0: no BCD support detected (connected to SDP or proprietary device).

1: BCD support detected (connected to ACA, CDP or DCP).

Bit 4 **DCDET**: Data contact detection (DCD) status

This bit gives the result of DCD.

0: data lines contact not detected.

1: data lines contact detected.

Bit 3 **SDEN**: Secondary detection (SD) mode enable

This bit is set by the software to put the BCD into SD mode. Only one detection mode (DCD, PD, SD or OFF) should be selected to work correctly.

Bit 2 **PDEN**: Primary detection (PD) mode enable

This bit is set by the software to put the BCD into PD mode. Only one detection mode (DCD, PD, SD or OFF) should be selected to work correctly.

Bit 1 **DCDEN**: Data contact detection (DCD) mode enable

This bit is set by the software to put the BCD into DCD mode. Only one detection mode (DCD, PD, SD or OFF) should be selected to work correctly.

Bit 0 **BCDEN**: Battery charging detector (BCD) enable

This bit is set by the software to enable the BCD support within the USB device. When enabled, the USB PHY is fully controlled by BCD and cannot be used for normal communication. Once the BCD discovery is finished, the BCD should be placed in OFF mode by clearing this bit to '0' in order to allow the normal USB operation.

## Endpoint-specific registers

The number of these registers varies according to the number of endpoints that the USB peripheral is designed to handle. The USB peripheral supports up to 8 bidirectional endpoints. Each USB device must support a control endpoint whose address (EA bits) must be set to 0. The USB peripheral behaves in an undefined way if multiple endpoints are enabled having the same endpoint number value. For each endpoint, an USB\_EPnR register is available to store the endpoint specific information.

### USB endpoint n register (USB\_EPnR), n=[0..7]

Address offset: 0x00 to 0x1C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTR_RX	DTOG_RX	STAT_RX[1:0]		SETUP	EP_TYPE[1:0]		EP_KIND	CTR_TX	DTOG_TX	STAT_TX[1:0]		EA[3:0]			
rc_w0	t	t	t	r	rw	rw	rw	rc_w0	t	t	t	rw	rw	rw	rw

They are also reset when an USB reset is received from the USB bus or forced through bit FRES in the CTR register, except the CTR\_RX and CTR\_TX bits, which are kept unchanged to avoid missing a correct packet notification immediately followed by an USB reset event. Each endpoint has its USB\_EPnR register where *n* is the endpoint identifier.

Read-modify-write cycles on these registers should be avoided because between the read and the write operations some bits could be set by the hardware and the next write would modify them before the CPU has the time to detect the change. For this purpose, all bits affected by this problem have an 'invariant' value that must be used whenever their modification is not required. It is recommended to modify these registers with a load instruction where all the bits, which can be modified only by the hardware, are written with their 'invariant' value.

**Bit 15 CTR\_RX:** Correct transfer for reception

This bit is set by the hardware when an OUT/SETUP transaction is successfully completed on this endpoint; the software can only clear this bit. If the CTRM bit in USB\_CNTR register is set accordingly, a generic interrupt condition is generated together with the endpoint related interrupt condition, which is always activated. The type of occurred transaction, OUT or SETUP, can be determined from the SETUP bit described below.

A transaction ended with a NAK or STALL handshake does not set this bit, since no data is actually transferred, as in the case of protocol errors or data toggle mismatches.

This bit is read/write but only '0' can be written, writing '1' has no effect.

**Bit 14 DTOG\_RX:** Data toggle, for reception transfers

If the endpoint is not Isochronous, this bit contains the expected value of the data toggle bit (0=DATA0, 1=DATA1) for the next data packet to be received. Hardware toggles this bit, when the ACK handshake is sent to the USB host, following a data packet reception having a matching data PID value; if the endpoint is defined as a control one, hardware clears this bit at the reception of a SETUP PID addressed to this endpoint.

If the endpoint is using the double-buffering feature this bit is used to support packet buffer swapping too (Refer to [Section 30.5.3: Double-buffered endpoints](#)).

If the endpoint is Isochronous, this bit is used only to support packet buffer swapping since no data toggling is used for this sort of endpoints and only DATA0 packet are transmitted (Refer to [Section 30.5.4: Isochronous transfers](#)). Hardware toggles this bit just after the end of data packet reception, since no handshake is used for isochronous transfers.

This bit can also be toggled by the software to initialize its value (mandatory when the endpoint is not a control one) or to force specific data toggle/packet buffer usage. When the application software writes '0', the value of DTOG\_RX remains unchanged, while writing '1' makes the bit value toggle. This bit is read/write but it can be only toggled by writing 1.

**Bits 13:12 STAT\_RX [1:0]:** Status bits, for reception transfers

These bits contain information about the endpoint status, which are listed in [Table 126: Reception status encoding on page 895](#). These bits can be toggled by software to initialize their value. When the application software writes '0', the value remains unchanged, while writing '1' makes the bit value toggle. Hardware sets the STAT\_RX bits to NAK when a correct transfer has occurred (CTR\_RX=1) corresponding to a OUT or SETUP (control only) transaction addressed to this endpoint, so the software has the time to elaborate the received data before it acknowledge a new transaction

Double-buffered bulk endpoints implement a special transaction flow control, which control the status based upon buffer availability condition (Refer to [Section 30.5.3: Double-buffered endpoints](#)).

If the endpoint is defined as Isochronous, its status can be only "VALID" or "DISABLED", so that the hardware cannot change the status of the endpoint after a successful transaction. If the software sets the STAT\_RX bits to 'STALL' or 'NAK' for an Isochronous endpoint, the USB peripheral behavior is not defined. These bits are read/write but they can be only toggled by writing '1'.

**Bit 11 SETUP:** Setup transaction completed

This bit is read-only and it is set by the hardware when the last completed transaction is a SETUP. This bit changes its value only for control endpoints. It must be examined, in the case of a successful receive transaction (CTR\_RX event), to determine the type of transaction occurred. To protect the interrupt service routine from the changes in SETUP bits due to next incoming tokens, this bit is kept frozen while CTR\_RX bit is at 1; its state changes when CTR\_RX is at 0. This bit is read-only.

Bits 10:9 **EP\_TYPE[1:0]**: Endpoint type

These bits configure the behavior of this endpoint as described in [Table 127: Endpoint type encoding on page 895](#). Endpoint 0 must always be a control endpoint and each USB function must have at least one control endpoint which has address 0, but there may be other control endpoints if required. Only control endpoints handle SETUP transactions, which are ignored by endpoints of other kinds. SETUP transactions cannot be answered with NAK or STALL. If a control endpoint is defined as NAK, the USB peripheral will not answer, simulating a receive error, in the receive direction when a SETUP transaction is received. If the control endpoint is defined as STALL in the receive direction, then the SETUP packet is accepted anyway, transferring data and issuing the CTR interrupt. The reception of OUT transactions is handled in the normal way, even if the endpoint is a control one.

Bulk and interrupt endpoints have very similar behavior and they differ only in the special feature available using the EP\_KIND configuration bit.

The usage of Isochronous endpoints is explained in [Section 30.5.4: Isochronous transfers](#)

Bit 8 **EP\_KIND**: Endpoint kind

The meaning of this bit depends on the endpoint type configured by the EP\_TYPE bits. [Table 128](#) summarizes the different meanings.

**DBL\_BUF**: This bit is set by the software to enable the double-buffering feature for this bulk endpoint. The usage of double-buffered bulk endpoints is explained in [Section 30.5.3: Double-buffered endpoints](#).

**STATUS\_OUT**: This bit is set by the software to indicate that a status out transaction is expected: in this case all OUT transactions containing more than zero data bytes are answered 'STALL' instead of 'ACK'. This bit may be used to improve the robustness of the application to protocol errors during control transfers and its usage is intended for control endpoints only. When STATUS\_OUT is reset, OUT transactions can have any number of bytes, as required.

Bit 7 **CTR\_TX**: Correct Transfer for transmission

This bit is set by the hardware when an IN transaction is successfully completed on this endpoint; the software can only clear this bit. If the CTRM bit in the USB\_CNTR register is set accordingly, a generic interrupt condition is generated together with the endpoint related interrupt condition, which is always activated.

A transaction ended with a NAK or STALL handshake does not set this bit, since no data is actually transferred, as in the case of protocol errors or data toggle mismatches.

This bit is read/write but only '0' can be written.

Bit 6 **DTOG\_TX**: Data Toggle, for transmission transfers

If the endpoint is non-isochronous, this bit contains the required value of the data toggle bit (0=DATA0, 1=DATA1) for the next data packet to be transmitted. Hardware toggles this bit when the ACK handshake is received from the USB host, following a data packet transmission. If the endpoint is defined as a control one, hardware sets this bit to 1 at the reception of a SETUP PID addressed to this endpoint.

If the endpoint is using the double buffer feature, this bit is used to support packet buffer swapping too (Refer to [Section 30.5.3: Double-buffered endpoints](#))

If the endpoint is Isochronous, this bit is used to support packet buffer swapping since no data toggling is used for this sort of endpoints and only DATA0 packet are transmitted (Refer to [Section 30.5.4: Isochronous transfers](#)). Hardware toggles this bit just after the end of data packet transmission, since no handshake is used for Isochronous transfers.

This bit can also be toggled by the software to initialize its value (mandatory when the endpoint is not a control one) or to force a specific data toggle/packet buffer usage. When the application software writes '0, the value of DTOG\_TX remains unchanged, while writing '1 makes the bit value toggle. This bit is read/write but it can only be toggled by writing 1.

Bits 5:4 **STAT\_TX [1:0]**: Status bits, for transmission transfers

These bits contain the information about the endpoint status, listed in [Table 129](#). These bits can be toggled by the software to initialize their value. When the application software writes '0, the value remains unchanged, while writing '1 makes the bit value toggle. Hardware sets the STAT\_TX bits to NAK, when a correct transfer has occurred (CTR\_TX=1) corresponding to a IN or SETUP (control only) transaction addressed to this endpoint. It then waits for the software to prepare the next set of data to be transmitted.

Double-buffered bulk endpoints implement a special transaction flow control, which controls the status based on buffer availability condition (Refer to [Section 30.5.3: Double-buffered endpoints](#)).

If the endpoint is defined as Isochronous, its status can only be "VALID" or "DISABLED". Therefore, the hardware cannot change the status of the endpoint after a successful transaction. If the software sets the STAT\_TX bits to 'STALL' or 'NAK' for an Isochronous endpoint, the USB peripheral behavior is not defined. These bits are read/write but they can be only toggled by writing '1.

Bits 3:0 **EA[3:0]**: Endpoint address

Software must write in this field the 4-bit address used to identify the transactions directed to this endpoint. A value must be written before enabling the corresponding endpoint.

**Table 126. Reception status encoding**

STAT_RX[1:0]	Meaning
00	<b>DISABLED</b> : all reception requests addressed to this endpoint are ignored.
01	<b>STALL</b> : the endpoint is stalled and all reception requests result in a STALL handshake.
10	<b>NAK</b> : the endpoint is naked and all reception requests result in a NAK handshake.
11	<b>VALID</b> : this endpoint is enabled for reception.

**Table 127. Endpoint type encoding**

EP_TYPE[1:0]	Meaning
00	BULK
01	CONTROL
10	ISO
11	INTERRUPT

**Table 128. Endpoint kind meaning**

EP_TYPE[1:0]		EP_KIND meaning
00	BULK	DBL_BUF
01	CONTROL	STATUS_OUT
10	ISO	Not used
11	INTERRUPT	Not used

**Table 129. Transmission status encoding**

STAT_TX[1:0]	Meaning
00	<b>DISABLED</b> : all transmission requests addressed to this endpoint are ignored.
01	<b>STALL</b> : the endpoint is stalled and all transmission requests result in a STALL handshake.
10	<b>NAK</b> : the endpoint is naked and all transmission requests result in a NAK handshake.
11	<b>VALID</b> : this endpoint is enabled for transmission.

### 30.6.2 Buffer descriptor table

**Note:** *The buffer descriptor table is located inside the packet buffer memory in the separate "USB SRAM" address space.*

Although the buffer descriptor table is located inside the packet buffer memory ("USB SRAM" area), its entries can be considered as additional registers used to configure the location and size of the packet buffers used to exchange data between the USB macro cell and the device.

The first packet memory location is located at USB SRAM base address. The buffer descriptor table entry associated with the USB\_EPnR registers is described below. The packet memory should be accessed only by byte (8-bit) or half-word (16-bit) accesses. Word (32-bit) accesses are not allowed.

A thorough explanation of packet buffers and the buffer descriptor table usage can be found in [Structure and usage of packet buffers on page 873](#).

#### Transmission buffer address n (USB\_ADDRn\_TX)

Address offset: [USB\_BTABLE] + n\*8

**Note:** *In case of double-buffered or isochronous endpoints in the IN direction, this address location is referred to as USB\_ADDRn\_TX\_0.*

*In case of double-buffered or isochronous endpoints in the OUT direction, this address location is used for USB\_ADDRn\_RX\_0.*

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

##### Bits 15:1 ADDRn\_TX[15:1]: Transmission buffer address

These bits point to the starting address of the packet buffer containing data to be transmitted by the endpoint associated with the USB\_EPnR register at the next IN token addressed to it.

Bit 0 Must always be written as '0' since packet memory is half-word wide and all packet buffers must be half-word aligned.

#### Transmission byte count n (USB\_COUNTn\_TX)

Address offset: [USB\_BTABLE] + n\*8 + 2

**Note:** *In case of double-buffered or isochronous endpoints in the IN direction, this address location is referred to as USB\_COUNTn\_TX\_0.*

*In case of double-buffered or isochronous endpoints in the OUT direction, this address location is used for USB\_COUNTn\_RX\_0.*

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

Bits 15:10 These bits are not used since packet size is limited by USB specifications to 1023 bytes. Their value is not considered by the USB peripheral.

Bits 9:0 **COUNTn\_RX[9:0]**: Transmission byte count

These bits contain the number of bytes to be transmitted by the endpoint associated with the USB\_EPnR register at the next IN token addressed to it.

### Reception buffer address n (USB\_ADDRn\_RX)

Address offset: [USB\_BTABLE] + n\*8 + 4

**Note:**

*In case of double-buffered or isochronous endpoints in the OUT direction, this address location is referred to as USB\_ADDRn\_RX\_1.*

*In case of double-buffered or isochronous endpoints in the IN direction, this address location is used for USB\_ADDRn\_RX\_1.*

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

Bits 15:1 **ADDRn\_RX[15:1]**: Reception buffer address

These bits point to the starting address of the packet buffer, which will contain the data received by the endpoint associated with the USB\_EPnR register at the next OUT/SETUP token addressed to it.

Bit 0 This bit must always be written as '0' since packet memory is half-word wide and all packet buffers must be half-word aligned.

### Reception byte count n (USB\_COUNTn\_RX)

Address offset: [USB\_BTABLE] + n\*8 + 6

**Note:**

*In case of double-buffered or isochronous endpoints in the OUT direction, this address location is referred to as USB\_COUNTn\_RX\_1.*

*In case of double-buffered or isochronous endpoints in the IN direction, this address location is used for USB\_COUNTn\_RX\_1.*

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

This table location is used to store two different values, both required during packet reception. The most significant bits contains the definition of allocated buffer size, to allow buffer overflow detection, while the least significant part of this location is written back by the USB peripheral at the end of reception to give the actual number of received bytes. Due to the restrictions on the number of available bits, buffer size is represented using the number of allocated memory blocks, where block size can be selected to choose the trade-off between fine-granularity/small-buffer and coarse-granularity/large-buffer. The size of allocated buffer is a part of the endpoint descriptor and it is normally defined during the

enumeration process according to its maxPacketSize parameter value (See “Universal Serial Bus Specification”).

**Bit 15 BL\_SIZE:** Block size

This bit selects the size of memory block used to define the allocated buffer area.

- If BL\_SIZE=0, the memory block is 2-byte large, which is the minimum block allowed in a half-word wide memory. With this block size the allocated buffer size ranges from 2 to 62 bytes.
- If BL\_SIZE=1, the memory block is 32-byte large, which allows to reach the maximum packet length defined by USB specifications. With this block size the allocated buffer size theoretically ranges from 32 to 1024 bytes, which is the longest packet size allowed by USB standard specifications. However, the applicable size is limited by the available buffer memory.

**Bits 14:10 NUM\_BLOCK[4:0]:** Number of blocks

These bits define the number of memory blocks allocated to this packet buffer. The actual amount of allocated memory depends on the BL\_SIZE value as illustrated in [Table 130](#).

**Bits 9:0 COUNTn\_RX[9:0]:** Reception byte count

These bits contain the number of bytes received by the endpoint associated with the USB\_EPnR register during the last OUT/SETUP transaction addressed to it.

**Table 130. Definition of allocated buffer memory**

Value of NUM_BLOCK[4:0]	Memory allocated when BL_SIZE=0	Memory allocated when BL_SIZE=1
0 ('00000)	Not allowed	32 bytes
1 ('00001)	2 bytes	64 bytes
2 ('00010)	4 bytes	96 bytes
3 ('00011)	6 bytes	128 bytes
...	...	...
14 ('01110)	28 bytes	480 bytes
15 ('01111)	30 bytes	512 bytes
16 ('10000)	32 bytes	544 bytes
...	...	...
29 ('11101)	58 bytes	960 bytes
30 ('11110)	60 bytes	992 bytes
31 ('11111)	62 bytes	N/A

### 30.6.3 USB register map

The table below provides the USB register map and reset values.

**Table 131. USB register map and reset values**

Offset	Register	Reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
0x00	<b>USB_EP0R</b>		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																							
0x04	<b>USB_EP1R</b>		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.				
	Reset value																																							
0x08	<b>USB_EP2R</b>		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.				
	Reset value																																							
0x0C	<b>USB_EP3R</b>		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.				
	Reset value																																							
0x10	<b>USB_EP4R</b>		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.				
	Reset value																																							
0x14	<b>USB_EP5R</b>		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.				
	Reset value																																							
0x18	<b>USB_EP6R</b>		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.				
	Reset value																																							
0x1C	<b>USB_EP7R</b>		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																							
0x20-0x3F Reserved																																								
0x40	<b>USB_CNTR</b>		Res.	Res.	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX	CTR_RX						
	Reset value																																							
0x44	<b>USBISTR</b>		Res.	Res.	CTR	CTR	RXDP	RXDM	RMAOVR	RMAOVRM	ERR	LCK	WKUPM	SUSPM	RESETM	SOFM	ESOFM	L1REQM	DIR	RESUME																				
	Reset value																																							
0x48	<b>USB_FNR</b>		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																							
0x4C	<b>USB_DADDR</b>		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value																																							

**Table 131. USB register map and reset values (continued)**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
0x50	<b>USB_BTABLE</b>	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0															
	Reset value	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0															
0x54	<b>USB_LPMCSR</b>	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0															
	Reset value	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0															
0x58	<b>USB_BCDR</b>	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0															
	Reset value	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0															

Refer to [Section 2.2 on page 46](#) for the register boundary addresses.

## 31 HDMI-CEC controller (CEC)

This applies to STM32F05x, STM32F04x, STM32F07x and STM32F09x devices only.

### 31.1 Introduction

Consumer electronics control (CEC) is part of HDMI (high-definition multimedia interface) standard. It contains a protocol that provides high-level control functions between various audiovisual products. CEC operates at low speeds, with minimum processing and memory overhead.

The HDMI-CEC controller provides hardware support for this protocol.

### 31.2 HDMI-CEC controller main features

- Complies with HDMI-CEC v1.4 specification
- 32 kHz CEC kernel with 2 clock source options
  - HSI RC oscillator with fixed prescaler (HSI/244)
  - LSE oscillator
- Works in Stop mode for ultra-low-power applications
- Configurable signal-free time before start of transmission
  - Automatic by hardware, according to CEC state and transmission history
  - Fixed by software (7 timing options)
- Configurable peripheral address (OAR)
- Supports Listen mode
  - Enables reception of CEC messages sent to destination address different from OAR without interfering with the CEC line
- Configurable Rx-tolerance margin
  - Standard tolerance
  - Extended tolerance
- Receive-error detection
  - Bit rising error (BRE), with optional stop of reception (BRESTP)
  - Short bit period error (SBPE)
  - Long bit period error (LBPE)
- Configurable error-bit generation
  - on BRE detection (BREGEN)
  - on LBPE detection (LBPEGEN)
  - always generated on SBPE detection
- Transmission error detection (TXERR)
- Arbitration lost detection (ARBLST)
  - with automatic transmission retry
- Transmission underrun detection (TXUDR)
- Reception overrun detection (RXOVR)

## 31.3 HDMI-CEC functional description

### 31.3.1 HDMI-CEC pin

The CEC bus consists of a single bidirectional line that is used to transfer data in and out of the device. It is connected to a +3.3 V supply voltage via a 27 kΩ pull-up resistor. The output stage of the device must have an open-drain or open-collector to allow a wired-AND connection.

The HDMI-CEC controller manages the CEC bidirectional line as an alternate function of a standard GPIO, assuming that it is configured as alternate function open drain. The 27 kΩ pull-up must be added externally to the microcontroller.

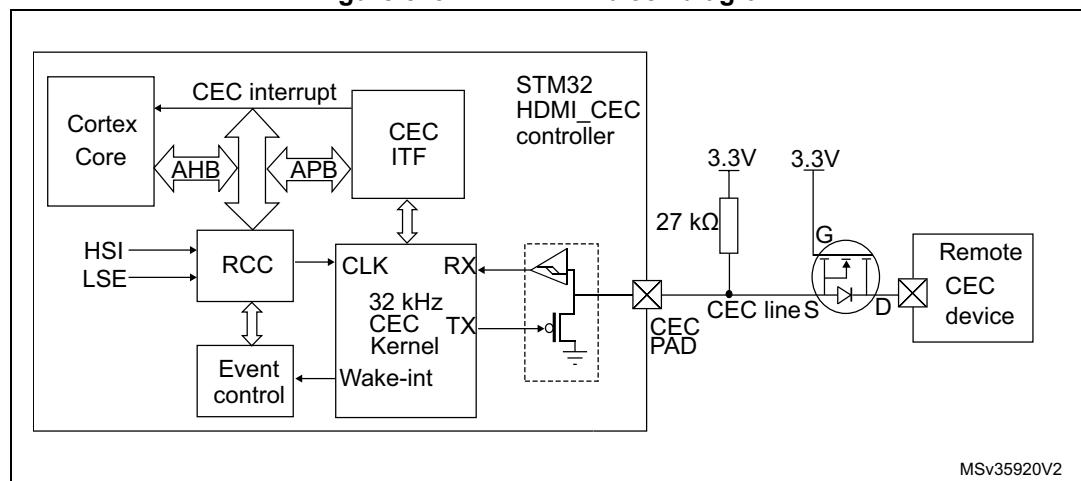
To not interfere with the CEC bus when the application power is removed, it is mandatory to isolate the CEC pin from the bus in such conditions. This can be done by using a MOS transistor, as shown on [Figure 329](#).

**Table 132. HDMI pin**

Name	Signal type	Remarks
CEC	Bidirectional	Two states: – 1 = high impedance – 0 = low impedance A 27 kΩ resistor must be added externally.

### 31.3.2 HDMI-CEC block diagram

**Figure 329. HDMI-CEC block diagram**



### 31.3.3 Message description

All transactions on the CEC line consist of an initiator and one or more followers. The initiator is responsible for sending the message structure and the data. The follower is the recipient of any data and is responsible for setting any acknowledgment bits.

A message is conveyed in a single frame that consists of a start bit followed by a header block and optionally an opcode and a variable number of operand blocks.

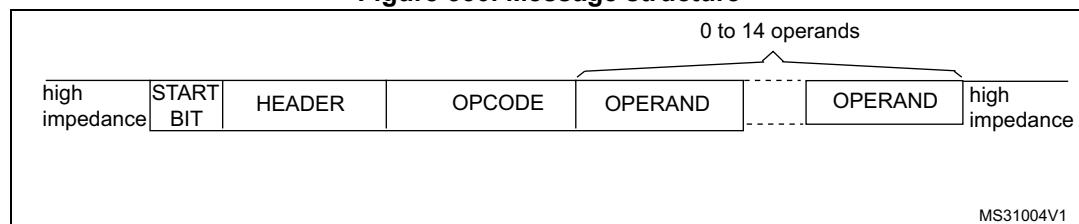
All these blocks are made of a 8-bit payload - most significant bit is transmitted first - followed by an end of message (EOM) bit and an acknowledge (ACK) bit.

The EOM bit is set in the last block of a message and kept reset in all others. In case a message contains additional blocks after an EOM is indicated, those additional blocks must be ignored. The EOM bit may be set in the header block to 'ping' other devices, to make sure they are active.

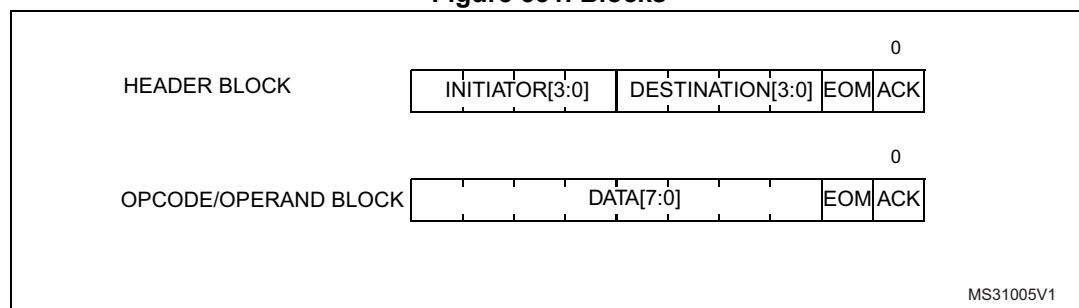
The acknowledge bit is always set to high impedance by the initiator so that it can be driven low either by the follower that has read its own address in the header, or by the follower that needs to reject a broadcast message.

The header consists of the source logical address field, and the destination logical address field. Note that the special address 0xF is used for broadcast messages.

**Figure 330. Message structure**



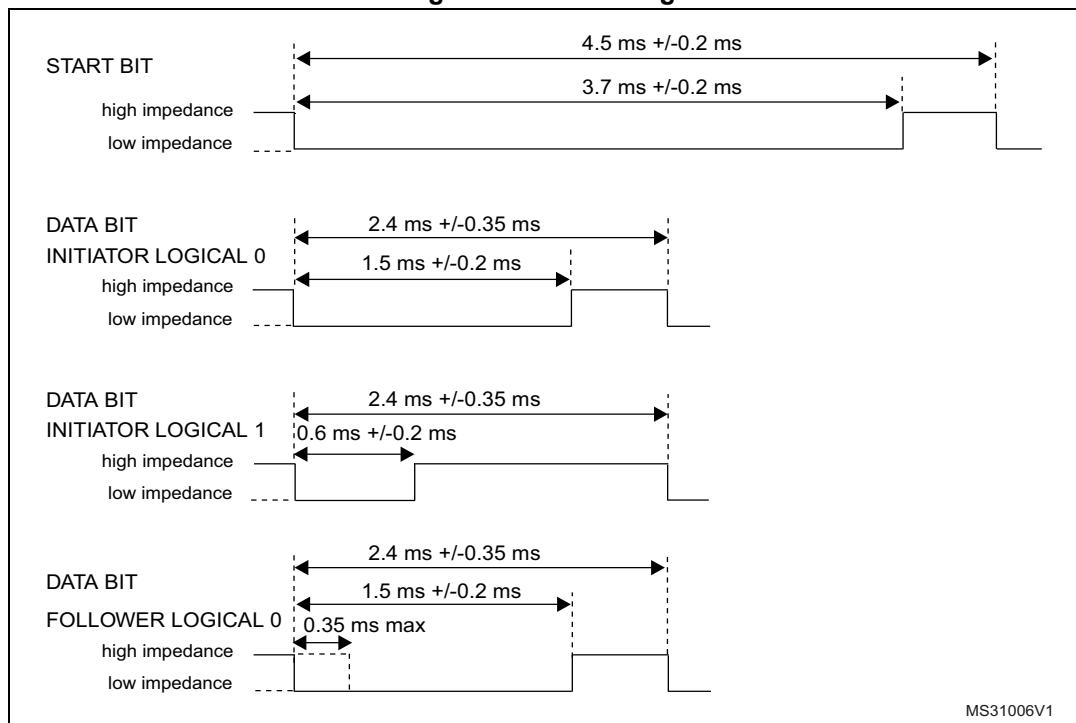
**Figure 331. Blocks**



### 31.3.4 Bit timing

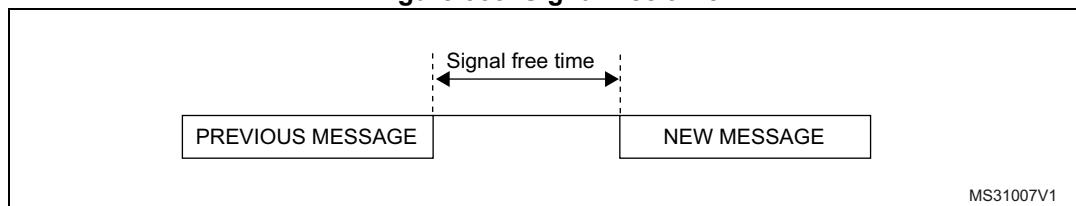
The format of the start bit is unique and identifies the start of a message. It must be validated by its low duration and its total duration.

All remaining data bits in the message, after the start bit, have consistent timing. The high-to-low transition at the end of the data bit is the start of the next data bit except for the final bit where the CEC line remains high.

**Figure 332. Bit timings**

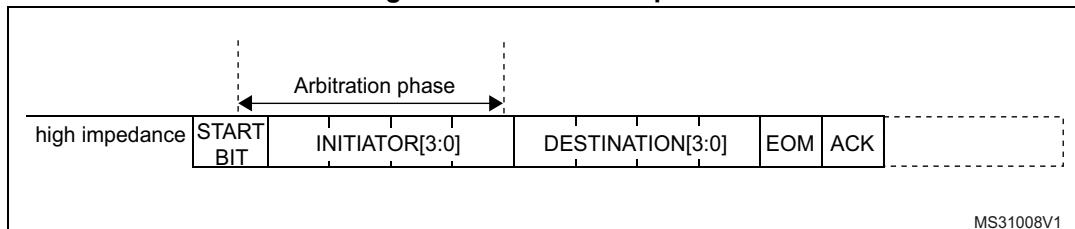
### 31.4 Arbitration

All devices transmitting - or retransmitting - a message onto the CEC line must ensure that it has been inactive for a number of bit periods. This signal-free time is defined as the time starting from the final bit of the previous frame and depends on the initiating device and the current status as shown in the figure below.

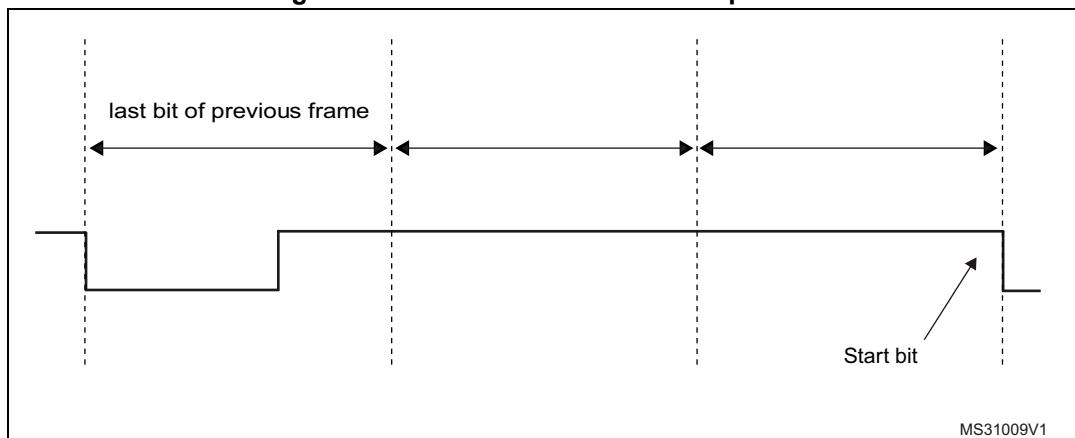
**Figure 333. Signal free time**

Since only one initiator is allowed at any one time, an arbitration mechanism is provided to avoid conflict when more than one initiator begins transmitting at the same time.

CEC line arbitration starts with the leading edge of the start bit and continues until the end of the initiator address bits within the header block. During this period, the initiator must monitor the CEC line, if whilst driving the line to high impedance it reads it back to 0. Assuming then it has lost arbitration, it stops transmitting and becomes a follower.

**Figure 334. Arbitration phase**

*Figure 335* shows an example for a SFT of three nominal bit periods.

**Figure 335. SFT of three nominal bit periods**

A configurable time window is counted before starting the transmission.

In the SFT = 0 configuration, HDMI-CEC performs automatic SFT calculation ensuring compliance with the HDMI-CEC standard:

- 2.5 data bit periods if the CEC is the last bus initiator with unsuccessful transmission
- 4 data bit periods if the CEC is the new bus initiator
- 6 data bit periods if the CEC is the last bus initiator with successful transmission

This is done to guarantee the maximum priority to a failed transmission and the lowest one to the last initiator that completed successfully its transmission.

Otherwise there is the possibility to configure the SFT bits to count a fixed timing value. Possible values are 0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5 data bit periods.

### 31.4.1 SFT option bit

In case of SFTOPT = 0 configuration, SFT starts being counted when the start-of-transmission command is set by software (TXSOM = 1).

In case of SFTOPT = 1, SFT starts automatically being counted by the HDMI-CEC device when a bus-idle or line error condition is detected. If the SFT timer is completed at the time TXSOM command is set then transmission starts immediately without latency. If the SFT timer is still running instead, the system waits until the timer elapses before transmission can start.

In case of SFTOPT = 1 a bus-event condition starting the SFT timer is detected in the following cases:

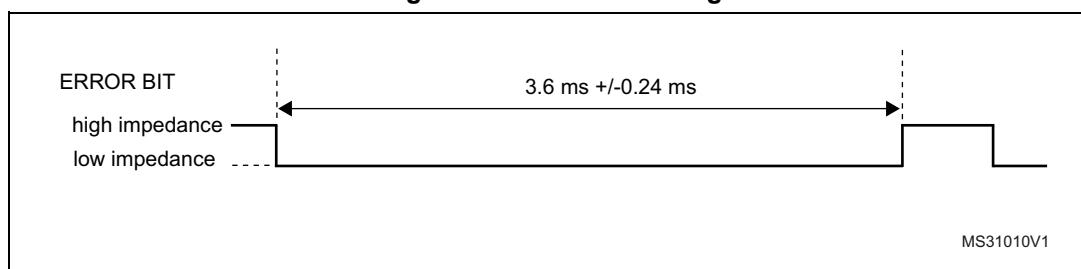
- In case of a regular end of transmission/reception, when TXEND/RXEND bits are set at the minimum nominal data bit duration of the last bit in the message (ACK bit).
- In case of a transmission error detection, SFT timer starts when the TXERR transmission error is detected (TXERR = 1).
- In case of a missing acknowledge from the CEC follower, the SFT timer starts when the TXACKE bit is set, that is at the nominal sampling time of the ACK bit.
- In case of a transmission underrun error, the SFT timer starts when the TXUDR bit is set at the end of the ACK bit.
- In case of a receive error detection implying reception abort, the SFT timer starts at the same time the error is detected. If an error bit is generated, then SFT starts being counted at the end of the error bit.
- In case of a wrong start bit or of any uncodified low impedance bus state from idle, the SFT timer is restarted as soon as the bus comes back to hi-impedance idleness.

## 31.5 Error handling

### 31.5.1 Bit error

If a data bit - excluding the start bit - is considered invalid, the follower is expected to notify such error by generating a low bit period on the CEC line of 1.4 to 1.6 times the nominal data bit period (3.6 ms nominally).

**Figure 336. Error bit timing**



### 31.5.2 Message error

A message is considered lost and therefore may be retransmitted under the following conditions:

- a message is not acknowledged in a directly addressed message
- a message is negatively acknowledged in a broadcast message
- a low impedance is detected on the CEC line while it is not expected (line error)

Three kinds of error flag can be detected when the CEC interface is receiving a data bit:

### 31.5.3 Bit rising error (BRE)

BRE (bit rising error) is set when a bit rising edge is detected outside the windows where it is expected (see [Figure 337](#)). BRE flag also generates a CEC interrupt if the BREIE = 1.

In the case of a BRE detection, the message reception can be stopped according to the BRESTP bit value and an error bit can be generated if BREGEN bit is set.

When BRE is detected in a broadcast message with BRESTP = 1 an error bit is generated even if BREGEN = 0 to enforce initiator's retry of the failed transmission. Error bit generation can be disabled by configuring BREGEN = 0, BRDNOGEN = 1.

### 31.5.4 Short bit period error (SBPE)

SBPE is set when a bit falling edge is detected earlier than expected (see [Figure 337](#)). SBPE flag also generates a CEC interrupt if the SBPEIE = 1.

An error bit is always generated on the line in case of a SBPE error detection. An error bit is not generated upon SBPE detection only when Listen mode is set (LSTN = 1) and the following conditions are met:

- A directly addressed message is received containing SBPE
- A broadcast message is received containing SBPE AND BRDNOGEN = 1

### 31.5.5 Long bit period error (LBPE)

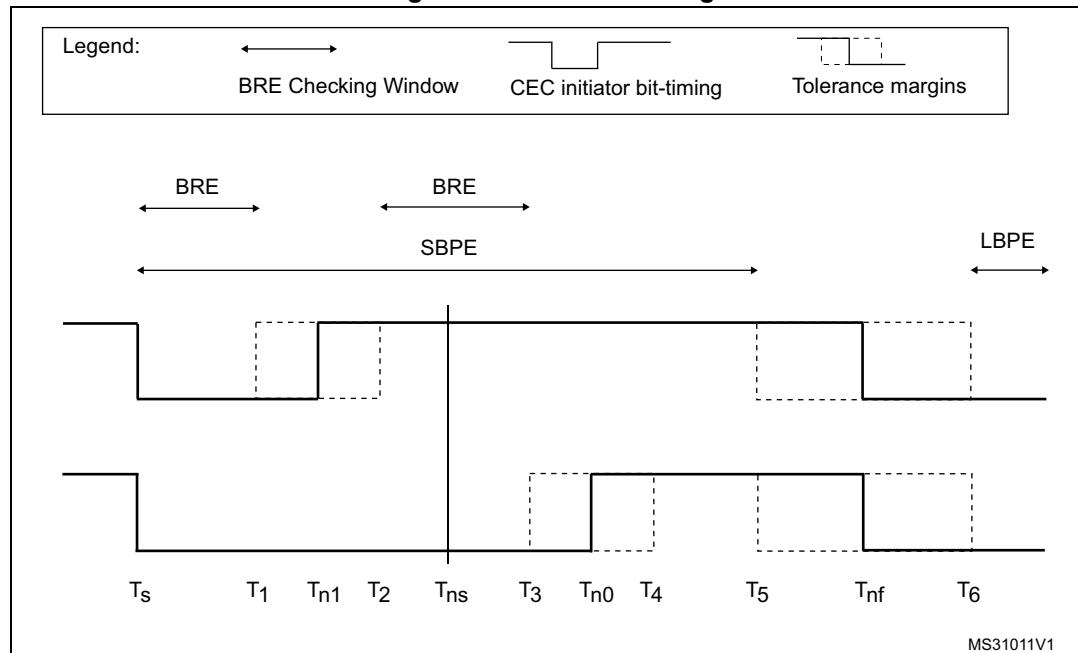
LBPE is set when a bit falling edge is not detected in a valid window (see [Figure 337](#)). LBPE flag also generates a CEC interrupt if the LBPEIE = 1.

LBPE always stops the reception, an error bit is generated on the line when LBPEGEN bit is set.

When LBPE is detected in a broadcast message an error bit is generated even if LBPEGEN = 0 to enforce initiator's retry of the failed transmission. Error bit generation can be disabled by configuring LBPEGEN = 0, BRDNOGEN = 1.

*Note:* The BREGEN = 1, BRESTP = 0 configuration must be avoided.

**Figure 337. Error handling**



**Table 133. Error handling timing parameters**

Time	RXTOL	ms	Description
T <sub>s</sub>	x	0	Bit start event.
T <sub>1</sub>	1	0.3	The earliest time for a low - high transition when indicating a logical 1.
	0	0.4	
T <sub>n1</sub>	x	0.6	The nominal time for a low - high transition when indicating a logical 1.
T <sub>2</sub>	0	0.8	The latest time for a low - high transition when indicating a logical 1.
	1	0.9	
T <sub>ns</sub>	x	1.05	Nominal sampling time.
T <sub>3</sub>	1	1.2	The earliest time a device is permitted return to a high impedance state (logical 0).
	0	1.3	
T <sub>n0</sub>	x	1.5	The nominal time a device is permitted return to a high impedance state (logical 0).
T <sub>4</sub>	0	1.7	The latest time a device is permitted return to a high impedance state (logical 0).
	1	1.8	
T <sub>5</sub>	1	1.85	The earliest time for the start of a following bit.
	0	2.05	
T <sub>nf</sub>	x	2.4	The nominal data bit period.
T <sub>6</sub>	0	2.75	The latest time for the start of a following bit.
	1	2.95	

### 31.5.6 Transmission error detection (TXERR)

The CEC initiator sets the TXERR flag if detecting low impedance on the CEC line when it is transmitting high impedance and is not expecting a follower asserted bit. TXERR flag also generates a CEC interrupt if the TXERRIE = 1.

TXERR assertion stops the message transmission. Application is in charge to retry the failed transmission up to five times.

TXERR checks are performed differently depending on the different states of the CEC line and on the RX tolerance configuration.

Figure 338. TXERR detection

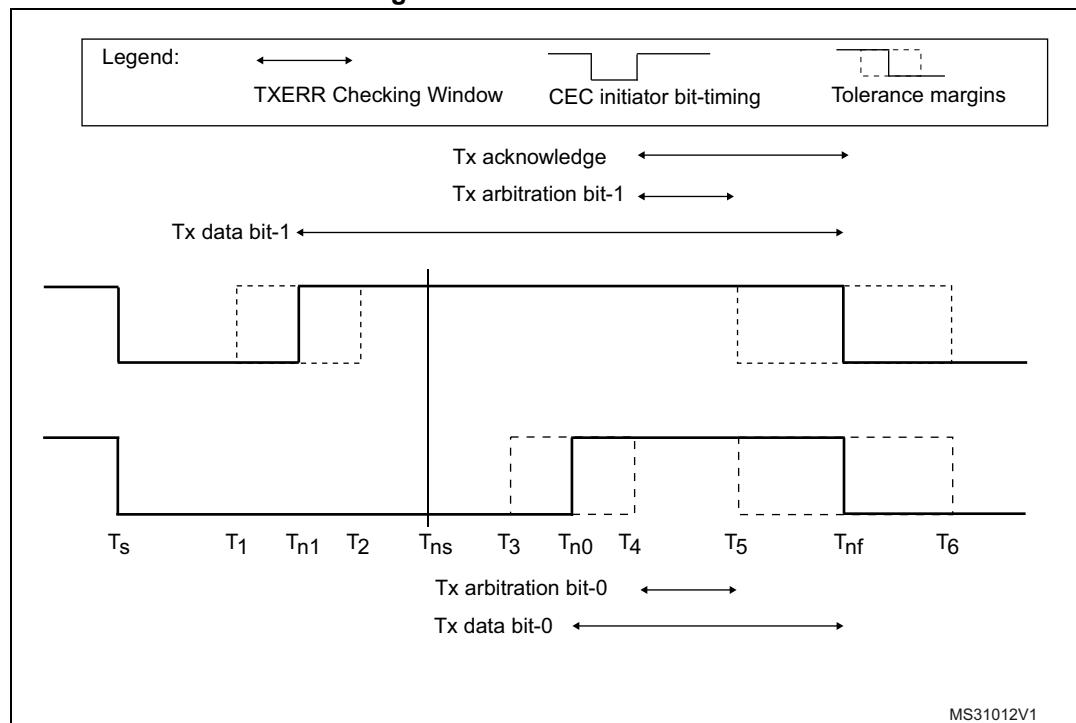


Table 134. TXERR timing parameters

Time	RXTOL	ms	Description
$T_s$	x	0	Bit start event.
$T_1$	1	0.3	The earliest time for a low - high transition when indicating a logical 1.
	0	0.4	
$T_{n1}$	x	0.6	The nominal time for a low - high transition when indicating a logical 1.
$T_2$	0	0.8	The latest time for a low - high transition when indicating a logical 1.
	1	0.9	
$T_{ns}$	x	1.05	Nominal sampling time.
$T_3$	1	1.2	The earliest time a device is permitted return to a high impedance state (logical 0).
	0	1.3	
$T_{n0}$	x	1.5	The nominal time a device is permitted return to a high impedance state (logical 0).
$T_4$	0	1.7	The latest time a device is permitted return to a high impedance state (logical 0).
	1	1.8	
$T_5$	1	1.85	The earliest time for the start of a following bit.
	0	2.05	
$T_{nf}$	x	2.4	The nominal data bit period.

**Table 134. TXERR timing parameters (continued)**

Time	RXTOL	ms	Description
$T_6$	0	2.75	The latest time for the start of a following bit.
	1	2.95	

## 31.6 HDMI-CEC interrupts

An interrupt can be produced:

- during reception if a receive block transfer is finished or if a receive error occurs.
- during transmission if a transmit block transfer is finished or if a transmit error occurs.

**Table 135. HDMI-CEC interrupts**

Interrupt event	Event flag	Enable control bit
Rx-byte received	RXBR	RXBRIE
End of reception	RXEND	RXENDIE
Rx-overrun	RXOVR	RXOVRIE
Rxbit rising error	BRE	BREIE
Rx-short bit period error	SBPE	SBPEIE
Rx-long bit period error	LBPE	LBPEIE
Rx-missing acknowledge error	RXACKE	RXACKEIE
Arbitration lost	ARBLST	ARBLSTIE
Tx-byte request	TXBR	TXBRIE
End of transmission	TXEND	TXENDIE
Tx-buffer underrun	TXUDR	TXUDRIE
Tx-error	TXERR	TXERRIE
Tx-missing acknowledge error	TXACKE	TXACKEIE

For code example refer to the Appendix sections [A.13.1: HDMI-CEC configure CEC code example](#), [A.13.2: HDMI-CEC transmission with interrupt enabled code example](#) and [A.13.3: HDMI-CEC interrupt management code example](#).

## 31.7 HDMI-CEC registers

Refer to [Section 1.2](#) for a list of abbreviations used in register descriptions. The registers have to be accessed by words (32 bits).

### 31.7.1 CEC control register (CEC\_CR)

Address offset: 0x00

Reset value: 0x0000 0000

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

Bits 31:3 Reserved, must be kept at reset value.

#### Bit 2 TXEOM: Tx end of message

The TXEOM bit is set by software to command transmission of the last byte of a CEC message. TXEOM is cleared by hardware at the same time and under the same conditions as for TXSOM.

0: TXDR data byte is transmitted with EOM = 0

1: TXDR data byte is transmitted with EOM = 1

*Note:* TXEOM must be set when CECEN = 1.

*TXEOM must be set before writing transmission data to TXDR.*

*If TXEOM is set when TXSOM = 0, transmitted message consists of 1 byte (HEADER) only (PING message).*

#### Bit 1 TXSOM: Tx start of message

TXSOM is set by software to command transmission of the first byte of a CEC message. If the CEC message consists of only one byte, TXEOM must be set before TXSOM.

Start-bit is effectively started on the CEC line after SFT is counted. If TXSOM is set while a message reception is ongoing, transmission starts after the end of reception.

TXSOM is cleared by hardware after the last byte of the message is sent with a positive acknowledge (TXEND = 1), in case of transmission underrun (TXUDR = 1), negative acknowledge (TXACKE = 1), and transmission error (TXERR = 1). It is also cleared by CECEN = 0. It is not cleared and transmission is automatically retried in case of arbitration lost (ARBLST = 1).

TXSOM can be also used as a status bit informing application whether any transmission request is pending or under execution. The application can abort a transmission request at any time by clearing the CECEN bit.

0: No CEC transmission is on-going

1: CEC transmission command

*Note:* TXSOM must be set when CECEN = 1.

*TXSOM must be set when transmission data is available into TXDR.*

*HEADER first four bits containing own peripheral address are taken from TXDR[7:4], not from CEC\_CFGR.OAR that is used only for reception.*

**Bit 0 CECEN: CEC enable**

The CECEN bit is set and cleared by software. CECEN = 1 starts message reception and enables the TXSOM control. CECEN = 0 disables the CEC peripheral, clears all bits of CEC\_CR register and aborts any on-going reception or transmission.

0: CEC peripheral is off.

1: CEC peripheral is on.

### 31.7.2 CEC configuration register (CEC\_CFGR)

This register is used to configure the HDMI-CEC controller.

Address offset: 0x04

Reset value: 0x0000 0000

**Caution:** It is mandatory to write CEC\_CFGR only when CECEN = 0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LSTN	OAR[14:0]														
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	SFT OP	BRDN OGEN	LBPE GEN	BRE GEN	BRE STP	RX TOL	SFT[2:0]		
							rw	rw	rw	rw	rw	rw	rw	rw	rw

**Bit 31 LSTN: Listen mode**

LSTN bit is set and cleared by software.

0: CEC peripheral receives only message addressed to its own address (OAR). Messages addressed to different destination are ignored. Broadcast messages are always received.

1: CEC peripheral receives messages addressed to its own address (OAR) with positive acknowledge. Messages addressed to different destination are received, but without interfering with the CEC bus: no acknowledge sent.

**Bits 30:16 OAR[14:0]: Own addresses configuration**

The OAR bits are set by software to select which destination logical addresses has to be considered in receive mode. Each bit, when set, enables the CEC logical address identified by the given bit position.

At the end of HEADER reception, the received destination address is compared with the enabled addresses. In case of matching address, the incoming message is acknowledged and received. In case of non-matching address, the incoming message is received only in listen mode (LSTN = 1), but without acknowledge sent. Broadcast messages are always received.

Example:

OAR = 0b000 0000 0010 0001 means that CEC acknowledges addresses 0x0 and 0x5.

Consequently, each message directed to one of these addresses is received.

**Bits 15:9 Reserved, must be kept at reset value.**

**Bit 8 SFTOP: SFT option bit**

The SFTOPT bit is set and cleared by software.

0: SFT timer starts when TXSOM is set by software.

1: SFT timer starts automatically at the end of message transmission/reception.

**Bit 7 BRDNOGEN:** Avoid error-bit generation in broadcast

The BRDNOGEN bit is set and cleared by software.

0: BRE detection with BRESTOP = 1 and BREGEN = 0 on a broadcast message generates an error-bit on the CEC line. LBPE detection with LBPEGEN = 0 on a broadcast message generates an error-bit on the CEC line.

1: Error-bit is not generated in the same condition as above. An error-bit is not generated even in case of an SBPE detection in a broadcast message if listen mode is set.

**Bit 6 LBPEGEN:** Generate error-bit on long bit period error

The LBPEGEN bit is set and cleared by software.

0: LBPE detection does not generate an error-bit on the CEC line.

1: LBPE detection generates an error-bit on the CEC line.

*Note: If BRDNOGEN = 0, an error-bit is generated upon LBPE detection in broadcast even if LBPEGEN = 0.*

**Bit 5 BREGEN:** Generate error-bit on bit rising error

The BREGEN bit is set and cleared by software.

0: BRE detection does not generate an error-bit on the CEC line.

1: BRE detection generates an error-bit on the CEC line (if BRESTOP is set).

*Note: If BRDNOGEN = 0, an error-bit is generated upon BRE detection with BRESTOP = 1 in broadcast even if BREGEN = 0.*

**Bit 4 BRESTOP:** Rx-stop on bit rising error

The BRESTOP bit is set and cleared by software.

0: BRE detection does not stop reception of the CEC message. Data bit is sampled at 1.05 ms.

1: BRE detection stops message reception.

**Bit 3 RXTOL:** Rx-tolerance

The RXTOL bit is set and cleared by software.

0: Standard tolerance margin:

- Start-bit, +/- 200 µs rise, +/- 200 µs fall
- Data-bit: +/- 200 µs rise, +/- 350 µs fall

1: Extended tolerance

- Start-bit: +/- 400 µs rise, +/- 400 µs fall
- Data-bit: +/-300 µs rise, +/- 500 µs fall

**Bits 2:0 SFT[2:0]:** Signal free time

SFT bits are set by software. In the SFT = 0x0 configuration, the number of nominal data bit periods waited before transmission is ruled by hardware according to the transmission history. In all the other configurations the SFT number is determined by software.

0x0

- 2.5 data-bit periods if CEC is the last bus initiator with unsuccessful transmission (ARBLST = 1, TXERR = 1, TXUDR = 1 or TXACKE = 1)
- 4 data-bit periods if CEC is the new bus initiator
- 6 data-bit periods if CEC is the last bus initiator with successful transmission (TXEOM = 1)

0x1: 0.5 nominal data bit periods

0x2: 1.5 nominal data bit periods

0x3: 2.5 nominal data bit periods

0x4: 3.5 nominal data bit periods

0x5: 4.5 nominal data bit periods

0x6: 5.5 nominal data bit periods

0x7: 6.5 nominal data bit periods

### 31.7.3 CEC Tx data register (CEC\_TXDR)

Address offset: 0x8

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
									w	w	w	w	w	w	w
TXD[7:0]															

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

Bits 7:0 **TXD[7:0]**: Tx data

TXD is a write-only register containing the data byte to be transmitted.

### 31.7.4 CEC Rx data register (CEC\_RXDR)

Address offset: 0xC

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	r	r	r	r	r	r	r
RXD[7:0]															

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

Bits 7:0 **RXD[7:0]**: Rx data

RXD is read-only and contains the last data byte that has been received from the CEC line.

### 31.7.5 CEC interrupt and status register (CEC\_ISR)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	TX ACKE	TX ERR	TX UDR	TX END	TXBR	ARB LST	RX ACKE	LBPE	SBPE	BRE	RX OVR	RX END	RXBR
			rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:13 Reserved, must be kept at reset value.

Bit 12 **TXACKE:** Tx-missing acknowledge error

In transmission mode, TXACKE is set by hardware to inform application that no acknowledge was received. In case of broadcast transmission, TXACKE informs application that a negative acknowledge was received. TXACKE aborts message transmission and clears TXSOM and TXEOM controls.

TXACKE is cleared by software write at 1.

Bit 11 **TXERR:** Tx-error

In transmission mode, TXERR is set by hardware if the CEC initiator detects low impedance on the CEC line while it is released. TXERR aborts message transmission and clears TXSOM and TXEOM controls.

TXERR is cleared by software write at 1.

Bit 10 **TXUDR:** Tx-buffer underrun

In transmission mode, TXUDR is set by hardware if application was not in time to load TXDR before of next byte transmission. TXUDR aborts message transmission and clears TXSOM and TXEOM control bits.

TXUDR is cleared by software write at 1

Bit 9 **TXEND:** End of transmission

TXEND is set by hardware to inform application that the last byte of the CEC message has been successfully transmitted. TXEND clears the TXSOM and TXEOM control bits.

TXEND is cleared by software write at 1.

Bit 8 **TXBR:** Tx-byte request

TXBR is set by hardware to inform application that the next transmission data has to be written to TXDR. TXBR is set when the 4th bit of currently transmitted byte is sent. Application must write the next byte to TXDR within six nominal data-bit periods before transmission underrun error occurs (TXUDR).

TXBR is cleared by software write at 1.

Bit 7 **ARBLST:** Arbitration lost

ARBLST is set by hardware to inform application that CEC device is switching to reception due to arbitration lost event following the TXSOM command. ARBLST can be due either to a contending CEC device starting earlier or starting at the same time but with higher HEADER priority. After ARBLST assertion TXSOM bit keeps pending for next transmission attempt.

ARBLST is cleared by software write at 1.

Bit 6 **RXACKE:** Rx-missing acknowledge

In receive mode, RXACKE is set by hardware to inform application that no acknowledge was seen on the CEC line. RXACKE applies only for broadcast messages and in listen mode also for not directly addressed messages (destination address not enabled in OAR). RXACKE aborts message reception.

RXACKE is cleared by software write at 1.

Bit 5 **LBPE:** Rx-long bit period error

LBPE is set by hardware in case a data-bit waveform is detected with long bit period error. LBPE is set at the end of the maximum bit-extension tolerance allowed by RXTOL, in case falling edge is still longing. LBPE always stops reception of the CEC message. LBPE generates an error-bit on the CEC line if LBPEGEN = 1. In case of broadcast, error-bit is generated even in case of LBPEGEN = 0.

LBPE is cleared by software write at 1.

**Bit 4 SBPE:** Rx-short bit period error

SBPE is set by hardware in case a data-bit waveform is detected with short bit period error. SBPE is set at the time the anticipated falling edge occurs. SBPE generates an error-bit on the CEC line. SBPE is cleared by software write at 1.

**Bit 3 BRE:** Rx-bit rising error

BRE is set by hardware in case a data-bit waveform is detected with bit rising error. BRE is set either at the time the misplaced rising edge occurs, or at the end of the maximum BRE tolerance allowed by RXTOL, in case rising edge is still longing. BRE stops message reception if BRESTP = 1. BRE generates an error-bit on the CEC line if BREGEN = 1.

BRE is cleared by software write at 1.

**Bit 2 RXOVR:** Rx-overrun

RXOVR is set by hardware if RXBR is not yet cleared at the time a new byte is received on the CEC line and stored into RXD. RXOVR assertion stops message reception so that no acknowledge is sent. In case of broadcast, a negative acknowledge is sent.

RXOVR is cleared by software write at 1.

**Bit 1 RXEND:** End of reception

RXEND is set by hardware to inform application that the last byte of a CEC message is received from the CEC line and stored into the RXD buffer. RXEND is set at the same time of RXBR.

RXEND is cleared by software write at 1.

**Bit 0 RXBR:** Rx-byte received

The RXBR bit is set by hardware to inform application that a new byte has been received from the CEC line and stored into the RXD buffer.

RXBR is cleared by software write at 1.

### 31.7.6 CEC interrupt enable register (CEC\_IER)

Address offset: 0x14

Reset value: 0x0000 0000

**Caution:** It is mandatory to write CEC\_IER only when CECEN = 0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	TXACKIE	TXERRIE	TXUDRIE	TXENDIE	TXBRIE	ARBLSTIE	RXACKIE	LBPEIE	SBPEIE	BREIE	RXOVRIE	RXENDIE	RXBRIE
			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:13 Reserved, must be kept at reset value.

**Bit 12 TXACKIE:** Tx-missing acknowledge error interrupt enable

The TXACKIE bit is set and cleared by software.

0: TXACKIE interrupt disabled

1: TXACKIE interrupt enabled

**Bit 11 TXERRIE:** Tx-error interrupt enable

The TXERRIE bit is set and cleared by software.

0: TXERRIE interrupt disabled

1: TXERRIE interrupt enabled

- Bit 10 **TXUDRIE**: Tx-underrun interrupt enable  
The TXUDRIE bit is set and cleared by software.  
0: TXUDR interrupt disabled  
1: TXUDR interrupt enabled
- Bit 9 **TXENDIE**: Tx-end of message interrupt enable  
The TXENDIE bit is set and cleared by software.  
0: TXEND interrupt disabled  
1: TXEND interrupt enabled
- Bit 8 **TXBRIE**: Tx-byte request interrupt enable  
The TXBRIE bit is set and cleared by software.  
0: TXBR interrupt disabled  
1: TXBR interrupt enabled
- Bit 7 **ARBLSTIE**: Arbitration lost interrupt enable  
The ARBLSTIE bit is set and cleared by software.  
0: ARBLST interrupt disabled  
1: ARBLST interrupt enabled
- Bit 6 **RXACKIE**: Rx-missing acknowledge error interrupt enable  
The RXACKIE bit is set and cleared by software.  
0: RXACKE interrupt disabled  
1: RXACKE interrupt enabled
- Bit 5 **LBPEIE**: Long bit period error interrupt enable  
The LBPEIE bit is set and cleared by software.  
0: LBPE interrupt disabled  
1: LBPE interrupt enabled
- Bit 4 **SBPEIE**: Short bit period error interrupt enable  
The SBPEIE bit is set and cleared by software.  
0: SBPE interrupt disabled  
1: SBPE interrupt enabled
- Bit 3 **BREIE**: Bit rising error interrupt enable  
The BREIE bit is set and cleared by software.  
0: BRE interrupt disabled  
1: BRE interrupt enabled
- Bit 2 **RXOVRIE**: Rx-buffer overrun interrupt enable  
The RXOVRIE bit is set and cleared by software.  
0: RXOVR interrupt disabled  
1: RXOVR interrupt enabled
- Bit 1 **RXENDIE**: End of reception interrupt enable  
The RXENDIE bit is set and cleared by software.  
0: RXEND interrupt disabled  
1: RXEND interrupt enabled
- Bit 0 **RXBRIE**: Rx-byte received interrupt enable  
The RXBRIE bit is set and cleared by software.  
0: RXBR interrupt disabled  
1: RXBR interrupt enabled

### 31.7.7 HDMI-CEC register map

The following table summarizes the HDMI-CEC registers.

**Table 136. HDMI-CEC register map and reset values**

Offset	Register name	Reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0								
0x00	<b>CEC_CR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.				
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
0x04	<b>CEC_CFGR</b>	Res.	LSTN	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.				
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x08	<b>CEC_TXDR</b>	Res.	OAR[14:0]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.				
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x0C	<b>CEC_RXDR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x10	<b>CEC_ISR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x14	<b>CEC_IER</b>	Res.	TXACKIE	0	TXACKIE	Res.																																				
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Refer to [Section 2.2](#) for the register boundary addresses.

## 32 Debug support (DBG)

### 32.1 Overview

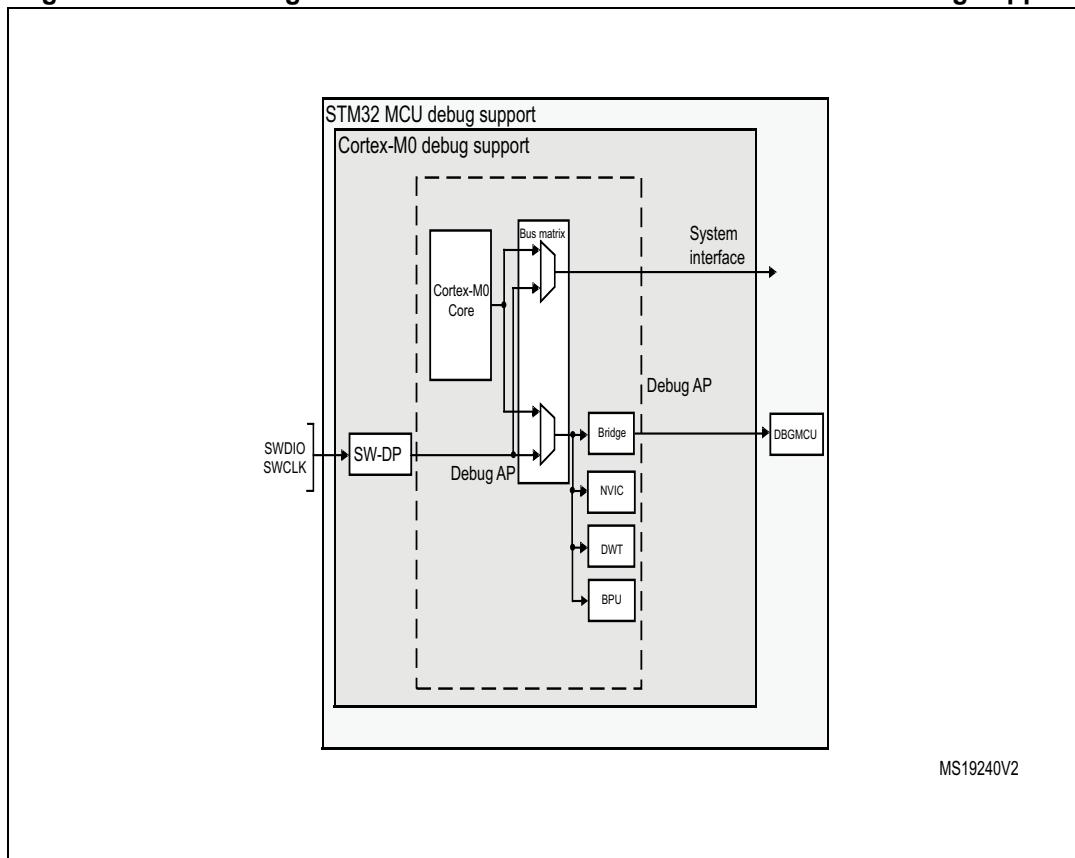
The STM32F0xx devices are built around a Cortex<sup>®</sup>-M0 core, which contains hardware extensions for advanced debugging features. The debug extensions allow the core to be stopped either on a given instruction fetch (breakpoint) or data access (watchpoint). When stopped, the core's internal state and the system's external state may be examined. Once examination is complete, the core and the system may be restored and program execution resumed.

The debug features are used by the debugger host when connecting to and debugging the STM32F0xx MCUs.

One interface for debug is available:

- Serial wire

**Figure 339. Block diagram of STM32F0xx MCU and Cortex<sup>®</sup>-M0-level debug support**



1. The debug features embedded in the Cortex<sup>®</sup>-M0 core are a subset of the Arm CoreSight Design Kit.

The Arm Cortex<sup>®</sup>-M0 core provides integrated on-chip debug support. It is comprised of:

- SW-DP: Serial wire
- BPU: Break point unit
- DWT: Data watchpoint trigger

It also includes debug features dedicated to the STM32F0xx:

- Flexible debug pinout assignment
- MCU debug box (support for low-power modes, control over peripheral clocks, etc.)

**Note:** *For further information on debug functionality supported by the Arm Cortex®-M0 core, refer to the Cortex®-M0 Technical Reference Manual (see [Section 32.2: Reference Arm documentation](#)).*

## 32.2 Reference Arm documentation

- Cortex®-M0 Technical Reference Manual (TRM)  
It is available from:  
<http://infocenter.arm.com>
- Arm Debug Interface V5
- Arm CoreSight Design Kit revision r1p1 Technical Reference Manual

## 32.3 Pinout and debug port pins

The STM32F0xx MCUs are available in various packages with different numbers of available pins.

### 32.3.1 SWD port pins

Two pins are used as outputs for the SW-DP as alternate functions of general purpose I/Os. These pins are available on all packages.

**Table 137. SW debug port pins**

SW-DP pin name	SW debug port		Pin assignment
	Type	Debug assignment	
SWDIO	IO	Serial Wire Data Input/Output	PA13
SWCLK	I	Serial Wire Clock	PA14

### 32.3.2 SW-DP pin assignment

After reset (SYSRESETn or PORESETn), the pins used for the SW-DP are assigned as dedicated pins, immediately usable by the debugger host.

However, the MCU offers the possibility to disable the SWD port and can then release the associated pins for general-purpose I/O (GPIO) usage. For more details on how to disable SW-DP port pins, refer to [Section 8.3.2: I/O pin alternate function multiplexer and mapping on page 151](#).

### 32.3.3 Internal pull-up and pull-down on SWD pins

Once the SW I/O is released by the user software, the GPIO controller takes control of these pins. The reset states of the GPIO control registers put the I/Os in the equivalent states:

- SWDIO: input pull-up
- SWCLK: input pull-down

*Having embedded pull-up and pull-down resistors removes the need to add external resistors.*

## 32.4 ID codes and locking mechanism

There are several ID codes inside the MCU. ST strongly recommends the tool manufacturers (for example Keil, IAR, Raisonance) to lock their debugger using the MCU device ID located at address 0x40015800.

Only the DEV\_ID[15:0] should be used for identification by the debugger/programmer tools (the revision ID must not be taken into account).

### 32.4.1 MCU device ID code

The STM32F0xx products integrate an MCU ID code. This ID identifies the ST MCU part number and the die revision.

This code is accessible by the software debug port (two pins) or by the user software.

For code example refer to the Appendix section [A.12.1: DBG read device ID code example](#).

#### DBGMCU\_IDCODE

Address: 0x40015800

Only 32-bit access supported. Read-only

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

Bits 31:16 **REV\_ID[15:0]** Revision identifier

This field indicates the revision of the device. Refer to [Table 143](#).

Bits 15:12 Reserved: read 0b0110.

Bits 11:0 **DEV\_ID[11:0]**: Device identifier

This field indicates the device ID. Refer to [Table 143](#).

**Table 138. DEV\_ID and REV\_ID field values**

Device	DEV_ID	Revision code	Revision number	REV_ID
STM32F03x	0x444	A or 1	1.0	0x1000
STM32F04x	0x445	A	1.0	0x1000
STM32F05x	0x440	A	1.0	0x1000
		B or 1	2.0	0x2000
STM32F07x	0x448	A	1.0	0x1000
		Z	1.1	0x1001
		B	2.0	0x2000
		Y or 1	2.1	0x2001
STM32F09x	0x442	A	1.0	0x1000

## 32.5 SWD port

### 32.5.1 SWD protocol introduction

This synchronous serial protocol uses two pins:

- SWCLK: clock from host to target
- SWDIO: bidirectional

The protocol allows two banks of registers (DPACC registers and APACC registers) to be read and written to.

Bits are transferred LSB-first on the wire.

For SWDIO bidirectional management, the line must be pulled-up on the board (100 kΩ recommended by Arm).

Each time the direction of SWDIO changes in the protocol, a turnaround time is inserted where the line is not driven by the host nor the target. By default, this turnaround time is one bit time, however this can be adjusted by configuring the SWCLK frequency.

### 32.5.2 SWD protocol sequence

Each sequence consist of three phases:

1. Packet request (8 bits) transmitted by the host
2. Acknowledge response (3 bits) transmitted by the target
3. Data transfer phase (33 bits) transmitted by the host or the target

**Table 139. Packet request (8-bits)**

Bit	Name	Description
0	Start	Must be “1”
1	APnDP	0: DP Access 1: AP Access
2	RnW	0: Write Request 1: Read Request
4:3	A[3:2]	Address field of the DP or AP registers (refer to <a href="#">Table 143 on page 928</a> )
5	Parity	Single bit parity of preceding bits
6	Stop	0
7	Park	Not driven by the host. Must be read as “1” by the target because of the pull-up

Refer to the Cortex®-M0 *TRM* for a detailed description of DPACC and APACC registers.

The packet request is always followed by the turnaround time (default 1 bit) where neither the host nor target drive the line.

**Table 140. ACK response (3 bits)**

Bit	Name	Description
0..2	ACK	001: FAULT 010: WAIT 100: OK

The ACK Response must be followed by a turnaround time only if it is a READ transaction or if a WAIT or FAULT acknowledge has been received.

**Table 141. DATA transfer (33 bits)**

Bit	Name	Description
0..31	WDATA or RDATA	Write or Read data
32	Parity	Single parity of the 32 data bits

The DATA transfer must be followed by a turnaround time only if it is a READ transaction.

### 32.5.3 SW-DP state machine (reset, idle states, ID code)

The State Machine of the SW-DP has an internal ID code which identifies the SW-DP. It follows the JEP-106 standard. This ID code is the default Arm one and is set to **0x0BB11477** (corresponding to Cortex®-M0).

*Note:* Note that the SW-DP state machine is inactive until the target reads this ID code.

- The SW-DP state machine is in RESET STATE either after power-on reset, or after the line is high for more than 50 cycles
- The SW-DP state machine is in IDLE STATE if the line is low for at least two cycles after RESET state.
- After RESET state, it is **mandatory** to first enter into an IDLE state AND to perform a READ access of the DP-SW ID CODE register. Otherwise, the target will issue a FAULT acknowledge response on another transactions.

Further details of the SW-DP state machine can be found in the Cortex®-M0 TRM and the CoreSight Design Kit r1p0 TRM.

### 32.5.4 DP and AP read/write accesses

- Read accesses to the DP are not posted: the target response can be immediate (if ACK=OK) or can be delayed (if ACK=WAIT).
- Read accesses to the AP are posted. This means that the result of the access is returned on the next transfer. If the next access to be done is NOT an AP access, then the DP-RDBUFF register must be read to obtain the result.  
The READOK flag of the DP-CTRL/STAT register is updated on every AP read access or RDBUFF read request to know if the AP read access was successful.
- The SW-DP implements a write buffer (for both DP or AP writes), that enables it to accept a write operation even when other transactions are still outstanding. If the write buffer is full, the target acknowledge response is "WAIT". With the exception of

IDCODE read or CTRL/STAT read or ABORT write which are accepted even if the write buffer is full.

- Because of the asynchronous clock domains SWCLK and HCLK, two extra SWCLK cycles are needed after a write transaction (after the parity bit) to make the write effective internally. These cycles should be applied while driving the line low (IDLE state)  
This is particularly important when writing the CTRL/STAT for a power-up request. If the next transaction (requiring a power-up) occurs immediately, it will fail.

### 32.5.5 SW-DP registers

Access to these registers are initiated when APnDP=0

**Table 142. SW-DP registers**

A[3:2]	R/W	CTRLSEL bit of SELECT register	Register	Notes
00	Read		IDCODE	The manufacturer code is set to the default Arm code for Cortex-M0: <b>0x0BB11477</b> (identifies the SW-DP)
00	Write		ABORT	
01	Read/Write	0	DP-CTRL/STAT	Purpose is to: – request a system or debug power-up – configure the transfer operation for AP accesses – control the pushed compare and pushed verify operations. – read some status flags (overrun, power-up acknowledges)
01	Read/Write	1	WIRE CONTROL	Purpose is to configure the physical serial port protocol (like the duration of the turnaround time)
10	Read		READ RESEND	Enables recovery of the read data from a corrupted debugger transfer, without repeating the original AP transfer.
10	Write		SELECT	The purpose is to select the current access port and the active 4-words register window
11	Read/Write		READ BUFFER	This read buffer is useful because AP accesses are posted (the result of a read AP request is available on the next AP transaction). This read buffer captures data from the AP, presented as the result of a previous read, without initiating a new transaction

### 32.5.6 SW-AP registers

Access to these registers are initiated when APnDP=1

There are many AP Registers addressed as the combination of:

- The shifted value A[3:2]
- The current value of the DP SELECT register.

**Table 143. 32-bit debug port registers addressed through the shifted value A[3:2]**

Address	A[3:2] value	Description
0x0	00	Reserved, must be kept at reset value.
0x4	01	DP CTRL/STAT register. Used to: – Request a system or debug power-up – Configure the transfer operation for AP accesses – Control the pushed compare and pushed verify operations. – Read some status flags (overrun, power-up acknowledges)
0x8	10	DP SELECT register: Used to select the current access port and the active 4-words register window. – Bits 31:24: APSEL: select the current AP – Bits 23:8: reserved – Bits 7:4: APBANKSEL: select the active 4-words register window on the current AP – Bits 3:0: reserved
0xC	11	DP RDBUFF register: Used to allow the debugger to get the final result after a sequence of operations (without requesting new JTAG-DP operation)

## 32.6 Core debug

Core debug is accessed through the core debug registers. Debug access to these registers is by means of the debug access port. It consists of four registers:

**Table 144. Core debug registers**

Register	Description
DHCSR	<i>The 32-bit Debug Halting Control and Status Register</i> This provides status information about the state of the processor enable core debug halt and step the processor
DCRSR	<i>The 17-bit Debug Core Register Selector Register:</i> This selects the processor register to transfer data to or from.
DCRDR	<i>The 32-bit Debug Core Register Data Register:</i> This holds data for reading and writing registers to and from the processor selected by the DCRSR (Selector) register.
DEMCR	<i>The 32-bit Debug Exception and Monitor Control Register:</i> This provides Vector Catching and Debug Monitor Control.

These registers are not reset by a system reset. They are only reset by a power-on reset. Refer to the Cortex®-M0 TRM for further details.

To Halt on reset, it is necessary to:

- enable the bit0 (VC\_CORRESET) of the Debug and Exception Monitor Control Register
- enable the bit0 (C\_DEBUGEN) of the Debug Halting Control and Status Register

## 32.7 BPU (Break Point Unit)

The Cortex-M0 BPU implementation provides four breakpoint registers. The BPU is a subset of the Flash Patch and Breakpoint (FPB) block available in Armv7-M (Cortex-M3 & Cortex-M4).

### 32.7.1 BPU functionality

The processor breakpoints implement PC based breakpoint functionality.

Refer to the Armv6-M ARM and the Arm CoreSight Components Technical Reference Manual for more information about the BPU CoreSight identification registers, and their addresses and access types.

## 32.8 DWT (Data Watchpoint)

The Cortex-M0 DWT implementation provides two watchpoint register sets.

### 32.8.1 DWT functionality

The processor watchpoints implement both data address and PC based watchpoint functionality, a PC sampling register, and support comparator address masking, as described in the Armv6-M ARM.

### 32.8.2 DWT Program Counter Sample Register

A processor that implements the data watchpoint unit also implements the Armv6-M optional *DWT Program Counter Sample Register* (DWT\_PCSR). This register permits a debugger to periodically sample the PC without halting the processor. This provides coarse grained profiling. See the *ARMv6-M ARM* for more information.

The Cortex-M0 DWT\_PCSR records both instructions that pass their condition codes and those that fail.

## 32.9 MCU debug component (DBGMCU)

The MCU debug component helps the debugger provide support for:

- Low-power modes
- Clock control for timers, watchdog and I2C during a breakpoint

### 32.9.1 Debug support for low-power modes

To enter low-power mode, the instruction WFI or WFE must be executed.

The MCU implements several low-power modes which can either deactivate the CPU clock or reduce the power of the CPU.

The core does not allow FCLK or HCLK to be turned off during a debug session. As these are required for the debugger connection, during a debug, they must remain active. The MCU integrates special means to allow the user to debug software in low-power modes.

For this, the debugger host must first set some debug configuration registers to change the low-power mode behavior:

- In Sleep mode: FCLK and HCLK are still active. Consequently, this mode does not impose any restrictions on the standard debug features.
- In Stop/Standy mode, the DBG\_STOP bit must be previously set by the debugger.

This enables the internal RC oscillator clock to feed FCLK and HCLK in Stop mode.

For code example refer to the Appendix section [A.12.2: DBG debug in Low-power mode code example](#).

### 32.9.2 Debug support for timers, watchdog and I<sup>2</sup>C

During a breakpoint, it is necessary to choose how the counter of timers and watchdog should behave:

- They can continue to count inside a breakpoint. This is usually required when a PWM is controlling a motor, for example.
- They can stop to count inside a breakpoint. This is required for watchdog purposes.

For the I<sup>2</sup>C, the user can choose to block the SMBUS timeout during a breakpoint.

### 32.9.3 Debug MCU configuration register (DBGMCU\_CR)

This register allows the configuration of the MCU under DEBUG. This concerns:

- Low-power mode support

This DBGMCU\_CR is mapped at address 0x4001 5804.

It is asynchronously reset by the PORRESET (and not the system reset). It can be written by the debugger under system reset.

If the debugger host does not support these features, it is still possible for the user software to write to these registers.

Address: 0x40015804

Only 32-bit access supported

POR Reset: 0x0000 0000 (not reset by system reset)

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

Bits 31:3 Reserved, must be kept at reset value.

Bit 2 **DBG\_STANDBY**: Debug Standby mode

0: (FCLK=Off, HCLK=Off) The whole digital part is unpowered.

From software point of view, exiting from Standby is identical than fetching reset vector (except a few status bit indicated that the MCU is resuming from Standby)

1: (FCLK=On, HCLK=On) In this case, the digital part is not unpowered and FCLK and HCLK are provided by the internal RC oscillator which remains active. In addition, the MCU generate a system reset during Standby mode so that exiting from Standby is identical than fetching from reset

Bit 1 **DBG\_STOP**: Debug Stop mode

0: (FCLK=Off, HCLK=Off) In STOP mode, the clock controller disables all clocks (including HCLK and FCLK). When exiting from STOP mode, the clock configuration is identical to the one after RESET (CPU clocked by the 8 MHz internal RC oscillator (HSI)). Consequently, the software must reprogram the clock controller to enable the PLL, the Xtal, etc.

1: (FCLK=On, HCLK=On) In this case, when entering STOP mode, FCLK and HCLK are provided by the internal RC oscillator which remains active in STOP mode. When exiting STOP mode, the software must reprogram the clock controller to enable the PLL, the Xtal, etc. (in the same way it would do in case of **DBG\_STOP=0**)

**Note:** A side effect is that Systick, if enabled, continues to receive the clock and can consequently generate periodic interrupts and potentially wake up events.

### 32.9.4 Debug MCU APB1 freeze register (DBGMCU\_APB1\_FZ)

The DBGMCU\_APB1\_FZ register is used to configure the MCU under DEBUG. It concerns some APB peripherals:

- Timer clock counter freeze
- I2C SMBUS timeout freeze
- System window watchdog and independent watchdog counter freeze support

This DBGMCU\_APB1\_FZ is mapped at address 0x4001 5808.

The register is asynchronously reset by the POR (and not the system reset). It can be written by the debugger under system reset.

Address offset: 0x08

Only 32-bit access are supported.

Power on reset (POR): 0x0000 0000 (not reset by system reset)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	DBG_CAN_STOP	Res.	Res.	Res.	DBG_I2C1_SMBUS_TIMEOUT	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	DBG_IWDG_STOP	DBG_WWDG_STOP	DBG_RTC_STOP	Res.	DBG_TIM14_STOP	Res.	Res.	DBG_TIM7_STOP	DBG_TIM6_STOP	Res.	Res.	DBG_TIM3_STOP	DBG_TIM2_STOP
			rw	rw	rw		rw			rw	rw			rw	rw

Bits 31:26 Reserved, must be kept at reset value.

Bit 25 **DBG\_CAN\_STOP**: CAN stopped when core is halted

- 0: Same behavior as in normal mode
- 1: The CAN receive registers are frozen

Bits 22:24 Reserved, must be kept at reset value.

Bit 21 **DBG\_I2C1\_SMBUS\_TIMEOUT**: SMBUS timeout mode stopped when core is halted

- 0: Same behavior as in normal mode
- 1: The SMBUS timeout is frozen

Bits 20:13 Reserved, must be kept at reset value.

Bit 12 **DBG\_IWDG\_STOP**: Debug independent watchdog stopped when core is halted

- 0: The independent watchdog counter clock continues even if the core is halted
- 1: The independent watchdog counter clock is stopped when the core is halted

- Bit 11 **DBG\_WWDG\_STOP**: Debug window watchdog stopped when core is halted  
0: The window watchdog counter clock continues even if the core is halted  
1: The window watchdog counter clock is stopped when the core is halted
- Bit 10 **DBG\_RTC\_STOP**: Debug RTC stopped when core is halted  
0: The clock of the RTC counter is fed even if the core is halted  
1: The clock of the RTC counter is stopped when the core is halted
- Bit 9 Reserved, must be kept at reset value.
- Bit 8 **DBG\_TIM14\_STOP**: TIM14 counter stopped when core is halted  
0: The counter clock of TIM14 is fed even if the core is halted  
1: The counter clock of TIM14 is stopped when the core is halted
- Bits 7:6 Reserved, must be kept at reset value.
- Bit 5 **DBG\_TIM7\_STOP**: TIM7 counter stopped when core is halted.  
0: The counter clock of TIM7 is fed even if the core is halted  
1: The counter clock of TIM7 is stopped when the core is halted
- Bit 4 **DBG\_TIM6\_STOP**: TIM6 counter stopped when core is halted  
0: The counter clock of TIM6 is fed even if the core is halted  
1: The counter clock of TIM6 is stopped when the core is halted
- Bits 3:2 Reserved, must be kept at reset value.
- Bit 1 **DBG\_TIM3\_STOP**: TIM3 counter stopped when core is halted  
0: The counter clock of TIM3 is fed even if the core is halted  
1: The counter clock of TIM3 is stopped when the core is halted
- Bit 0 **DBG\_TIM2\_STOP**: TIM2 counter stopped when core is halted  
0: The counter clock of TIM2 is fed even if the core is halted  
1: The counter clock of TIM2 is stopped when the core is halted

### 32.9.5 Debug MCU APB2 freeze register (DBGMCU\_APB2\_FZ)

The DBGMCU\_APB2\_FZ register is used to configure the MCU under DEBUG. It concerns some APB peripherals:

- Timer clock counter freeze

This register is mapped at address 0x4001580C.

It is asynchronously reset by the POR (and not the system reset). It can be written by the debugger under system reset.

Address offset: 0x0C

Only 32-bit access is supported.

POR: 0x0000 0000 (not reset by system reset)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DBG_TIM17_STOP	DBG_TIM16_STOP	DBG_TIM15_STOP
													rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	DBG_TIM1_STOP	Res.	Res.	Res.								
				rw											

Bits 31:19 Reserved, must be kept at reset value.

Bit 18 **DBG\_TIM17\_STOP**: TIM17 counter stopped when core is halted

- 0: The counter clock of TIM17 is fed even if the core is halted
- 1: The counter clock of TIM17 is stopped when the core is halted

Bit 17 **DBG\_TIM16\_STOP**: TIM16 counter stopped when core is halted

- 0: The counter clock of TIM16 is fed even if the core is halted
- 1: The counter clock of TIM16 is stopped when the core is halted

Bit 16 **DBG\_TIM15\_STOP**: TIM15 counter stopped when core is halted

- 0: The counter clock of TIM15 is fed even if the core is halted
- 1: The counter clock of TIM15 is stopped when the core is halted

Bits 15:12 Reserved, must be kept at reset value.

Bit 11 **DBG\_TIM1\_STOP**: TIM1 counter stopped when core is halted

- 0: The counter clock of TIM 1 is fed even if the core is halted
- 1: The counter clock of TIM 1 is stopped when the core is halted

Bits 0:10 Reserved, must be kept at reset value.

### 32.9.6 DBG register map

The following table summarizes the Debug registers.

**Table 145. DBG register map and reset values**

1. The reset value is product dependent. For more information, refer to [Section 32.4.1: MCU device ID code](#).



## 33 Device electronic signature

The device electronic signature is stored in the System memory area of the Flash memory module, and can be read using the debug interface or by the CPU. It contains factory-programmed identification and calibration data that allow the user firmware or other external devices to automatically match to the characteristics of the STM32F0xx microcontroller.

### 33.1 Unique device ID register (96 bits)

The unique device identifier is ideally suited:

- for use as serial numbers (for example USB string serial numbers or other end applications)
- for use as part of the security keys in order to increase the security of code in Flash memory while using and combining this unique ID with software cryptographic primitives and protocols before programming the internal Flash memory
- to activate secure boot processes, etc.

The 96-bit unique device identifier provides a reference number which is unique for any device and in any context. These bits cannot be altered by the user.

Base address: 0xFFFF F7AC

Address offset: 0x00

Read only = 0xXXXX XXXX where X is factory-programmed

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

Bits 31:0 **UID[31:0]**: X and Y coordinates on the wafer expressed in BCD format

Address offset: 0x04

Read only = 0xXXXX XXXX where X is factory-programmed

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UID[63:48]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
UID[47:32]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:8 **UID[63:40]: LOT\_NUM[23:0]**  
Lot number (ASCII encoded)

Bits 7:0 **UID[39:32]: WAF\_NUM[7:0]**  
Wafer number (8-bit unsigned number)

Address offset: 0x08

Read only = 0xXXXX XXXX where X is factory-programmed

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UID[95:80]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
UID[79:64]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **UID[95:64]: LOT\_NUM[55:24]**  
Lot number (ASCII encoded)

### 33.2 Flash memory size data register

Base address: 0x1FFF F7CC

Address offset: 0x00

Read only = 0xXXXX where X is factory-programmed

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FLASH_SIZE															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:0 **FLASH\_SIZE[15:0]: Flash memory size**

This bitfield indicates the size of the device Flash memory expressed in Kbytes.

As an example, 0x040 corresponds to 64 Kbytes.

## Appendix A Code examples

### A.1 Introduction

This appendix shows the code examples of the sequence described in this document.

These code examples are extracted from the STM32F0xx Snippet firmware package **STM32SnippetsF0** available on [www.st.com](http://www.st.com).

These code examples used the peripheral bit and register description from the CMSIS header file (stm32f0xx.h).

Code lines starting with // should be uncommented if the given register has been modified before.

### A.2 Flash operation code examples

#### A.2.1 Flash memory unlocking sequence

```
/* (1) Wait till no operation is on going */
/* (2) Check that the Flash is unlocked */
/* (3) Perform unlock sequence */
while ((FLASH->SR & FLASH_SR_BSY) != 0) /* (1) */
{
    /* For robust implementation, add here time-out management */
}
if ((FLASH->CR & FLASH_CR_LOCK) != 0) /* (2) */
{
    FLASH->KEYR = FLASH_FKEY1; /* (3) */
    FLASH->KEYR = FLASH_FKEY2;
}
```

#### A.2.2 Main Flash programming sequence

```
/* (1) Set the PG bit in the FLASH_CR register to enable programming */
/* (2) Perform the data write (half-word) at the desired address */
/* (3) Wait until the BSY bit is reset in the FLASH_SR register */
/* (4) Check the EOP flag in the FLASH_SR register */
/* (5) clear it by software by writing it at 1 */
/* (6) Reset the PG Bit to disable programming */
FLASH->CR |= FLASH_CR_PG; /* (1) */
*(__IO uint16_t*)flash_addr = data; /* (2) */
while ((FLASH->SR & FLASH_SR_BSY) != 0) /* (3) */
{
    /* For robust implementation, add here time-out management */
}
```

```

if ((FLASH->SR & FLASH_SR_EOP) != 0) /* (4) */
{
    FLASH->SR = FLASH_SR_EOP; /* (5) */
}
else
{
    /* Manage the error cases */
}
FLASH->CR &= ~FLASH_CR_PG; /* (6) */

```

### A.2.3 Page erase sequence

```

/* (1) Set the PER bit in the FLASH_CR register to enable page erasing */
/* (2) Program the FLASH_AR register to select a page to erase */
/* (3) Set the STRT bit in the FLASH_CR register to start the erasing */
/* (4) Wait until the BSY bit is reset in the FLASH_SR register */
/* (5) Check the EOP flag in the FLASH_SR register */
/* (6) Clear EOP flag by software by writing EOP at 1 */
/* (7) Reset the PER Bit to disable the page erase */
FLASH->CR |= FLASH_CR_PER; /* (1) */
FLASH->AR = page_addr; /* (2) */
FLASH->CR |= FLASH_CR_STRT; /* (3) */
while ((FLASH->SR & FLASH_SR_BSY) != 0) /* (4) */
{
    /* For robust implementation, add here time-out management */
}
if ((FLASH->SR & FLASH_SR_EOP) != 0) /* (5) */
{
    FLASH->SR = FLASH_SR_EOP; /* (6) */
}
else
{
    /* Manage the error cases */
}
FLASH->CR &= ~FLASH_CR_PER; /* (7) */

```

## A.2.4 Mass erase sequence

```

/* (1) Set the MER bit in the FLASH_CR register to enable mass erasing */
/* (2) Set the STRT bit in the FLASH_CR register to start the erasing */
/* (3) Wait until the BSY bit is reset in the FLASH_SR register */
/* (4) Check the EOP flag in the FLASH_SR register */
/* (5) Clear EOP flag by software by writing EOP at 1 */
/* (6) Reset the PER Bit to disable the mass erase */
FLASH->CR |= FLASH_CR_MER; /* (1) */
FLASH->CR |= FLASH_CR_STRT; /* (2) */
while ((FLASH->SR & FLASH_SR_BSY) != 0) /* (3) */
{
    /* For robust implementation, add here time-out management */
}

if ((FLASH->SR & FLASH_SR_EOP) != 0) /* (4) */
{
    FLASH->SR = FLASH_SR_EOP; /* (5) */
}
else
{
    /* Manage the error cases */
}
FLASH->CR &= ~FLASH_CR_MER; /* (6) */

```

## A.2.5 Option byte unlocking sequence

```

/* (1) Wait till no operation is on going */
/* (2) Check that the Flash is unlocked */
/* (3) Perform unlock sequence for Flash */
/* (4) Check that the Option Bytes are unlocked */
/* (5) Perform unlock sequence for Option Bytes */
while ((FLASH->SR & FLASH_SR_BSY) != 0) /* (1) */
{
    /* For robust implementation, add here time-out management */
}
if ((FLASH->CR & FLASH_CR_LOCK) != 0) /* (2) */
{
    FLASH->KEYR = FLASH_FKEY1; /* (3) */
    FLASH->KEYR = FLASH_FKEY2;
}
if ((FLASH->CR & FLASH_CR_OPTWRE) == 0) /* (4) */
{
    FLASH->OPTKEYR = FLASH_OPTKEY1; /* (5) */
    FLASH->OPTKEYR = FLASH_OPTKEY2;
}

```

## A.2.6 Option byte programming sequence

```

/* (1) Set the PG bit in the FLASH_CR register to enable programming */
/* (2) Perform the data write */
/* (3) Wait until the BSY bit is reset in the FLASH_SR register */
/* (4) Check the EOP flag in the FLASH_SR register */
/* (5) Clear the EOP flag by software by writing it at 1 */
/* (6) Reset the PG Bit to disable programming */
FLASH->CR |= FLASH_CR_OPTPG; /* (1) */
*opt_addr = data; /* (2) */
while ((FLASH->SR & FLASH_SR_BSY) != 0) /* (3) */
{
    /* For robust implementation, add here time-out management */
}
if ((FLASH->SR & FLASH_SR_EOP) != 0) /* (4) */
{
    FLASH->SR = FLASH_SR_EOP; /* (5) */
}
else
{
    /* Manage the error cases */
}
FLASH->CR &= ~FLASH_CR_OPTPG; /* (6) */

```

## A.2.7 Option byte erasing sequence

```

/* (1) Set the OPTER bit in the FLASH_CR register to enable option byte
   erasing */
/* (2) Set the STRT bit in the FLASH_CR register to start the erasing */
/* (3) Wait until the BSY bit is reset in the FLASH_SR register */
/* (4) Check the EOP flag in the FLASH_SR register */
/* (5) Clear EOP flag by software by writing EOP at 1 */
/* (6) Reset the PER Bit to disable the page erase */
FLASH->CR |= FLASH_CR_OPTER; /* (1) */
FLASH->CR |= FLASH_CR_STRT; /* (2) */
while ((FLASH->SR & FLASH_SR_BSY) != 0) /* (3) */
{
    /* For robust implementation, add here time-out management */
}
if ((FLASH->SR & FLASH_SR_EOP) != 0) /* (4) */
{
    FLASH->SR = FLASH_SR_EOP; /* (5) */
}
else
{
    /* Manage the error cases */
}
FLASH->CR &= ~FLASH_CR_OPTER; /* (6) */

```

## A.3 Clock controller

### A.3.1 HSE start sequence code example

```
/***
 * Description: This function enables the interrupt on HSE ready,
 *               and start the HSE as external clock.
 */
INLINE void StartHSE(void)
{
    /* Configure NVIC for RCC */
    /* (1) Enable Interrupt on RCC */
    /* (2) Set priority for RCC */
    NVIC_EnableIRQ(RCC_CRS IRQn); /* (1)*/
    NVIC_SetPriority(RCC_CRS IRQn, 0); /* (2) */

    /* (1) Enable interrupt on HSE ready */
    /* (2) Enable the CSS
       Enable the HSE and set HSEBYP to use the external clock
       instead of an oscillator
       Enable HSE */
    /* Note : the clock is switched to HSE in the RCC_CRS_IRQHandler ISR */
    RCC->CIR |= RCC_CIR_HSERDYIE; /* (1) */
    RCC->CR |= RCC_CR_CSSON | RCC_CR_HSEBYP | RCC_CR_HSEON; /* (2) */
}

/***
 * Description: This function handles RCC interrupt request
 *               and switch the system clock to HSE.
 */
void RCC_CRS_IRQHandler(void)
{
    /* (1) Check the flag HSE ready */
    /* (2) Clear the flag HSE ready */
    /* (3) Switch the system clock to HSE */

    if ((RCC->CIR & RCC_CIR_HSERDYF) != 0) /* (1) */
    {
        RCC->CIR |= RCC_CIR_HSERDYC; /* (2) */
        RCC->CFGR = ((RCC->CFGR & (~RCC_CFGR_SW)) | RCC_CFGR_SW_0); /* (3) */
    }
    else
    {
        /* Report an error */
    }
}
```

### A.3.2 PLL configuration modification code example

```

/* (1) Test if PLL is used as System clock */
/* (2) Select HSI as system clock */
/* (3) Wait for HSI switched */
/* (4) Disable the PLL */
/* (5) Wait until PLLRDY is cleared */
/* (6) Set the PLL multiplier to 6 */
/* (7) Enable the PLL */
/* (8) Wait until PLLRDY is set */
/* (9) Select PLL as system clock */
/* (10) Wait until the PLL is switched on */
if ((RCC->CFGR & RCC_CFGR_SWS) == RCC_CFGR_SWS_PLL) /* (1) */
{
    RCC->CFGR &= (uint32_t) (~RCC_CFGR_SW); /* (2) */
    while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_HSI) /* (3) */
    {
        /* For robust implementation, add here time-out management */
    }
}
RCC->CR &= (uint32_t) (~RCC_CR_PLLON); /* (4) */
while((RCC->CR & RCC_CR_PLLRDY) != 0) /* (5) */
{
    /* For robust implementation, add here time-out management */
}
RCC->CFGR = RCC->CFGR & (~RCC_CFGR_PLLMUL) | (RCC_CFGR_PLLMUL6); /* (6) */
RCC->CR |= RCC_CR_PLLON; /* (7) */
while((RCC->CR & RCC_CR_PLLRDY) == 0) /* (8) */
{
    /* For robust implementation, add here time-out management */
}
RCC->CFGR |= (uint32_t) (RCC_CFGR_SW_PLL); /* (9) */
while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_PLL) /* (10) */
{
    /* For robust implementation, add here time-out management */
}

```

### A.3.3 MCO selection code example

```

/* Select system clock to be output on the MCO without prescaler */
RCC->CFGR |= RCC_CFGR_MCO_SYSCLK;

```

### A.3.4 Clock measurement configuration with TIM14 code example

```
/**  
 * Description: This function configures the TIM14 as input capture  
 * and enables the interrupt on TIM14  
 */  
_INLINE void ConfigureTIM14asInputCapture(void)  
{  
    /* (1) Enable the peripheral clock of Timer 14 */  
    /* (2) Select the active input TI1, Program the input filter, and prescaler  
     */  
    /* (3) Enable interrupt on Capture/Compare */  
    RCC->APB1ENR |= RCC_APB1ENR_TIM14EN; /* (1) */  
    TIM14->CCMR1 |= TIM_CCMR1_IC1F_0 | TIM_CCMR1_IC1F_1 \  
        | TIM_CCMR1_CC1S_0 | TIM_CCMR1_IC1PSC_1; /* (2) */  
    TIM14->DIER |= TIM_DIER_CC1IE; /* (3) */  
  
    /* Configure NVIC for TIM14 */  
    /* (4) Enable Interrupt on TIM14 */  
    /* (5) Set priority for TIM14 */  
    NVIC_EnableIRQ(TIM14_IRQn); /* (4) */  
    NVIC_SetPriority(TIM14_IRQn, 0); /* (5) */  
  
    /* (6) Select HSE/32 as input on TI1 */  
    /* (7) Enable counter */  
    /* (8) Enable capture */  
    TIM14->OR |= TIM14_OR_TI1_RMP_1; /* (6) */  
    TIM14->CR1 |= TIM_CR1_CEN; /* (7) */  
    TIM14->CCER |= TIM_CCER_CC1E; /* (8) */  
}
```

*Note:* The measurement is done in the TIM14 interrupt subroutine.

## A.4 GPIO

### A.4.1 Lock sequence code example

```
/***
 * Description: This function locks the targeted pins of Port A
 * configuration
 * This function can be easily modified to lock Port B
 * Parameter: lock contains the port pin mask to be locked
 */
void LockGPIOA(uint16_t lock)
{
    /* (1) Write LCKK bit to 1 and set the pin bits to lock */
    /* (2) Write LCKK bit to 0 and set the pin bits to lock */
    /* (3) Write LCKK bit to 1 and set the pin bits to lock */
    /* (4) Read the Lock register */
    /* (5) Check the Lock register (optionnal) */
    GPIOA->LCKR = GPIO_LCKR_LCKK + lock; /* (1) */
    GPIOA->LCKR = lock; /* (2) */
    GPIOA->LCKR = GPIO_LCKR_LCKK + lock; /* (3) */
    GPIOA->LCKR; /* (4) */
    if ((GPIOA->LCKR & GPIO_LCKR_LCKK) == 0) /* (5) */
    {
        /* Manage an error */
    }
}
```

### A.4.2 Alternate function selection sequence code example

```
/* This sequence select AF2 for GPIOA4, 8 and 9. This can be easily adapted
with another port by changing all GPIOA references by another GPIO port,
and the alternate function number can be changed by replacing 0x04 or
0x02 for
each pin by the targeted alternate function in the 2 last code lines. */
/* (1) Enable the peripheral clock of GPIOA */
/* (2) Select alternate function mode on GPIOA pin 4, 8 and 9 */
/* (3) Select AF4 on PA4 in AFRL for TIM14_CH1 */
/* (4) Select AF2 on PA8 and PA9 in AFRH for TIM1_CH1 and TIM1_CH2 */
RCC->AHBENR |= RCC_AHBENR_GPIOAEN; /* (1) */
GPIOA->MODER = (GPIOA->MODER & ~GPIO_MODER_MODER4 | GPIO_MODER_MODER8
| GPIO_MODER_MODER9)) | GPIO_MODER_MODER4_1
| GPIO_MODER_MODER8_1 | GPIO_MODER_MODER9_1; /* (2) */
GPIOA->AFR[0] |= 0x04 << GPIO_AFRL_AFRL4_Pos; /* (3) */
GPIOA->AFR[1] |= (0x02 << GPIO_AFRL_AFRH8_Pos) | (0x02 <<
GPIO_AFRL_AFRH9_Pos); /* (4) */
```

### A.4.3 Analog GPIO configuration code example

```

/* (1) Enable the peripheral clock of GPIOA, GPIOB and GPIOC */
/* (2) Select analog mode for PA1 */
/* (3) Select analog mode for PB1 */
/* (4) Select analog mode for PC0 */
RCC->AHBENR |= RCC_AHBENR_GPIOAEN | RCC_AHBENR_GPIOBEN
    | RCC_AHBENR_GPIOCEN; /* (1) */
GPIOA->MODER |= GPIO_MODER_MODER1; /* (2) */
GPIOB->MODER |= GPIO_MODER_MODER1; /* (3) */
GPIOC->MODER |= GPIO_MODER_MODER0; /* (4) */

```

## A.5 DMA

### A.5.1 DMA channel configuration sequence code example

```

/* The following example is given for the ADC. It can be easily ported on
any peripheral supporting DMA transfer taking of the associated channel
to the peripheral, this must check in the datasheet. */
/* (1) Enable the peripheral clock on DMA */
/* (2) Enable DMA transfer on ADC */
/* (3) Configure the peripheral data register address */
/* (4) Configure the memory address */
/* (5) Configure the number of DMA transfer to be performs on channel 1 */
/* (6) Configure increment, size and interrupts */
/* (7) Enable DMA Channel 1 */
RCC->AHBENR |= RCC_AHBENR_DMA1EN; /* (1) */
ADC1->CFGGR1 |= ADC_CFGGR1_DMAEN; /* (2) */
DMA1_Channel1->CPAR = (uint32_t) (&(ADC1->DR)); /* (3) */
DMA1_Channel1->CMAR = (uint32_t) (ADC_array); /* (4) */
DMA1_Channel1->CNDTR = 3; /* (5) */
DMA1_Channel1->CCR |= DMA_CCR_MINC | DMA_CCR_MSIZE_0 | DMA_CCR_PSIZE_0
    | DMA_CCR_TEIE | DMA_CCR_TCIE; /* (6) */
DMA1_Channel1->CCR |= DMA_CCR_EN; /* (7) */
/* Configure NVIC for DMA */
/* (1) Enable Interrupt on DMA Channel 1 */
/* (2) Set priority for DMA Channel 1 */
NVIC_EnableIRQ(DMA1_Channel1_IRQn); /* (1) */
NVIC_SetPriority(DMA1_Channel1_IRQn, 0); /* (2) */

```

## A.6 Interrupts and event

### A.6.1 NVIC initialization example

```
/* (1) Enable Interrupt on ADC */
/* (2) Set priority for ADC to 2*/
NVIC_EnableIRQ(ADC1_COMP IRQn); /* (1) */
NVIC_SetPriority(ADC1_COMP IRQn,2); /* (2) */
```

### A.6.2 External interrupt selection code example

```
/* (1) Enable the peripheral clock of GPIOA */
/* (2) Select Port A for pin 0 external interrupt by writing 0000 in
EXTI0 (reset value)*/
/* (3) Configure the corresponding mask bit in the EXTI_IMR register */
/* (4) Configure the Trigger Selection bits of the Interrupt line on
rising edge*/
/* (5) Configure the Trigger Selection bits of the Interrupt line on
falling edge*/
RCC->AHBENR |= RCC_AHBENR_GPIOAEN; /* (1) */
//SYSCFG->EXTICR[1] &= (uint16_t)~SYSCFG_EXTICR1_EXTI0_PA; /* (2) */
EXTI->IMR = 0x0001; /* (3) */
EXTI->RTSR = 0x0001; /* (4) */
EXTI->FTSR = 0x0001; /* (5) */
/* Configure NVIC for External Interrupt */
/* (1) Enable Interrupt on EXTI0_1 */
/* (2) Set priority for EXTI0_1 */
NVIC_EnableIRQ(EXTI0_1 IRQn); /* (1) */
NVIC_SetPriority(EXTI0_1 IRQn,0); /* (2) */
```

## A.7 ADC

### A.7.1 ADC calibration code example

```
/* (1) Ensure that ADEN = 0 */
/* (2) Clear ADEN by setting ADDIS*/
/* (3) Clear DMAEN */
/* (4) Launch the calibration by setting ADCAL */
/* (5) Wait until ADCAL=0 */
if ((ADC1->CR & ADC_CR_ADEN) != 0) /* (1) */
{
    ADC1->CR |= ADC_CR_ADDIS; /* (2) */
}
while ((ADC1->CR & ADC_CR_ADEN) != 0)
{
/* For robust implementation, add here time-out management */
}
ADC1->CFGREG1 &= ~ADC_CFGREG1_DMAEN; /* (3) */
ADC1->CR |= ADC_CR_ADCAL; /* (4) */
while ((ADC1->CR & ADC_CR_ADCAL) != 0) /* (5) */
{
    /* For robust implementation, add here time-out management */
}
```

### A.7.2 ADC enable sequence code example

```
/* (1) Ensure that ADRDY = 0 */
/* (2) Clear ADRDY */
/* (3) Enable the ADC */
/* (4) Wait until ADC ready */
if ((ADC1->ISR & ADC_ISR_ADRDY) != 0) /* (1) */
{
    ADC1->ISR |= ADC_CR_ADRDY; /* (2) */
}
ADC1->CR |= ADC_CR_ADEN; /* (3) */
while ((ADC1->ISR & ADC_ISR_ADRDY) == 0) /* (4) */
{
    /* For robust implementation, add here time-out management */
}
```

### A.7.3 ADC disable sequence code example

```

/* (1) Stop any ongoing conversion */
/* (2) Wait until ADSTP is reset by hardware i.e. conversion is stopped */
/* (3) Disable the ADC */
/* (4) Wait until the ADC is fully disabled */
ADC1->CR |= ADC_CR_ADSTP; /* (1) */
while ((ADC1->CR & ADC_CR_ADSTP) != 0) /* (2) */
{
    /* For robust implementation, add here time-out management */
}
ADC1->CR |= ADC_CR_ADDIS; /* (3) */
while ((ADC1->CR & ADC_CR_ADEN) != 0) /* (4) */
{
    /* For robust implementation, add here time-out management */
}

```

### A.7.4 ADC clock selection code example

```

/* This code selects the HSI14 as clock source. */
/* (1) Enable the peripheral clock of the ADC */
/* (2) Start HSI14 RC oscillator */
/* (3) Wait HSI14 is ready */
/* (4) Select HSI14 by writing 00 in CKMODE (reset value) */
RCC->APB2ENR |= RCC_APB2ENR_ADC1EN; /* (1) */
RCC->CR2 |= RCC_CR2_HSI14ON; /* (2) */
while ((RCC->CR2 & RCC_CR2_HSI14RDY) == 0) /* (3) */
{
    /* For robust implementation, add here time-out management */
}
//ADC1->CFGR2 &= (~ADC_CFGR2_CKMODE); /* (4) */

```

## A.7.5 Single conversion sequence code example - Software trigger

```

/* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
/* (2) Select CHSEL0, CHSEL9, CHSEL10 and CHSEL17 for VRefInt */
/* (3) Select a sampling mode of 111 i.e. 239.5 ADC clk to be greater
   than 17.1us */
/* (4) Wake-up the VREFINT (only for VBAT, Temp sensor and VRefInt) */
//ADC1->CFGGR2 &= ~ADC_CFGGR2_CKMODE; /* (1) */
ADC1->CHSELR = ADC_CHSELR_CHSEL0 | ADC_CHSELR_CHSEL9
               | ADC_CHSELR_CHSEL10 | ADC_CHSELR_CHSEL17; /* (2) */
ADC1->SMPR |= ADC_SMPR_SMP_0 | ADC_SMPR_SMP_1 | ADC_SMPR_SMP_2; /* (3) */
ADC->CCR |= ADC_CCR_VREFEN; /* (4) */
while (1)
{
    /* Performs the AD conversion */
    ADC1->CR |= ADC_CR_ADSTART; /* Start the ADC conversion */
    for (i=0; i < 4; i++)
    {
        while ((ADC1->ISR & ADC_ISR_EOC) == 0) /* Wait end of conversion */
        {
            /* For robust implementation, add here time-out management */
        }
        ADC_Result[i] = ADC1->DR; /* Store the ADC conversion result */
    }
    ADC1->CFGGR1 ^= ADC_CFGGR1_SCANDIR; /* Toggle the scan direction */
}

```

## A.7.6 Continuous conversion sequence code example - Software trigger

```

/* This code example configures the AD conversion in continuous mode and in
   backward scan. It also enable the interrupts. */
/* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
/* (2) Select the continuous mode and scanning direction */
/* (3) Select CHSEL1, CHSEL9, CHSEL10 and CHSEL17 */
/* (4) Select a sampling mode of 111 i.e. 239.5 ADC clk to be greater than
   17.1us */
/* (5) Enable interrupts on EOC, EOSEQ and overrun */
/* (6) Wake-up the VREFINT (only for VBAT, Temp sensor and VRefInt) */
//ADC1->CFGGR2 &= ~ADC_CFGGR2_CKMODE; /* (1) */
ADC1->CFGGR1 |= ADC_CFGGR1_CONT | ADC_CFGGR1_SCANDIR; /* (2) */
ADC1->CHSELR = ADC_CHSELR_CHSEL1 | ADC_CHSELR_CHSEL9
               | ADC_CHSELR_CHSEL10 | ADC_CHSELR_CHSEL17; /* (3) */
ADC1->SMPR |= ADC_SMPR_SMP_0 | ADC_SMPR_SMP_1 | ADC_SMPR_SMP_2; /* (4) */
ADC1->IER = ADC_IER_EOCIE | ADC_IER_EOSEQIE | ADC_IER_OVRIE; /* (5) */
ADC->CCR |= ADC_CCR_VREFEN; /* (6) */

/* Configure NVIC for ADC */
/* (7) Enable Interrupt on ADC */
/* (8) Set priority for ADC */
NVIC_EnableIRQ(ADC1_COMP IRQn); /* (7) */
NVIC_SetPriority(ADC1_COMP IRQn, 0); /* (8) */

```

## A.7.7 Single conversion sequence code example - Hardware trigger

```

/* Configure the ADC, the ADC and its clock having previously been
   enabled. */
/* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
/* (2) Select the external trigger on falling edge and external trigger on
      TIM15_TRGO */
/* (3) Select CHSEL0, 1, 2 and 3 */
//ADC1->CFGGR2 &= ~ADC_CFGGR2_CKMODE; /* (1) */
ADC1->CFGGR1 |= ADC_CFGGR1_EXTEN_0 | ADC_CFGGR1_EXTSEL_2; /* (2) */
ADC1->CHSELR = ADC_CHSELR_CHSEL0 | ADC_CHSELR_CHSEL1
               | ADC_CHSELR_CHSEL2 | ADC_CHSELR_CHSEL3; /* (3) */

```

## A.7.8 Continuous conversion sequence code example - Hardware trigger

```

/* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
/* (2) Select the external trigger on TIM15_TRGO (EXTSEL = 100), falling
   edge (EXTEN = 10), the continuous mode (CONT = 1) */
/* (3) Select CHSEL0/1/2/3 */
/* (4) Enable interrupts on EOC, EOSEQ and overrun */
//ADC1->CFGGR2 &= ~ADC_CFGGR2_CKMODE; /* (1) */
ADC1->CFGGR1 |= ADC_CFGGR1_EXTEN_1 | ADC_CFGGR1_EXTSEL_2
                 | ADC_CFGGR1_CONT; /* (2) */
ADC1->CHSELR = ADC_CHSELR_CHSEL0 | ADC_CHSELR_CHSEL1
                | ADC_CHSELR_CHSEL2 | ADC_CHSELR_CHSEL3; /* (3) */
ADC1->IER = ADC_IER_EOCIE | ADC_IER_EOSEQIE | ADC_IER_OVRIE; /* (4) */

/* Configure NVIC for ADC */
/* (1) Enable Interrupt on ADC */
/* (2) Set priority for ADC */
NVIC_EnableIRQ(ADC1_COMP IRQn); /* (1) */
NVIC_SetPriority(ADC1_COMP IRQn, 0); /* (2) */

```

### A.7.9 DMA one shot mode sequence code example

```

/* (1) Enable the peripheral clock on DMA */
/* (2) Enable DMA transfer on ADC - DMACFG is kept at 0
   for one shot mode */
/* (3) Configure the peripheral data register address */
/* (4) Configure the memory address */
/* (5) Configure the number of DMA tranfer to be performs
   on DMA channel 1 */
/* (6) Configure increment, size and interrupts */
/* (7) Enable DMA Channel 1 */
RCC->AHBENR |= RCC_AHBENR_DMA1EN; /* (1) */
ADC1->CFGGR1 |= ADC_CFGGR1_DMAEN; /* (2) */
DMA1_Channel1->CPAR = (uint32_t) (&(ADC1->DR)); /* (3) */
DMA1_Channel1->CMAR = (uint32_t) (ADC_array); /* (4) */
DMA1_Channel1->CNDTR = NUMBER_OF_ADC_CHANNEL; /* (5) */
DMA1_Channel1->CCR |= DMA_CCR_MINC | DMA_CCR_MSIZE_0 | DMA_CCR_PSIZE_0
                      | DMA_CCR_TEIE | DMA_CCR_TCIE ; /* (6) */
DMA1_Channel1->CCR |= DMA_CCR_EN; /* (7) */

```

### A.7.10 DMA circular mode sequence code example

```

/* (1) Enable the peripheral clock on DMA */
/* (2) Enable DMA transfer on ADC and circular mode */
/* (3) Configure the peripheral data register address */
/* (4) Configure the memory address */
/* (5) Configure the number of DMA tranfer to be performs
   on DMA channel 1 */
/* (6) Configure increment, size, interrupts and circular mode */
/* (7) Enable DMA Channel 1 */
RCC->AHBENR |= RCC_AHBENR_DMA1EN; /* (1) */
ADC1->CFGGR1 |= ADC_CFGGR1_DMAEN | ADC_CFGGR1_DMACFG; /* (2) */
DMA1_Channel1->CPAR = (uint32_t) (&(ADC1->DR)); /* (3) */
DMA1_Channel1->CMAR = (uint32_t) (ADC_array); /* (4) */
DMA1_Channel1->CNDTR = NUMBER_OF_ADC_CHANNEL; /* (5) */
DMA1_Channel1->CCR |= DMA_CCR_MINC | DMA_CCR_MSIZE_0 | DMA_CCR_PSIZE_0
                      | DMA_CCR_TEIE | DMA_CCR_CIRC; /* (6) */
DMA1_Channel1->CCR |= DMA_CCR_EN; /* (7) */

```

### A.7.11 Wait mode sequence code example

```

/* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
/* (2) Select the continuous mode and the wait mode */
/* (3) Select CHSEL1/2/3 */
ADC1->CFGGR2 &= ~ADC_CFGGR2_CKMODE; /* (1) */
ADC1->CFGGR1 |= ADC_CFGGR1_CONT | ADC_CFGGR1_WAIT; /* (2) */
ADC1->CHSELR = ADC_CHSELR_CHSEL1 | ADC_CHSELR_CHSEL2
                | ADC_CHSELR_CHSEL3; /* (3) */
ADC1->CR |= ADC_CR_ADSTART; /* start the ADC conversions */

```

### A.7.12 Auto Off and no wait mode sequence code example

```

/* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
/* (2) Select the external trigger on TIM15_TRGO and rising edge
   and auto off */
/* (3) Select CHSEL1/2/3/4 */
/* (4) Enable interrupts on EOC, EOSEQ and overrun */
ADC1->CFGGR2 &= ~ADC_CFGGR2_CKMODE; /* (1) */
ADC1->CFGGR1 |= ADC_CFGGR1_EXTEN_1 | ADC_CFGGR1_EXTSEL_2
    | ADC_CFGGR1_AUTOFF; /* (2) */
ADC1->CHSELR = ADC_CHSELR_CHSEL1 | ADC_CHSELR_CHSEL2
    | ADC_CHSELR_CHSEL3 | ADC_CHSELR_CHSEL4; /* (3) */
ADC1->IER = ADC_IER_EOCIE | ADC_IER_EOSEQIE | ADC_IER_OVRIE; /* (4) */

```

### A.7.13 Auto Off and wait mode sequence code example

```

/* (1) Select HSI14 by writing 00 in CKMODE (reset value) */
/* (2) Select the external trigger on TIM15_TRGO and falling edge,
   the continuous mode, scanning direction and auto off */
/* (3) Select CHSEL1, CHSEL9, CHSEL10 and CHSEL17 */
/* (4) Enable interrupts on EOC, EOSEQ and overrun */
ADC1->CFGGR2 &= ~ADC_CFGGR2_CKMODE; /* (1) */
ADC1->CFGGR1 |= ADC_CFGGR1_EXTEN_0 | ADC_CFGGR1_EXTSEL_2
    | ADC_CFGGR1_SCANDIR | ADC_CFGGR1_AUTOFF; /* (2) */
ADC1->CHSELR = ADC_CHSELR_CHSEL1 | ADC_CHSELR_CHSEL2
    | ADC_CHSELR_CHSEL3 | ADC_CHSELR_CHSEL4; /* (3) */
ADC1->IER = ADC_IER_EOCIE | ADC_IER_EOSEQIE | ADC_IER_OVRIE; /* (4) */

```

### A.7.14 Analog watchdog code example

```

/* (1) Select the continuous mode
   and configure the Analog watchdog to monitor only CH17 */
/* (2) Define analog watchdog range : 16b-MSW is the high limit
   and 16b-LSW is the low limit */
/* (3) Enable interrupt on Analog Watchdog */
ADC1->CFGGR1 |= ADC_CFGGR1_CONT
    | (17 << 26) | ADC_CFGGR1_AWDEN | ADC_CFGGR1_AWDSSGL; /* (1) */
ADC1->TR = (vrefint_high << 16) + vrefint_low; /* (2) */
ADC1->IER = ADC_IER_AWDIE; /* (3) */

```

### A.7.15 Temperature configuration code example

```
/* (1) Select CHSEL16 for temperature sensor */
/* (2) Select a sampling mode of 111 i.e. 239.5 ADC clk to be greater than
   17.1us */
/* (3) Wake-up the Temperature sensor (only for VBAT, Temp sensor and
   VRefInt) */
ADC1->CHSELR = ADC_CHSELR_CHSEL16; /* (1) */
ADC1->SMPR |= ADC_SMPR_SMP_0 | ADC_SMPR_SMP_1 | ADC_SMPR_SMP_2; /* (2) */
ADC->CCR |= ADC_CCR_TSEN; /* (3) */
```

### A.7.16 Temperature computation code example

```
/* Temperature sensor calibration value address */
#define TEMP110_CAL_ADDR ((uint16_t*) ((uint32_t) 0x1FFFF7C2))
#define TEMP30_CAL_ADDR ((uint16_t*) ((uint32_t) 0x1FFFF7B8))
#define VDD_CALIB ((uint16_t) (330))
#define VDD_APPLI ((uint16_t) (300))
int32_t temperature; /* will contain the temperature in degrees Celsius */
temperature = (((int32_t) ADC1->DR * VDD_APPLI / VDD_CALIB)
               - (int32_t) *TEMP30_CAL_ADDR );
temperature = temperature * (int32_t)(110 - 30);
temperature = temperature / (int32_t)(*TEMP110_CAL_ADDR
                                     - *TEMP30_CAL_ADDR);
temperature = temperature + 30;
```

## A.8 DAC

### A.8.1 Independent trigger without wave generation code example

```
/* (1) Enable the peripheral clock of the DAC */
/* (2) Enable DMA transfer on DAC ch1 and ch2,
       enable interrupt on DMA underrun DAC ch1 and ch2,
       enable the DAC ch1 and ch2,
       select TIM6 as trigger by keeping 000 in TSEL1
       select TIM7 as trigger by writing 010 in TSEL2 */
RCC->APB1ENR |= RCC_APB1ENR_DACEN; /* (1) */
DAC->CR |= DAC_CR_TSEL2_1 | DAC_CR_DMAUDRIE2 | DAC_CR_DMAEN2
           | DAC_CR_TEN2 | DAC_CR_EN2
           | DAC_CR_DMAUDRIE1 | DAC_CR_DMAEN1 | DAC_CR_BOFF1
           | DAC_CR_TEN1 | DAC_CR_EN1; /* (2) */
DAC->DHR12R1 = DAC_OUT1_VALUE; /* Initialize the DAC value on ch1 */
DAC->DHR12R2 = DAC_OUT2_VALUE; /* Initialize the DAC value on ch2 */
```

## A.8.2 Independent trigger with single LFSR generation code example

```

/* (1) Enable the peripheral clock of the DAC */
/* (2) Configure WAVEx at 01 and LFSR mask amplitude (MAMPx) at 1000 for
   a 511-bits amplitude,
   enable the DAC ch1 and ch2,
   disable buffer on ch1 and ch2,
   select TIM7 as trigger by writing 010 in TSEL2,
   and select TIM6 as trigger by keeping 000 in TSEL1 */
RCC->APB1ENR |= RCC_APB1ENR_DACEN; /* (1) */
DAC->CR |= DAC_CR_WAVE1_0 | DAC_CR_MAMP1_3
      | DAC_CR_MAMP2_3 | DAC_CR_WAVE2_0
      | DAC_CR_TSEL2_1 | DAC_CR_BOFF2
      | DAC_CR_TEN2 | DAC_CR_EN2
      | DAC_CR_BOFF1 | DAC_CR_TEN1
      | DAC_CR_EN1; /* (2) */
DAC->DHR12R1 = DAC_OUT1_VALUE; /* Initialize the DAC output value */
DAC->DHR12R2 = DAC_OUT2_VALUE; /* Initialize the DAC output value */

```

## A.8.3 Independent trigger with different LFSR generation code example

```

/* (1) Enable the peripheral clock of the DAC */
/* (2) Configure WAVEx at 01 and LFSR mask amplitude (MAMPx) at 1000 for a
   511-bits amplitude,
   LFSR mask amplitude (MAMP2) at 0111 i.e. a 255-bits amplitude for
   ch2,
   enable the DAC ch1 and ch2,
   disable buffer on ch1 and ch2,
   select TIM7 as trigger by writing 010 in TSEL2,
   and select TIM6 as trigger by keeping 000 in TSEL1 */
RCC->APB1ENR |= RCC_APB1ENR_DACEN; /* (1) */
DAC->CR |= DAC_CR_WAVE1_0 | DAC_CR_WAVE2_0 | DAC_CR_MAMP1_3
      | DAC_CR_MAMP2_2 | DAC_CR_MAMP2_1 | DAC_CR_MAMP2_0
      | DAC_CR_TSEL2_1 | DAC_CR_BOFF2 | DAC_CR_TEN2 | DAC_CR_EN2
      | DAC_CR_BOFF1 | DAC_CR_TEN1 | DAC_CR_EN1; /* (2) */
DAC->DHR12R1 = DAC_OUT1_VALUE; /* Initialize DAC output value */
DAC->DHR12R2 = DAC_OUT2_VALUE; /* Initialize DAC output value */

```

### A.8.4 Independent trigger with single triangle generation code example

```

/* (1) Enable the peripheral clock of the DAC */
/* (2) Configure WAVEx at 10 and LFSR mask amplitude (MAMPx) at 1001 for a
   1023-bits amplitude,
   enable the DAC ch1 and ch2,
   disable buffer on ch1 and ch2,
   select TIM7 as trigger by writing 010 in TSEL2
   and select TIM6 as trigger by keeping 000 in TSEL1 */
/* (3) Define the low value of the triangle on channel1 */
/* (4) Define the low value of the triangle on channel2 */
RCC->APB1ENR |= RCC_APB1ENR_DACEN; /* (1) */
DAC->CR |= DAC_CR_WAVE1_1 | DAC_CR_WAVE2_1
           | DAC_CR_MAMP1_3 | DAC_CR_MAMP1_0
           | DAC_CR_MAMP2_3 | DAC_CR_MAMP2_0
           | DAC_CR_TSEL2_1 | DAC_CR_BOFF2 | DAC_CR_TEN2 | DAC_CR_EN2
           | DAC_CR_BOFF1 | DAC_CR_TEN1 | DAC_CR_EN1; /* (2) */
DAC->DHR12R1 = DAC_OUT1_VALUE; /* (3) */
DAC->DHR12R2 = DAC_OUT2_VALUE; /* (4) */

```

### A.8.5 Independent trigger with different triangle generation code example

```

/* (1) Enable the peripheral clock of the DAC */
/* (2) Configure WAVEx at 10,
   configure mask amplitude for ch1 (MAMP1) at 1001 for a 1023-bits
   amplitude,
   and mask amplitude for ch2 (MAMP1) at 1011 for a 4095-bits amplitude,
   enable the DAC ch1 and ch2,
   disable buffer on ch1 and ch2,
   select TIM7 as trigger by writing 010 in TSEL2,
   and select TIM6 as trigger by keeping 000 in TSEL1 */
/* (3) Define the low value of the triangle on channel1 */
/* (4) Define the low value of the triangle on channel2 */
RCC->APB1ENR |= RCC_APB1ENR_DACEN; /* (1) */
DAC->CR |= DAC_CR_WAVE1_1 | DAC_CR_WAVE2_1
           | DAC_CR_MAMP1_3 | DAC_CR_MAMP1_0
           | DAC_CR_MAMP2_3 | DAC_CR_MAMP2_1 | DAC_CR_MAMP2_0
           | DAC_CR_TSEL2_1 | DAC_CR_BOFF2 | DAC_CR_TEN2 | DAC_CR_EN2
           | DAC_CR_BOFF1 | DAC_CR_TEN1 | DAC_CR_EN1; /* (2) */
DAC->DHR12R1 = DAC_OUT1_VALUE; /* (3) */
DAC->DHR12R2 = DAC_OUT2_VALUE; /* (4) */

```

### A.8.6 Simultaneous software start code example

```

/* Load the dual DAC channel data to the desired DHR register */
DAC->DHR12RD = (uint32_t)((signal1[x] << 16) + signal2[x]);

```

### A.8.7 Simultaneous trigger without wave generation code example

```

/* (1) Enable the peripheral clock of the DAC */
/* (2) Enable DMA transfer on DAC ch1 for both channels,
   enable the DAC ch1 and ch2,
   select TIM7 as trigger by writing 010 in TSEL1 and TSEL2 */
RCC->APB1ENR |= RCC_APB1ENR_DACEN; /* (1) */
DAC->CR |= DAC_CR_TSEL1_1 | DAC_CR_TEN2 | DAC_CR_EN2
           | DAC_CR_TSEL2_1 | DAC_CR_TEN1 | DAC_CR_EN1; /* (2) */
/* Initialize the dual DAC value */
DAC->DHR12RD = (uint32_t)((2048 << 16) + 2048);

```

### A.8.8 Simultaneous trigger with single LFSR generation code example

```

/* (1) Enable the peripheral clock of the DAC */
/* (2) Configure WAVEx at 01 and LFSR mask amplitude (MAMPx) at 1000 for a
   511-bits amplitude,
   enable the DAC ch1 and ch2,
   disable buffer on ch1 and ch2,
   select TIM7 as trigger by writing 010 in TSEL1 and TSEL2 */
RCC->APB1ENR |= RCC_APB1ENR_DACEN; /* (1) */
DAC->CR |= DAC_CR_WAVE1_0 | DAC_CR_WAVE2_0
           | DAC_CR_MAMP1_3 | DAC_CR_MAMP2_3
           | DAC_CR_TSEL2_1 | DAC_CR_BOFF2
           | DAC_CR_TEN2 | DAC_CR_EN2
           | DAC_CR_TSEL1_1 | DAC_CR_BOFF1
           | DAC_CR_TEN1 | DAC_CR_EN1; /* (2) */
/* Initialize the dual register */
DAC->DHR12RD = (uint32_t)((DAC_OUT2_VALUE << 16) + DAC_OUT1_VALUE);

```

### A.8.9 Simultaneous trigger with different LFSR generation code example

```

/* (1) Enable the peripheral clock of the DAC */
/* (2) Configure WAVEx at 01 and LFSR mask amplitude (MAMP1) at 1000 for a
   511-bits amplitude,
   set LFSR mask amplitude (MAMP2) at 0111 i.e. a 255-bits amplitude for
   ch2,
   enable the DAC ch1 and ch2,
   disable buffer on ch1 and ch2,
   select TIM7 as trigger by writing 010 in TSEL1 and TSEL2 */
RCC->APB1ENR |= RCC_APB1ENR_DACEN; /* (1) */
DAC->CR |= DAC_CR_WAVE1_0 | DAC_CR_WAVE2_0 | DAC_CR_MAMP1_3
           | DAC_CR_MAMP2_2 | DAC_CR_MAMP2_1 | DAC_CR_MAMP2_0
           | DAC_CR_TSEL2_1 | DAC_CR_BOFF2
           | DAC_CR_TEN2 | DAC_CR_EN2
           | DAC_CR_TSEL1_1 | DAC_CR_BOFF1
           | DAC_CR_TEN1 | DAC_CR_EN1; /* (2) */
/* Initialize the dual register */
DAC->DHR12RD = (uint32_t)((DAC_OUT2_VALUE << 16) + DAC_OUT1_VALUE);

```

### A.8.10 Simultaneous trigger with single triangle generation code example

```

/* (1) Enable the peripheral clock of the DAC */
/* (2) Configure WAVEx at 10 and LFSR mask amplitude (MAMPx) at 1001 for a
   1023-bits amplitude,
   enable the DAC ch1 and ch2,
   disable buffer on ch1 and ch2,
   select TIM7 as trigger by writing 010 in TSEL1 and TSEL2 */
RCC->APB1ENR |= RCC_APB1ENR_DACEN; /* (1) */
DAC->CR |= DAC_CR_WAVE1_1 | DAC_CR_WAVE2_1
| DAC_CR_MAMP1_3 | DAC_CR_MAMP1_0
| DAC_CR_MAMP2_3 | DAC_CR_MAMP2_0
| DAC_CR_TSEL2_1 | DAC_CR_BOFF2
| DAC_CR_TEN2 | DAC_CR_EN2
| DAC_CR_TSEL1_1 | DAC_CR_BOFF1
| DAC_CR_TEN1 | DAC_CR_EN1; /* (2) */
/* Initialize the dual register */
DAC->DHR12RD = (uint32_t)((DAC_OUT2_VALUE << 16) + DAC_OUT1_VALUE);

```

### A.8.11 Simultaneous trigger with different triangle generation code example

```

/* (1) Enable the peripheral clock of the DAC */
/* (2) Configure WAVEx at 10,
   configure mask amplitude for ch1 (MAMP1) at 1001 for a 1023-bits
   amplitude and mask amplitude for ch2 (MAMP1) at 1011 for a 4095-bits
   amplitude,
   enable the DAC ch1 and ch2,
   select TIM7 as trigger by writing 010 in TSEL1 and TSEL2 */
RCC->APB1ENR |= RCC_APB1ENR_DACEN; /* (1) */
DAC->CR |= DAC_CR_WAVE1_1 | DAC_CR_WAVE2_1
| DAC_CR_MAMP1_3 | DAC_CR_MAMP1_0
| DAC_CR_MAMP2_3 | DAC_CR_MAMP2_1 | DAC_CR_MAMP2_0
| DAC_CR_TSEL2_1 | DAC_CR_TEN2 | DAC_CR_EN2
| DAC_CR_TSEL1_1 | DAC_CR_TEN1 | DAC_CR_EN1; /* (2) */
/* Initialize the dual register */
DAC->DHR12RD = (uint32_t)((DAC_OUT2_VALUE << 16) + DAC_OUT1_VALUE);

```

### A.8.12 DMA initialization code example

```
/* (1) Enable DMA transfer on DAC ch1 for both channels,
   enable interrupt on DMA underrun DAC,
   enable the DAC ch1 and ch2,
   select TIM7 as trigger by writing 010 in TSEL1 and TSEL2 */
DAC->CR |= DAC_CR_TSEL1_1 | DAC_CR_TEN2 | DAC_CR_EN2
           | DAC_CR_TSEL2_1 | DAC_CR_DMAUDRIE1 | DAC_CR_DMAEN1
           | DAC_CR_TEN1 | DAC_CR_EN1; /* (1) */

/* (1) Enable the peripheral clock on DMA */
/* (2) Configure the peripheral data register address */
/* (3) Configure the memory address */
/* (4) Configure the number of DMA tranfer to be performs on channel 3 */
/* (5) Configure increment, size (32-bits), interrupts, transfer from
   memory to peripheral and circular mode */
/* (6) Enable DMA Channel 3 */
RCC->AHBENR |= RCC_AHBENR_DMA1EN; /* (1) */
DMA1_Channel3->CPAR = (uint32_t) (&(DAC->DHR12RD)); /* (2) */
DMA1_Channel3->CMAR = (uint32_t) signal_data; /* (3) */
DMA1_Channel3->CNDTR = SIGNAL_ARRAY_SIZE; /* (4) */
DMA1_Channel3->CCR |= DMA_CCR_MINC | DMA_CCR_MSIZE_1 | DMA_CCR_PSIZE_1
           | DMA_CCR_TEIE | DMA_CCR_CIRC; /* (5) */
DMA1_Channel3->CCR |= DMA_CCR_EN; /* (6) */
```

## A.9 Timers

### A.9.1 Upcounter on TI2 rising edge code example

```
/* (1) Enable the peripheral clock of Timer 1 */
/* (2) Enable the peripheral clock of GPIOA */
/* (3) Select Alternate function mode (10) on GPIOA pin 9 */
/* (4) Select TIM1_CH2 on PA9 by enabling AF2 for pin 9 in GPIOA AFRH
   register */

RCC->APB2ENR |= RCC_APB2ENR_TIM1EN; /* (1) */
RCC->AHBENR |= RCC_AHBENR_GPIOAEN; /* (2) */
GPIOA->MODER = (GPIOA->MODER & ~(GPIO_MODER_MODER9))
  | (GPIO_MODER_MODER9_1); /* (3) */
GPIOA->AFR[1] |= 0x2 << ((9-8)*4); /* (4) */

/* (1) Configure channel 2 to detect rising edges on the TI2 input by
   writing CC2S = '01', and configure the input filter duration by
   writing the IC2F[3:0] bits in the TIMx_CCMR1 register (if no filter
   is needed, keep IC2F=0000).*/
/* (2) Select rising edge polarity by writing CC2P=0 in the TIMx_CCER
   register (reset value). */
/* (3) Configure the timer in external clock mode 1 by writing SMS=111
   Select TI2 as the trigger input source by writing TS=110
   in the TIMx_SMCR register.*/
/* (4) Enable the counter by writing CEN=1 in the TIMx_CR1 register. */
TIMx->CCMR1 |= TIM_CCMR1_IC2F_0 | TIM_CCMR1_IC2F_1
  | TIM_CCMR1_CC2S_0; /* (1) */
TIMx->CCER &= (uint16_t)(~TIM_CCER_CC2P); /* (2) */
TIMx->SMCR |= TIM_SMCR_SMS | TIM_SMCR_TS_2 | TIM_SMCR_TS_1; /* (3) */
TIMx->CR1 |= TIM_CR1_CEN; /* (4) */
```

## A.9.2 Up counter on each 2 ETR rising edges code example

```

/* (1) Enable the peripheral clock of Timer 1 */
/* (2) Enable the peripheral clock of GPIOA */
/* (3) Select Alternate function mode (10) on GPIOA pin 12 */
/* (4) Select TIM1_ETR on PA12 by enabling AF2 for pin 12 in GPIOA AFRH
   register */

RCC->APB2ENR |= RCC_APB2ENR_TIM1EN; /* (1) */
RCC->AHBENR |= RCC_AHBENR_GPIOAEN; /* (2) */
GPIOA->MODER = (GPIOA->MODER & ~(GPIO_MODER_MODER12))
  | (GPIO_MODER_MODER12_1); /* (3) */
GPIOA->AFR[1] |= 0x2 << ((12-8)*4); /* (4) */

/* (1) As no filter is needed in this example, write ETF[3:0]=0000
   in the TIMx_SMCR register. Keep the reset value.
   Set the prescaler by writing ETPS[1:0]=01 in the TIMx_SMCR
   register.
   Select rising edge detection on the ETR pin by writing ETP=0
   in the TIMx_SMCR register. Keep the reset value.
   Enable external clock mode 2 by writing ECE=1 in the TIMx_SMCR
   register. */

/* (2) Enable the counter by writing CEN=1 in the TIMx_CR1 register. */
TIMx->SMCR |= TIM_SMCR_ETPS_0 | TIM_SMCR_ECE; /* (1) */
TIMx->CR1 |= TIM_CR1_CEN; /* (2) */

```

## A.9.3 Input capture configuration code example

```

/* (1) Select the active input TI1 (CC1S = 01),
   program the input filter for 8 clock cycles (IC1F = 0011),
   select the rising edge on CC1 (CC1P = 0, reset value)
   and prescaler at each valid transition (IC1PS = 00, reset value) */
/* (2) Enable capture by setting CC1E */
/* (3) Enable interrupt on Capture/Compare */
/* (4) Enable counter */

TIMx->CCMR1 |= TIM_CCMR1_CC1S_0
  | TIM_CCMR1_IC1F_0 | TIM_CCMR1_IC1F_1; /* (1) */
TIMx->CCER |= TIM_CCER_CC1E; /* (2) */
TIMx->DIER |= TIM_DIER_CC1IE; /* (3) */
TIMx->CR1 |= TIM_CR1_CEN; /* (4) */

```

### A.9.4 Input capture data management code example

```
This code must be inserted in the Timer interrupt subroutine.  
if ((TIMx->SR & TIM_SR_CC1IF) != 0)  
{  
    if ((TIMx->SR & TIM_SR_CC1OF) != 0) /* Check the overflow */  
    {  
        /* Overflow error management */  
        gap = 0; /* Reinitialize the laps computing */  
        TIMx->SR &= ~(TIM_SR_CC1OF | TIM_SR_CC1IF); /* Clear the flags */  
        return;  
    }  
    if (gap == 0) /* Test if it is the first rising edge */  
    {  
        counter0 = TIMx->CCR1; /* Read the capture counter which clears the  
                                CC1ICF */  
        gap = 1; /* Indicate that the first rising edge has yet been detected */  
    }  
    else  
    {  
        counter1 = TIMx->CCR1; /* Read the capture counter which clears the  
                                CC1ICF */  
        if (counter1 > counter0) /* Check capture counter overflow */  
        {  
            Counter = counter1 - counter0;  
        }  
        else  
        {  
            Counter = counter1 + 0xFFFF - counter0 + 1;  
        }  
        counter0 = counter1;  
    }  
}  
    else  
    {  
        /* Unexpected Interrupt */  
        /* Manage an error for robust application */  
    }  
}
```

**Note:** This code manages only a single counter overflow. To manage many counter overflows the update interrupt must be enabled (UIE = 1) and properly managed.

## A.9.5 PWM input configuration code example

```

/* (1) Select the active input TI1 for TIMx_CCR1 (CC1S = 01),
   select the active input TI1 for TIMx_CCR2 (CC2S = 10) */
/* (2) Select TI1FP1 as valid trigger input (TS = 101)
   configure the slave mode in reset mode (SMS = 100) */
/* (3) Enable capture by setting CC1E and CC2E
   select the rising edge on CC1 and CC1N (CC1P = 0 and CC1NP = 0, reset
   value),
   select the falling edge on CC2 (CC2P = 1). */
/* (4) Enable interrupt on Capture/Compare 1 */
/* (5) Enable counter */

TIMx->CCMR1 |= TIM_CCMR1_CC1S_0 | TIM_CCMR1_CC2S_1; /* (1)*/
TIMx->SMCR |= TIM_SMCR_TS_2 | TIM_SMCR_TS_0
  | TIM_SMCR_SMS_2; /* (2) */
TIMx->CCER |= TIM_CCER_CC1E | TIM_CCER_CC2E | TIM_CCER_CC2P; /* (3) */
TIMx->DIER |= TIM_DIER_CC1IE; /* (4) */
TIMx->CR1 |= TIM_CR1_CEN; /* (5) */

```

## A.9.6 PWM input with DMA configuration code example

```

/* (1) Enable the peripheral clock on DMA */
/* (2) Configure the peripheral data register address for DMA channel x */
/* (3) Configure the memory address for DMA channel x */
/* (4) Configure the number of DMA tranfers to be performed
   on DMA channel x */
/* (5) Configure no increment (reset value), size (16-bits), interrupts,
   transfer from peripheral to memory and circular mode
   for DMA channel x */
/* (6) Enable DMA Channel x */

RCC->AHBENR |= RCC_AHBENR_DMA1EN; /* (1) */
DMA1_Channel2->CPAR = (uint32_t) (&(TIM1->CCR1)); /* (2) */
DMA1_Channel2->CMAR = (uint32_t) (&Period); /* (3) */
DMA1_Channel2->CNDTR = 1; /* (4) */
DMA1_Channel2->CCR |= DMA_CCR_MSIZE_0 | DMA_CCR_PSIZE_0
  | DMA_CCR_TEIE | DMA_CCR_CIRC; /* (5) */
DMA1_Channel2->CCR |= DMA_CCR_EN; /* (6) */
/* repeat (2) to (6) for channel 3 */

DMA1_Channel3->CPAR = (uint32_t) (&(TIM1->CCR2)); /* (2) */
DMA1_Channel3->CMAR = (uint32_t) (&DutyCycle); /* (3) */
DMA1_Channel3->CNDTR = 1; /* (4) */
DMA1_Channel3->CCR |= DMA_CCR_MSIZE_0 | DMA_CCR_PSIZE_0
  | DMA_CCR_TEIE | DMA_CCR_CIRC; /* (5) */
DMA1_Channel3->CCR |= DMA_CCR_EN; /* (6) */

/* Configure NVIC for DMA */
/* (7) Enable Interrupt on DMA Channels x */
/* (8) Set priority for DMA Channels x */
NVIC_EnableIRQ(DMA1_Channel2_3_IRQn); /* (7) */
NVIC_SetPriority(DMA1_Channel2_3_IRQn, 3); /* (8) */

```

### A.9.7 Output compare configuration code example

```

/* (1) Set prescaler to 3, so APBCLK/4 i.e 12MHz */
/* (2) Set ARR = 12000 -1 */
/* (3) Set CCRx = ARR, as timer clock is 12MHz, an event occurs each 1 ms */
/* (4) Select toggle mode on OC1 (OC1M = 011),
   disable preload register on OC1 (OC1PE = 0, reset value) */
/* (5) Select active high polarity on OC1 (CC1P = 0, reset value),
   enable the output on OC1 (CC1E = 1) */
/* (6) Enable output (MOE = 1) */
/* (7) Enable counter */
TIMx->PSC |= 3; /* (1) */
TIMx->ARR = 12000 - 1; /* (2) */
TIMx->CCR1 = 12000 - 1; /* (3) */
TIMx->CCMR1 |= TIM_CCMR1_OC1M_0 | TIM_CCMR1_OC1M_1; /* (4) */
TIMx->CCER |= TIM_CCER_CC1E; /* (5) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (6) */
TIMx->CR1 |= TIM_CR1_CEN; /* (7) */

```

### A.9.8 Edge-aligned PWM configuration example

```

/* (1) Set prescaler to 47, so APBCLK/48 i.e 1MHz */
/* (2) Set ARR = 8, as timer clock is 1MHz the period is 9 us */
/* (3) Set CCRx = 4, , the signal will be high during 4 us */
/* (4) Select PWM mode 1 on OC1 (OC1M = 110),
   enable preload register on OC1 (OC1PE = 1) */
/* (5) Select active high polarity on OC1 (CC1P = 0, reset value),
   enable the output on OC1 (CC1E = 1) */
/* (6) Enable output (MOE = 1) */
/* (7) Enable counter (CEN = 1)
   select edge aligned mode (CMS = 00, reset value)
   select direction as upcounter (DIR = 0, reset value) */
/* (8) Force update generation (UG = 1) */
TIMx->PSC = 47; /* (1) */
TIMx->ARR = 8; /* (2) */
TIMx->CCR1 = 4; /* (3) */
TIMx->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1
  | TIM_CCMR1_OC1PE; /* (4) */
TIMx->CCER |= TIM_CCER_CC1E; /* (5) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (6) */
TIMx->CR1 |= TIM_CR1_CEN; /* (7) */
TIMx->EGR |= TIM_EGR_UG; /* (8) */

```

### A.9.9 Center-aligned PWM configuration example

```
/* (1) Set prescaler to 47, so APBCLK/48 i.e 1MHz */
/* (2) Set ARR = 8, as timer clock is 1MHz and center-aligned counting,
   the period is 16 us */
/* (3) Set CCRx = 7, the signal will be high during 14 us */
/* (4) Select PWM mode 1 on OC1 (OC1M = 110),
   enable preload register on OC1 (OC1PE = 1, reset value) */
/* (5) Select active high polarity on OC1 (CC1P = 0, reset value),
   enable the output on OC1 (CC1E = 1)*/
/* (6) Enable output (MOE = 1)*/
/* (7) Enable counter (CEN = 1)
   select center-aligned mode 1 (CMS = 01) */
/* (8) Force update generation (UG = 1) */

TIMx->PSC = 47; /* (1) */
TIMx->ARR = 8; /* (2) */
TIMx->CCR1 = 7; /* (3) */
TIMx->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1
              | TIM_CCMR1_OC1PE; /* (4) */
TIMx->CCER |= TIM_CCER_CC1E; /* (5) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (6) */
TIMx->CR1 |= TIM_CR1_CMS_0 | TIM_CR1_CEN; /* (7) */
TIMx->EGR |= TIM_EGR_UG; /* (8) */
```

### A.9.10 ETR configuration to clear OCxREF code example

```

/* This code is similar to the edge-aligned PWM configuration but it enables
   the clearing on OC1 for ETRclearing (OC1CE = 1) in CCMR1 (5) and ETR is
   configured in SMCR (7).*/
/* (1) Set prescaler to 47, so APBCLK/48 i.e 1MHz */
/* (2) Set ARR = 8, as timer clock is 1MHz the period is 9 us */
/* (3) Set CCRx = 4, , the signal will be high during 4 us */
/* (4) Select PWM mode 1 on OC1 (OC1M = 110),
       enable preload register on OC1 (OC1PE = 1),
       enable clearing on OC1 for ETR clearing (OC1CE = 1) */
/* (5) Select active high polarity on OC1 (CC1P = 0, reset value),
       enable the output on OC1 (CC1E = 1) */
/* (6) Enable output (MOE = 1) */
/* (7) Select ETR as OCREF clear source (OCCS = 1),
       select External Trigger Prescaler off (ETPS = 00, reset value),
       disable external clock mode 2 (ECE = 0, reset value),
       select active at high level (ETP = 0, reset value) */
/* (8) Enable counter (CEN = 1),
       select edge aligned mode (CMS = 00, reset value),
       select direction as upcounter (DIR = 0, reset value) */
/* (9) Force update generation (UG = 1) */

TIMx->PSC = 47; /* (1) */
TIMx->ARR = 8; /* (2) */
TIMx->CCR1 = 4; /* (3) */
TIMx->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1PE
           | TIM_CCMR1_OC1CE; /* (4) */
TIMx->CCER |= TIM_CCER_CC1E; /* (5) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (6) */
TIMx->SMCR |= TIM_SMCR_OCCS; /* (7) */
TIMx->CR1 |= TIM_CR1_CEN; /* (8) */
TIMx->EGR |= TIM_EGR_UG; /* (9) */

```

### A.9.11 Encoder interface code example

```

/* (1) Configure TI1FP1 on TI1 (CC1S = 01),
   configure TI1FP2 on TI2 (CC2S = 01) */
/* (2) Configure TI1FP1 and TI1FP2 non inverted (CC1P = CC2P = 0, reset
   value) */
/* (3) Configure both inputs are active on both rising and falling edges
   (SMS = 011) */
/* (4) Enable the counter by writing CEN=1 in the TIMx_CR1 register. */
TIMx->CCMR1 |= TIM_CCMR1_CC1S_0 | TIM_CCMR1_CC2S_0; /* (1) */
TIMx->CCER &= ~(TIM_CCER_CC21 | TIM_CCER_CC2P); /* (2) */
TIMx->SMCR |= TIM_SMCR_SMS_0 | TIM_SMCR_SMS_1; /* (3) */
TIMx->CR1 |= TIM_CR1_CEN; /* (4) */

```

### A.9.12 Reset mode code example

```

/* (1) Configure channel 1 to detect rising edges on the TI1 input
   by writing CC1S = '01',
   and configure the input filter duration by writing the IC1F[3:0]
   bits in the TIMx_CCMR1 register (if no filter is needed, keep
   IC1F=0000).*/
/* (2) Select rising edge polarity by writing CC1P=0 in the TIMx_CCER
   register
   Not necessary as it keeps the reset value. */
/* (3) Configure the timer in reset mode by writing SMS=100
   Select TI1 as the trigger input source by writing TS=101
   in the TIMx_SMCR register.*/
/* (4) Set prescaler to 48000-1 in order to get an increment each 1ms */
/* (5) Enable the counter by writing CEN=1 in the TIMx_CR1 register. */
TIMx->CCMR1 |= TIM_CCMR1_CC1S_0; /* (1)*/
TIMx->CCER &= (uint16_t)(~TIM_CCER_CC1P); /* (2) */
TIMx->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_TS_2 | TIM_SMCR_TS_0; /* (3) */
TIM1->PSC = 47999; /* (4) */
TIMx->CR1 |= TIM_CR1_CEN; /* (5) */

```

### A.9.13 Gated mode code example

```

/* (1) Configure channel 1 to detect low level on the TI1 input
   by writing CC1S = '01',
   and configure the input filter duration by writing the IC1F[3:0]
   bits in the TIMx_CCMR1 register (if no filter is needed,
   keep IC1F=0000).*/
/* (2) Select polarity by writing CC1P=1 in the TIMx_CCER register */
/* (3) Configure the timer in gated mode by writing SMS=101
   Select TI1 as the trigger input source by writing TS=101
   in the TIMx_SMCR register.*/
/* (4) Set prescaler to 12000-1 in order to get an increment each 250us */
/* (5) Enable the counter by writing CEN=1 in the TIMx_CR1 register. */
TIMx->CCMR1 |= TIM_CCMR1_CC1S_0; /* (1)*/
TIMx->CCER |= TIM_CCER_CC1P; /* (2) */
TIMx->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_0
              | TIM_SMCR_TS_2 | TIM_SMCR_TS_0; /* (3) */
TIMx->PSC = 11999; /* (4) */
TIMx->CR1 |= TIM_CR1_CEN; /* (5) */

```

### A.9.14 Trigger mode code example

```

/* (1) Configure channel 2 to detect rising edge on the TI2 input
   by writing CC2S = '01',
   and configure the input filter duration by writing the IC1F[3:0]
   bits in the TIMx_CCMR1 register (if no filter is needed,
   keep IC1F=0000). */
/* (2) Select polarity by writing CC2P=0 (reset value) in the TIMx_CCER
   register */
/* (3) Configure the timer in trigger mode by writing SMS=110
   Select TI2 as the trigger input source by writing TS=110
   in the TIMx_SMCR register. */
/* (4) Set prescaler to 12000-1 in order to get an increment each 250us */
TIMx->CCMR1 |= TIM_CCMR1_CC2S_0; /* (1)*/
TIMx->CCER &= ~TIM_CCER_CC2P; /* (2)*/
TIMx->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_1
            | TIM_SMCR_TS_2 | TIM_SMCR_TS_1; /* (3)*/
TIM1->PSC = 11999; /* (4)*/

```

### A.9.15 External clock mode 2 + trigger mode code example

```

/* (1) Configure no input filter (ETF=0000, reset value)
   configure prescaler disabled (ETPS = 0, reset value)
   select detection on rising edge on ETR (ETP = 0, reset value)
   enable external clock mode 2 (ECE = 1) */
/* (2) Configure no input filter (IC1F=0000, reset value)
   select input capture source on TI1 (CC1S = 01) */
/* (3) Select polarity by writing CC1P=0 (reset value) in the TIMx_CCER
   register */
/* (4) Configure the timer in trigger mode by writing SMS=110
   Select TI1 as the trigger input source by writing TS=101
   in the TIMx_SMCR register. */
TIMx->SMCR |= TIM_SMCR_ECE; /* (1) */
TIMx->CCMR1 |= TIM_CCMR1_CC1S_0; /* (2) */
TIMx->CCER &= ~TIM_CCER_CC1P; /* (3) */
TIMx->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_1
            | TIM_SMCR_TS_2 | TIM_SMCR_TS_0; /* (4) */
/* Use TI2FP2 as trigger 1 */
/* (1) Map TI2FP2 on TI2 by writing CC2S=01 in the TIMx_CCMR1 register */
/* (2) TI2FP2 must detect a rising edge, write CC2P=0 and CC2NP=0
   in the TIMx_CCER register (keep the reset value) */
/* (3) Configure TI2FP2 as trigger for the slave mode controller (TRGI)
   by writing TS=110 in the TIMx_SMCR register,
   TI2FP2 is used to start the counter by writing SMS to '110'
   in the TIMx_SMCR register (trigger mode) */
TIMx->CCMR1 |= TIM_CCMR1_CC2S_0; /* (1) */
//TIMx->CCER &= ~(TIM_CCER_CC2P | TIM_CCER_CC2NP); /* (2) */
TIMx->SMCR |= TIM_SMCR_TS_2 | TIM_SMCR_TS_1
            | TIM_SMCR_SMS_2 | TIM_SMCR_SMS_1; /* (3) */

```

### A.9.16 One-Pulse mode code example

```

/* The OPM waveform is defined by writing the compare registers */
/* (1) Set prescaler to 47, so APBCLK/48 i.e 1MHz */
/* (2) Set ARR = 7, as timer clock is 1MHz the period is 8 us */
/* (3) Set CCRx = 5, the burst will be delayed for 5 us (must be > 0) */
/* (4) Select PWM mode 2 on OC1 (OC1M = 111),
   enable preload register on OC1 (OC1PE = 1, reset value)
   enable fast enable (no delay) if PULSE_WITHOUT_DELAY is set */
/* (5) Select active high polarity on OC1 (CC1P = 0, reset value),
   enable the output on OC1 (CC1E = 1) */
/* (6) Enable output (MOE = 1) */
/* (7) Write '1 in the OPM bit in the TIMx_CR1 register to stop the counter
   at the next update event (OPM = 1),
   enable auto-reload register(ARPE = 1) */

TIMx->PSC = 47; /* (1) */
TIMx->ARR = 7; /* (2) */
TIMx->CCR1 = 5; /* (3) */
TIMx->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_0
             | TIM_CCMR1_OC1PE
#if PULSE_WITHOUT_DELAY > 0
             | TIM_CCMR1_OC1FE
#endif
; /* (4) */
TIMx->CCER |= TIM_CCER_CC1E; /* (5) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (6) */
TIMx->CR1 |= TIM_CR1_OPM | TIM_CR1_ARPE; /* (7) */

```

### A.9.17 Timer prescaling another timer code example

```

/* TIMy is slave of TIMx */
/* (1) Select Update Event as Trigger output (TRGO) by writing MMS = 010
   in TIMx_CR2. */
/* (2) Configure TIMy in slave mode using ITR1 as internal trigger
   by writing TS = 000 in TIMy_SMCR (reset value)
   Configure TIMy in external clock mode 1, by writing SMS=111 in the
   TIMy_SMCR register. */
/* (3) Set TIMx prescaler to 47999 in order to get an increment each 1ms */
/* (4) Set TIMx Autoreload to 999 in order to get an overflow (so an UEV)
   each second */
/* (5) Set TIMx Autoreload to 24*3600-1 in order to get an overflow each 24-
   hour */
/* (6) Enable the counter by writing CEN=1 in the TIMx_CR1 register. */
/* (7) Enable the counter by writing CEN=1 in the TIMy_CR1 register. */

TIMx->CR2 |= TIM_CR2_MMS_1; /* (1) */
TIMy->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_1 | TIM_SMCR_SMS_0; /* (2) */
TIMx->PSC = 47999; /* (3) */
TIMx->ARR = 999; /* (4) */
TIMy->ARR = (24 * 3600) - 1; /* (5) */
TIMx->CR1 |= TIM_CR1_CEN; /* (6) */
TIMy->CR1 |= TIM_CR1_CEN; /* (7) */

```

### A.9.18 Timer enabling another timer code example

```

/* TIMy is slave of TIMx */
/* (1) Configure Timer x master mode to send its Output Compare 1 Reference
   (OC1REF) signal as trigger output
   (MMS=100 in the TIM1_CR2 register). */
/* (2) Configure the Timer x OC1REF waveform (TIM1_CCMR1 register)
   Channel 1 is in PWM mode 1 when the counter is less than the
   capture/compare register (write OC1M = 110) */
/* (3) Configure TIMy in slave mode using ITR1 as internal trigger
   by writing TS = 000 in TIMy_SMCR (reset value)
   Configure TIMy in gated mode, by writing SMS=101 in the
   TIMy_SMCR register. */
/* (4) Set TIMx prescaler to 2 */
/* (5) Set TIMy prescaler to 2 */
/* (6) Set TIMx Autoreload to 999 in order to get an overflow (so an UEV)
   each 100ms */
/* (7) Set capture compare register to a value between 0 and 999 */
TIMx->CR2 |= TIM_CR2_MMS_2; /* (1) */
TIMx->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1; /* (2) */
TIMy->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_0; /* (3) */
TIMx->PSC = 2; /* (4) */
TIMy->PSC = 2; /* (5) */
TIMx->ARR = 999; /* (6) */
TIMx->CCR1 = 700; /* (7) */
/* Configure the slave timer to generate toggling on each count */
/* (1) Configure the TIMy in PWM mode 1 (write OC1M = 110) */
/* (2) Set TIMy Autoreload to 1 */
/* (3) Set capture compare register to 1 */
TIMy->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1; /* (1) */
TIMy->ARR = 1; /* (2) */
TIMy->CCR1 = 1; /* (3) */
/* Enable the output of TIMx OC1 */
/* (1) Select active high polarity on OC1 (CC1P = 0, reset value),
   enable the output on OC1 (CC1E = 1) */
/* (2) Enable output (MOE = 1) */
TIMx->CCER |= TIM_CCER_CC1E; /* (1) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (2) */
/* Enable the output of TIMy OC1 */
/* (1) Select active high polarity on OC1 (CC1P = 0, reset value),
   enable the output on OC1 (CC1E = 1) */
/* (2) Enable output (MOE = 1) */
TIMy->CCER |= TIM_CCER_CC1E; /* (1) */
TIMy->BDTR |= TIM_BDTR_MOE; /* (2) */
/* (1) Enable the slave counter first by writing CEN=1
   in the TIMy_CRL register. */
/* (2) Enable the master counter by writing CEN=1
   in the TIMx_CRL register. */
TIMy->CR1 |= TIM_CR1_CEN; /* (1) */
TIMx->CR1 |= TIM_CR1_CEN; /* (2) */

```

### A.9.19 Master and slave synchronization code example

```

/* (1) Configure Timer x master mode to send its enable signal
   as trigger output (MMS=001 in the TIM1_CR2 register). */
/* (2) Configure the Timer x Channel 1 waveform (TIM1_CCMR1 register)
   is in PWM mode 1 (write OC1M = 110) */
/* (3) Configure TIMy in slave mode using ITR1 as internal trigger
   by writing TS = 000 in TIMy_SMCR (reset value)
   Configure TIMy in gated mode, by writing SMS=101 in the
   TIMy_SMCR register. */
/* (4) Set TIMx prescaler to 2 */
/* (5) Set TIMy prescaler to 2 */
/* (6) Set TIMx Autoreload to 99 in order to get an overflow (so an UEV)
   each 10ms */
/* (7) Set capture compare register to a value between 0 and 99 */
TIMx->CR2 |= TIM_CR2_MMS_0; /* (1) */
TIMx->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1; /* (2) */
TIMy->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_0; /* (3) */
TIMx->PSC = 2; /* (4) */
TIMy->PSC = 2; /* (5) */
TIMx->ARR = 99; /* (6) */
TIMx->CCR1 = 25; /* (7) */
/* Configure the slave timer Channel 1 as PWM as Timer
   to show synchronicity */
/* (1) Configure the TIMy in PWM mode 1 (write OC1M = 110) */
/* (2) Set TIMy Autoreload to 99 */
/* (3) Set capture compare register to 25 */
TIMy->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1; /* (1) */
TIMy->ARR = 99; /* (2) */
TIMy->CCR1 = 25; /* (3) */
/* Enable the output of TIMx OC1 */
/* (1) Select active high polarity on OC1 (CC1P = 0, reset value),
   enable the output on OC1 (CC1E = 1) */
/* (2) Enable output (MOE = 1) */
TIMx->CCER |= TIM_CCER_CC1E; /* (1) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (2) */
/* Enable the output of TIMy OC1 */
/* (1) Select active high polarity on OC1 (CC1P = 0, reset value),
   enable the output on OC1 (CC1E = 1) */
/* (2) Enable output (MOE = 1) */
TIMy->CCER |= TIM_CCER_CC1E; /* (1) */
TIMy->BDTR |= TIM_BDTR_MOE; /* (2) */
/* (1) Reset Timer x by writing '1 in UG bit (TIMx_EGR register) */
/* (2) Reset Timer y by writing '1 in UG bit (TIMy_EGR register) */
TIMx->EGR |= TIM_EGR_UG; /* (1) */
TIMy->EGR |= TIM_EGR_UG; /* (2) */
/* (1) Enable the slave counter first by writing CEN=1 in the TIMy_CR1
   register.
   TIMy will start synchronously with the master timer */
/* (2) Start the master counter by writing CEN=1
   in the TIMx_CR1 register. */
TIMy->CR1 |= TIM_CR1_CEN; /* (1) */
TIMx->CR1 |= TIM_CR1_CEN; /* (2) */

```

### A.9.20 Two timers synchronized by an external trigger code example

```

/* (1) Configure TIMx master mode to send its enable signal
   as trigger output (MMS=001 in the TIM1_CR2 register). */
/* (2) Configure TIMx in slave mode to get the input trigger from TI1
   by writing TS = 100 in TIMx_SMCR
   Configure TIMx in trigger mode, by writing SMS=110 in the
   TIMx_SMCR register.
   Configure TIMx in Master/Slave mode by writing MSM = 1
   in TIMx_SMCR */
/* (3) Configure TIMy in slave mode to get the input trigger from Timer1
   by writing TS = 000 in TIMy_SMCR (reset value)
   Configure TIMy in trigger mode, by writing SMS=110 in the
   TIMy_SMCR register. */
/* (4) Reset Timer x counter by writing '1 in UG bit (TIMx_EGR register) */
/* (5) Reset Timer y counter by writing '1 in UG bit (TIMy_EGR register) */
TIMx->CR2 |= TIM_CR2_MMS_0; /* (1)*/
TIMx->SMCR |= TIM_SMCR_TS_2 | TIM_SMCR_SMS_2 | TIM_SMCR_SMS_1
    | TIM_SMCR_MSM; /* (2) */
TIMy->SMCR |= TIM_SMCR_SMS_2 | TIM_SMCR_SMS_1; /* (3) */
TIMx->EGR |= TIM_EGR_UG; /* (4) */
TIMy->EGR |= TIM_EGR_UG; /* (5) */
/* Configure the Timer Channel 2 as PWM */
/* (1) Configure the Timer x Channel 2 waveform (TIM1_CCMR1 register)
   is in PWM mode 1 (write OC2M = 110) */
/* (2) Set TIMx prescaler to 2 */
/* (3) Set TIMx Autoreload to 99 in order to get an overflow (so an UEV)
   each 10ms */
/* (4) Set capture compare register to a value between 0 and 99 */
TIMx->CCMR1 |= TIM_CCMR1_OC2M_2 | TIM_CCMR1_OC2M_1; /* (1) */
TIMx->PSC = 2; /* (2) */
TIMx->ARR = 99; /* (3) */
TIMx->CCR2 = 25; /* (4) */
/* Configure the slave timer Channel 1 as PWM as Timer
   to show synchronicity */
/* (1) Configure the TIMy in PWM mode 1 (write OC1M = 110) */
/* (2) Set TIMy prescaler to 2 */
/* (3) Set TIMx Autoreload to 99 */
/* (4) Set capture compare register to 25 */
TIMy->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1; /* (1) */
TIMy->PSC = 2; /* (2) */
TIMy->ARR = 99; /* (3) */
TIMy->CCR1 = 25; /* (4) */
/* Enable the output of TIMx OC1 */
/* (1) Select active high polarity on OC1 (CC1P = 0, reset value),
   enable the output on OC1 (CC1E = 1) */
/* (2) Enable output (MOE = 1) */
TIMx->CCER |= TIM_CCER_CC2E; /* (1) */
TIMx->BDTR |= TIM_BDTR_MOE; /* (2) */
/* Enable the output of TIMy OC1 */
/* (1) Select active high polarity on OC1 (CC1P = 0, reset value),
   enable the output on OC1 (CC1E = 1) */
/* (2) Enable output (MOE = 1) */
TIMy->CCER |= TIM_CCER_CC1E; /* (1) */
TIMy->BDTR |= TIM_BDTR_MOE; /* (2) */

```

### A.9.21 DMA burst feature code example

```

/* In this example TIMx has been previously configured
   in PWM center-aligned */
/* Configure DMA Burst Feature */
/* Configure the corresponding DMA channel */
/* (1) Set DMA channel peripheral address is the DMAR register address */
/* (2) Set DMA channel memory address is the address of the buffer
       in the RAM containing the data to be transferred by DMA
       into CCRx registers */
/* (3) Set the number of data transfer to sizeof(Duty_Cycle_Table) */
/* (4) Configure DMA transfer in CCR register,
       enable the circular mode by setting CIRC bit (optional),
       set memory size to 16_bits MSIZE = 01,
       set peripheral size to 32_bits PSIZE = 10,
       enable memory increment mode by setting MINC,
       set data transfer direction read from memory by setting DIR. */
/* (5) Configure TIMx_DCR register with DBL = 3 transfers
       and DBA = (@TIMx->CCR2 - @TIMx->CR1) >> 2 = 0xE */
/* (6) Enable the TIMx update DMA request by setting UDE bit in DIER
       register */
/* (7) Enable TIMx */
/* (8) Enable DMA channel */
DMA1_Channel2->CPAR = (uint32_t)(&(TIMx->DMAR)); /* (1) */
DMA1_Channel2->CMAR = (uint32_t)(Duty_Cycle_Table); /* (2) */
DMA1_Channel2->CNDTR = 10*3; /* (3) */
DMA1_Channel2->CCR |= DMA_CCR_CIRC | DMA_CCR_MSIZE_0 | DMA_CCR_PSIZE_1
                      | DMA_CCR_MINC | DMA_CCR_DIR; /* (4) */
TIMx->DCR = (3 << 8)
            + (((uint32_t)(&TIMx->CCR2))
            - ((uint32_t)(&TIMx->CR1))) >> 2; /* (5) */
TIMx->DIER |= TIM_DIER_UDE; /* (6) */
TIMx->CR1 |= TIM_CR1_CEN; /* (7) */
DMA1_Channel2->CCR |= DMA_CCR_EN; /* (8) */

```

## A.10 IRTIM code example

### A.10.1 TIM16 and TIM17 configuration code example

```

/* The following configuration is for RC5 standard */
/* TIM16 is used for the enveloppe while TIM17 is used for the carrier */
#define TIM_ENV TIM16
#define TIM_CAR TIM17
/* (1) Enable the peripheral clocks of Timer 16 and 17 and SYSCFG */
/* (2) Enable the peripheral clock of GPIOB */
/* (3) Select alternate function mode on GPIOB pin 9 */
/* (4) Select AF0 on PB9 in AFRH for IR_OUT (reset value) */
/* (5) Enable the high sink driver capability by setting I2C_PB9_FM+ bit
   in SYSCFG_CFR1 */
RCC->APB2ENR |= RCC_APB2ENR_TIM16EN | RCC_APB2ENR_TIM17EN
    | RCC_APB2ENR_SYSCFGCOMPEN; /* (1) */
RCC->AHBENR |= RCC_AHBENR_GPIOBEN; /* (2) */
GPIOB->MODER = (GPIOB->MODER & ~GPIO_MODER_MODER9)
    | GPIO_MODER_MODER9_1; /* (3) */
GPIOB->AFR[1] &= ~(0x0F << ((9 - 8) * 4)); /* (4) */
SYSCFG->CFGR1 |= SYSCFG_CFGR1_I2C_FMP_PB9; /* (5) */
/* Configure TIM_CAR as carrier signal */
/* (1) Set prescaler to 1, so APBCLK i.e 48MHz */
/* (2) Set ARR = 1333, as timer clock is 48MHz the frequency is 36kHz */
/* (3) Set CCRx = 1333/4, , the signal will bhave a 25% duty cycle */
/* (4) Select PWM mode 1 on OC1 (OC1M = 110),
   enable preload register on OC1 (OC1PE = 1) */
/* (5) Select active high polarity on OC1 (CC1P = 0, reset value),
   enable the output on OC1 (CC1E = 1) */
/* (6) Enable output (MOE = 1) */
TIM_CAR->PSC = v; /* (1) */
TIM_CAR->ARR = 1333; /* (2) */
TIM_CAR->CCR1 = (uint16_t)(1333 / 4); /* (3) */
TIM_CAR->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1
    | TIM_CCMR1_OC1PE; /* (4) */
TIM_CAR->CCER |= TIM_CCER_CC1E; /* (5) */
TIM_CAR->BDTR |= TIM_BDTR_MOE; /* (6) */
/* Configure TIM_ENV is the modulation enveloppe */
/* (1) Set prescaler to 1, so APBCLK i.e 48MHz */
/* (2) Set ARR = 42627, as timer clock is 48MHz the period is 888 us */
/* (3) Select Forced inactive on OC1 (OC1M = 100) */
/* (4) Select active high polarity on OC1 (CC1P = 0, reset value),
   enable the output on OC1 (CC1E = 1) */
/* (5) Enable output (MOE = 1) */
/* (6) Enable Update interrupt (UIE = 1) */
TIM_ENV->PSC = 0; /* (1) */
TIM_ENV->ARR = 42627; /* (2) */
TIM_ENV->CCMR1 |= TIM_CCMR1_OC1M_2; /* (3) */
TIM_ENV->CCER |= TIM_CCER_CC1E; /* (4) */
TIM_ENV->BDTR |= TIM_BDTR_MOE; /* (5) */
TIM_ENV->DIER |= TIM_DIER_UIE; /* (6) */
/* Enable and reset TIM_CAR only */
/* (1) Enable counter (CEN = 1),
   select edge aligned mode (CMS = 00, reset value),
   select direction as upcounter (DIR = 0, reset value) */

```

```

/* (2) Force update generation (UG = 1) */
TIM_CAR->CR1 |= TIM_CR1_CEN; /* (1) */
TIM_CAR->EGR |= TIM_EGR_UG; /* (2) */
/* Configure TIM_ENV interrupt */
/* (1) Enable Interrupt on TIM_ENV */
/* (2) Set priority for TIM_ENV */
NVIC_EnableIRQ(TIM_ENV IRQn); /* (1) */
NVIC_SetPriority(TIM_ENV IRQn,0); /* (2) */

```

## A.10.2 IRQHandler for IRTIM code example

```

/**
 * Description: This function handles TIM_16 interrupt request.
 * This interrupt subroutine computes the laps between 2
 * rising edges on T1IC.
 * This laps is stored in the "Counter" variable.
 */
void TIM16_IRQHandler(void)
{
    uint8_t bit_msg = 0;

    if ((SendOperationReady == 1)
        && (BitsSentCounter < (RC5_GlobalFrameLength * 2)))
    {
        if (BitsSentCounter < 32)
        {
            SendOperationCompleted = 0x00;
            bit_msg = (uint8_t)((ManchesterCodedMsg >> BitsSentCounter)& 1);

            if (bit_msg== 1)
            {
                /* Force active level - OC1REF is forced high */
                TIM_ENV->CCMR1 |= TIM_CCMR1_OC1M_0;
            }
            else
            {
                /* Force inactive level - OC1REF is forced low */
                TIM_ENV->CCMR1 &= (uint16_t)(~TIM_CCMR1_OC1M_0);
            }
        }
        BitsSentCounter++;
    }
    else
    {
        SendOperationCompleted = 0x01;
        SendOperationReady = 0;
        BitsSentCounter = 0;
    }
    /* Clear TIM_ENV update interrupt */
    TIM_ENV->SR &= (uint16_t)(~TIM_SR UIF);
}

```

## A.11 bxCAN code example

### A.11.1 bxCAN initialization mode code example

```

/* (1) Enter CAN init mode to write the configuration */
/* (2) Wait the init mode entering */
/* (3) Exit sleep mode */
/* (4) Loopback mode, set timing to 1Mb/s: BS1 = 4, BS2 = 3,
   prescaler = 6 */
/* (5) Leave init mode */
/* (6) Wait the init mode leaving */
/* (7) Enter filter init mode, (16-bit + mask, filter 0 for FIFO 0) */
/* (8) Acivate filter 0 */
/* (9) Set the Id and the mask (all bits of standard id care) */
/* (10) Leave filter init */
/* (11) Set FIFO0 message pending IT enable */
CAN->MCR |= CAN_MCR_INRQ; /* (1) */
while ((CAN->MSR & CAN_MSR_INAK) != CAN_MSR_INAK) /* (2) */
{
    /* add time out here for a robust application */
}
CAN->MCR &=~ CAN_MCR_SLEEP; /* (3) */
CAN->BTR |= CAN_BTR_LBKM | 2 << 20 | 3 << 16 | 5 << 0; /* (4) */
CAN->MCR &=~ CAN_MCR_INRQ; /* (5) */
while ((CAN->MSR & CAN_MSR_INAK) == CAN_MSR_INAK) /* (6) */
{
    /* add time out here for a robust application */
}
CAN->FMR |= CAN_FMR_INIT; /* (7) */
CAN->FA1R |= CAN_FA1R_FACT0; /* (8) */
CAN->sFilterRegister[0].FR1 = CAN_ID << 5 | 0xFF70U << 16; /* (9) */
CAN->FMR &=~ CAN_FMR_INIT; /* (10) */
CAN->IER |= CAN_IER_FMPIE0; /* (11) */

```

### A.11.2 bxCAN transmit code example

```

/* (1) check mailbox 0 is empty */
/* (2) fill data length = 1 */
/* (3) fill 8-bit data */
/* (4) fill Id field and request a transmission */
if ((CAN->TSR & CAN_TSR_TME0) == CAN_TSR_TME0) /* (1) */
{
    CAN->sTxMailBox[0].TDTR = 1; /* (2) */
    CAN->sTxMailBox[0].TDLR = CMD; /* (3) */
    CAN->sTxMailBox[0].TIR = (uint32_t)(CAN_ID << 21
                                         | CAN_TI0R_TXRQ); /* (4) */
}

```

### A.11.3 bxCAN receive code example

```

/* check if a message is filtered and received by FIFO 0 */
if ((CAN->RF0R & CAN_RF0R_FMP0)!=0)
{
    CAN_ReceiveMessage = CAN->sFIFOMailBox[0].RDLR; /* read data */
    CAN->RF0R |= CAN_RF0R_RFOM0; /* release FIFO */
    if ((CAN_ReceiveMessage & 0xFF) == CMD)
    {
        /* Process */
    }
}

```

## A.12 DBG code example

### A.12.1 DBG read device ID code example

```

/* Read MCU Id, 32-bit access */
MCU_Id = DBGMCU->IDCODE;

```

### A.12.2 DBG debug in Low-power mode code example

```

/* To be able to debug in stop mode */
DBGMCU->CR |= DBGMCU_CR_DBG_STOP;

```

## A.13 HDMI-CEC code example

### A.13.1 HDMI-CEC configure CEC code example

```

/* (1) OAR = 0x0001 => OA = 0x0 */
/* (2) Receive byte interrupt enable, receive end interrupt enable */
/* (3) CEC enable */
CEC->CFG_R = (0x001<<16); /* (1) */
CEC->IER = CEC_IER_RXBRIE|CEC_IER_RXENDIE; /* (2) */
CEC->CR = CEC_CR_CECEN; /* (3) */

```

### A.13.2 HDMI-CEC transmission with interrupt enabled code example

```

/* (1) Set transmit IT */
/* (2) Fill transmit data register with nibbles (initiator 0x0, destination
   0x1) */
/* (3) Set start of message bit */
CEC->IER |= CEC_IER_TXBRIE; /* (1) */
CEC->TXDR = (uint32_t)(ADDRESS_INITIATOR << 4
                       | ADDRESS_DESTINATION); /* (2) */
CEC->CR |= CEC_CR_TXSOM; /* (3) */

```

### A.13.3 HDMI-CEC interrupt management code example

```

if ((CEC->ISR & CEC_ISR_RXEND) == CEC_ISR_RXEND)
{
    CEC->ISR = CEC_ISR_RXBR | CEC_ISR_RXEND; /* Reset flag */
    Received_Data = CEC->RXDR;
    if (Received_Data == CMD)
    {
        /* Process */
    }
}
else if ((CEC->ISR & CEC_ISR_RXBR) == CEC_ISR_RXBR)
{
    CEC->ISR = CEC_ISR_RXBR; /* Reset flag */
    Received_Data = CEC->RXDR;
    /* Process */
}
else if ((CEC->ISR & CEC_ISR_TXBR) == CEC_ISR_TXBR)
{
    CEC->IER &= ~CEC_IER_TXBRIE; /* Reset Tx IT */
    CEC->CR |= CEC_CR_TXEOM; /* this is the last byte */
    CEC->TXDR = CMD;
}

```

## A.14 I2C code example

### A.14.1 I2C configured in master mode to receive code example

```

/* (1) Timing register value is computed with the AN4235 xls file,
   fast Mode @400kHz with I2CCLK = 48MHz, rise time = 140ns,
   fall time = 40ns */
/* (2) Periph enable, receive interrupt enable */
/* (3) Slave address = 0x5A, read transfer, 1 byte to receive, autoend */
I2C2->TIMINGR = (uint32_t)0x00B01A4B; /* (1) */
I2C2->CR1 = I2C_CR1_PE | I2C_CR1_RXIE; /* (2) */
I2C2->CR2 = I2C_CR2_AUTOEND | (1<<16) | I2C_CR2_RD_WRN
             | (I2C1_OWN_ADDRESS << 1); /* (3) */

```

### A.14.2 I2C configured in master mode to transmit code example

```

/* (1) Timing register value is computed with the AN4235 xls file,
   fast Mode @400kHz with I2CCLK = 48MHz, rise time = 140ns,
   fall time = 40ns */
/* (2) Periph enable */
/* (3) Slave address = 0x5A, write transfer, 1 byte to transmit, autoend */
I2C2->TIMINGR = (uint32_t)0x00B01A4B; /* (1) */
I2C2->CR1 = I2C_CR1_PE; /* (2) */
I2C2->CR2 = I2C_CR2_AUTOEND | (1 << 16) | (I2C1_OWN_ADDRESS << 1); /* (3) */

```

### A.14.3 I2C configured in slave mode code example

```

/* (1) Timing register value is computed with the AN4235 xls file,
   fast Mode @400kHz with I2CCLK = 48MHz, rise time = 140ns,
   fall time = 40ns */
/* (2) Periph enable, address match interrupt enable */
/* (3) 7-bit address = 0x5A */
/* (4) Enable own address 1 */
I2C1->TIMINGR = (uint32_t)0x00B00000; /* (1) */
I2C1->CR1 = I2C_CR1_PE | I2C_CR1_ADDRIE; /* (2) */
I2C1->OAR1 |= (uint32_t)(I2C1_OWN_ADDRESS << 1); /* (3) */
I2C1->OAR1 |= I2C_OAR1_OA1EN; /* (4) */

```

### A.14.4 I2C master transmitter code example

```

/* Check Tx empty */
if ((I2C2->ISR & I2C_ISR_TXE) == I2C_ISR_TXE)
{
    I2C2->TXDR = I2C_BYTE_TO_SEND; /* Byte to send */
    I2C2->CR2 |= I2C_CR2_START; /* Go */
}

```

### A.14.5 I2C master receiver code example

```

if ((I2C2->ISR & I2C_ISR_RXNE) == I2C_ISR_RXNE)
{
    /* Read receive register, will clear RXNE flag */
    if (I2C2->RXDR == I2C_BYTE_TO_SEND)
    {
        /* Process */
    }
}

```

### A.14.6 I2C slave transmitter code example

```

uint32_t I2C_InterruptStatus = I2C1->ISR; /* Get interrupt status */
/* Check address match */
if ((I2C_InterruptStatus & I2C_ISR_ADDR) == I2C_ISR_ADDR)
{
    I2C1->ICR |= I2C_ICR_ADDRCF; /* Clear address match flag */
    /* Check if transfer direction is read (slave transmitter) */
    if ((I2C1->ISR & I2C_ISR_DIR) == I2C_ISR_DIR)
    {
        I2C1->CR1 |= I2C_CR1_TXIE; /* Set transmit IT */
    }
}
else if ((I2C_InterruptStatus & I2C_ISR_TXIS) == I2C_ISR_TXIS)
{
    I2C1->CR1 &= ~I2C_CR1_TXIE; /* Disable transmit IT */
    I2C1->TXDR = I2C_BYTE_TO_SEND; /* Byte to send */
}

```

### A.14.7 I2C slave receiver code example

```

uint32_t I2C_InterruptStatus = I2C1->ISR; /* Get interrupt status */
if ((I2C_InterruptStatus & I2C_ISR_ADDR) == I2C_ISR_ADDR)
{
    I2C1->ICR |= I2C_ICR_ADDRCF; /* Address match event */
}
else if ((I2C_InterruptStatus & I2C_ISR_RXNE) == I2C_ISR_RXNE)
{
    /* Read receive register, will clear RXNE flag */
    if (I2C1->RXDR == I2C_BYTE_TO_SEND)
    {
        /* Process */
    }
}

```

### A.14.8 I2C configured in master mode to transmit with DMA code example

```

/* (1) Timing register value is computed with the AN4235 xls file,
   fast Mode @400kHz with I2CCLK = 48MHz, rise time = 140ns,
   fall time = 40ns */
/* (2) Periph enable */
/* (3) Slave address = 0x5A, write transfer, 2 bytes to transmit,
   autoend */
I2C2->TIMINGR = (uint32_t)0x00B01A4B; /* (1) */
I2C2->CR1 = I2C_CR1_PE | I2C_CR1_TXDMAEN; /* (2) */
I2C2->CR2 = I2C_CR2_AUTOEND | (SIZE_OF_DATA << 16)
            | (I2C1_OWN_ADDRESS << 1); /* (3) */

```

### A.14.9 I2C configured in slave mode to receive with DMA code example

```

/* (1) Timing register value is computed with the AN4235 xls file,
   fast Mode @400kHz with I2CCLK = 48MHz, rise time = 140ns,
   fall time = 40ns */
/* (2) Periph enable, receive DMA enable */
/* (3) 7-bit address = 0x5A */
/* (4) Enable own address 1 */
I2C1->TIMINGR = (uint32_t)0x00B00000; /* (1) */
I2C1->CR1 = I2C_CR1_PE | I2C_CR1_RXDMAEN | I2C_CR1_ADDRIE; /* (2) */
I2C1->OAR1 |= (uint32_t)(I2C1_OWN_ADDRESS << 1); /* (3) */
I2C1->OAR1 |= I2C_OAR1_OA1EN; /* (4) */

```

## A.15 IWDG code example

### A.15.1 IWDG configuration code example

```

/* (1) Activate IWDG (not needed if done in option bytes) */
/* (2) Enable write access to IWDG registers */
/* (3) Set prescaler by 8 */
/* (4) Set reload value to have a rollover each 100ms */
/* (5) Check if flags are reset */
/* (6) Refresh counter */
IWDG->KR = IWDG_START; /* (1) */
IWDG->KR = IWDG_WRITE_ACCESS; /* (2) */
IWDG->PR = IWDG_PR_PR_0; /* (3) */
IWDG->RLR = IWDG_RELOAD; /* (4) */
while (IWDG->SR) /* (5) */
{
    /* add time out here for a robust application */
}
IWDG->KR = IWDG_REFRESH; /* (6) */

```

## A.15.2 IWDG configuration with window code example

```
/* (1) Activate IWDG (not needed if done in option bytes) */
/* (2) Enable write access to IWDG registers */
/* (3) Set prescaler by 8 */
/* (4) Set reload value to have a rollover each 100ms */
/* (5) Check if flags are reset */
/* (6) Set a 50ms window, this will refresh the IWDG */
IWDG->KR = IWDG_START; /* (1) */
IWDG->KR = IWDG_WRITE_ACCESS; /* (2) */
IWDG->PR = IWDG_PR_PR_0; /* (3) */
IWDG->RLR = IWDG_RELOAD; /* (4) */
while (IWDG->SR) /* (5) */
{
    /* add time out here for a robust application */
}
IWDG->WINR = IWDG_RELOAD >> 1; /* (6) */
```

# A.16 RTC code example

## A.16.1 RTC calendar configuration code example

```
/* (1) Write access for RTC registers */
/* (2) Enable init phase */
/* (3) Wait until it is allow to modify RTC register values */
/* (4) set prescaler, 40kHz/128 => 312 Hz, 312Hz/312 => 1Hz */
/* (5) New time in TR */
/* (6) Disable init phase */
/* (7) Disable write access for RTC registers */
RTC->WPR = 0xCA; /* (1) */
RTC->WPR = 0x53; /* (1) */
RTC->ISR |= RTC_ISR_INIT; /* (2) */
while ((RTC->ISR & RTC_ISR_INITF) != RTC_ISR_INITF) /* (3) */
{
    /* add time out here for a robust application */
}
RTC->PRER = 0x007F0137; /* (4) */
RTC->TR = RTC_TR_PM | Time; /* (5) */
RTC->ISR &= ~RTC_ISR_INIT; /* (6) */
RTC->WPR = 0xFE; /* (7) */
RTC->WPR = 0x64; /* (7) */
```

### A.16.2 RTC alarm configuration code example

```

/* (1) Write access for RTC registers */
/* (2) Disable alarm A to modify it */
/* (3) Wait until it is allow to modify alarm A value */
/* (4) Modify alarm A mask to have an interrupt each 1Hz */
/* (5) Enable alarm A and alarm A interrupt */
/* (6) Disable write access */
RTC->WPR = 0xCA; /* (1) */
RTC->WPR = 0x53; /* (1) */
RTC->CR &= ~RTC_CR_ALRAE; /* (2) */
while ((RTC->ISR & RTC_ISR_ALRAWF) != RTC_ISR_ALRAWF) /* (3) */
{
    /* add time out here for a robust application */
}
RTC->ALRMAR = RTC_ALRMAR_MSK4 | RTC_ALRMAR_MSK3
    | RTC_ALRMAR_MSK2 | RTC_ALRMAR_MSK1; /* (4) */
RTC->CR = RTC_CR_ALRAIE | RTC_CR_ALRAE; /* (5) */
RTC->WPR = 0xFE; /* (6) */
RTC->WPR = 0x64; /* (6) */

```

### A.16.3 RTC WUT configuration code example

```

/* (1) Write access for RTC registers */
/* (2) Disable wake up timer to modify it */
/* (3) Wait until it is allow to modify wake up reload value */
/* (4) Modify wake up value reload counter to have a wake up each 1Hz */
/* (5) Enable wake up counter and wake up interrupt */
/* (6) Disable write access */
RTC->WPR = 0xCA; /* (1) */
RTC->WPR = 0x53; /* (1) */
RTC->CR &= ~RTC_CR_WUTE; /* (2) */
while ((RTC->ISR & RTC_ISR_WUTWF) != RTC_ISR_WUTWF) /* (3) */
{
    /* add time out here for a robust application */
}
RTC->WUTR = 0x9C0; /* (4) */
RTC->CR = RTC_CR_WUTE | RTC_CR_WUTIE; /* (5) */
RTC->WPR = 0xFE; /* (6) */
RTC->WPR = 0x64; /* (6) */

```

### A.16.4 RTC read calendar code example

```

if((RTC->ISR & RTC_ISR_RSF) == RTC_ISR_RSF)
{
    TimeToCompute = RTC->TR; /* get time */
    DateToCompute = RTC->DR; /* need to read date also */
}

```

### A.16.5 RTC calibration code example

```

/* (1) Write access for RTC registers */
/* (2) Enable init phase */
/* (3) Wait until it is allow to modify RTC register values */
/* (4) set prescaler, 40kHz/125 => 320 Hz, 320Hz/320 => 1Hz */
/* (5) New time in TR */
/* (6) Disable init phase */
/* (7) Wait until it's allow to modify calibartion register */
/* (8) Set calibration to around +20ppm, which is a standard value @25°C */
/* Note: the calibration is relevant when LSE is selected for RTC clock */
/* (9) Disable write access for RTC registers */

RTC->WPR = 0xCA; /* (1) */
RTC->WPR = 0x53; /* (1) */
RTC->ISR |= RTC_ISR_INIT; /* (2) */
while ((RTC->ISR & RTC_ISR_INITF) != RTC_ISR_INITF) /* (3) */
{
    /* add time out here for a robust application */
}
RTC->PRER = (124<<16) | 319; /* (4) */
RTC->TR = RTC_TR_PM | Time; /* (5) */
RTC->ISR &= ~RTC_ISR_INIT; /* (6) */
while((RTC->ISR & RTC_ISR_RECALPF) == RTC_ISR_RECALPF) /* (7) */
{
    /* add time out here for a robust application */
}
RTC->CALR = RTC_CALR_CALP | 482; /* (8) */
RTC->WPR = 0xFE; /* (9) */
RTC->WPR = 0x64; /* (9) */

```

### A.16.6 RTC tamper and time stamp configuration code example

```

/* Tamper configuration:
 - Disable precharge (PU)
 - RTCCLK/256 tamper sampling frequency
 - Activate time stamp on tamper detection
 - input rising edge trigger detection on RTC_TAMP2 (PA0)
 - Tamper interrupt enable */
RTC->TAFCR = RTC_TAFCR_TAMPPUDIS | RTC_TAFCR_TAMPFREQ | RTC_TAFCR_TAMPTS
| RTC_TAFCR_TAMP2E | RTC_TAFCR_TAMPIE;

```

## A.16.7 RTC tamper and time stamp code example

```

/* Check tamper and timestamp flag */
if (((RTC->ISR & (RTC_ISR_TAMP2F)) == (RTC_ISR_TAMP2F))
    && ((RTC->ISR & (RTC_ISR_TSF)) == (RTC_ISR_TSF)))
{
    RTC->ISR &= ~RTC_ISR_TAMP2F; /* clear tamper flag */
    EXTI->PR = EXTI_PR_PR19; /* clear exti line 19 flag */
    TimeToCompute = RTC->TSTR; /* get tamper time in timestamp register */
    RTC->ISR &= ~RTC_ISR_TSF; /* clear timestamp flag */
}

```

## A.16.8 RTC clock output code example

```

/* (1) Write access for RTC registers */
/* (2) Disable alarm A to modify it */
/* (3) Wait until it is allow to modify alarm A value */
/* (4) Modify alarm A mask to have an interrupt each 1Hz */
/* (5) Enable alarm A and alarm A interrupt,
   enable calibration output (1Hz) */
/* (6) Disable write access */
RTC->WPR = 0xCA; /* (1) */
RTC->WPR = 0x53; /* (1) */
RTC->CR &= ~RTC_CR_ALRAE; /* (2) */
while ((RTC->ISR & RTC_ISR_ALRAWF) != RTC_ISR_ALRAWF) /* (3) */
{
    /* add time out here for a robust application */
}
RTC->ALRMAR = RTC_ALRMAR_MSK4 | RTC_ALRMAR_MSK3
    | RTC_ALRMAR_MSK2 | RTC_ALRMAR_MSK1; /* (4) */
RTC->CR = RTC_CR_ALRAIE | RTC_CR_ALRAE | RTC_CR_COE
    | RTC_CR_COSEL; /* (5) */
RTC->WPR = 0xFE; /* (6) */
RTC->WPR = 0x64; /* (6) */

```

## A.17 SPI code example

### A.17.1 SPI master configuration code example

```

/* (1) Master selection, BR: Fpclk/256 (due to C27 on the board, SPI_CLK is
   set to the minimum) CPOL and CPHA at zero (rising first edge) */
/* (2) Slave select output enabled, RXNE IT, 8-bit Rx fifo */
/* (3) Enable SPI1 */
SPI1->CR1 = SPI_CR1_MSTR | SPI_CR1_BR; /* (1) */
SPI1->CR2 = SPI_CR2_SSOE | SPI_CR2_RXNEIE | SPI_CR2_FRXTH
    | SPI_CR2_DS_2 | SPI_CR2_DS_1 | SPI_CR2_DS_0; /* (2) */
SPI1->CR1 |= SPI_CR1_SPE; /* (3) */

```

### A.17.2 SPI slave configuration code example

```
/* nSS hard, slave, CPOL and CPHA at zero (rising first edge) */
/* (1) RXNE IT, 8-bit Rx fifo */
/* (2) Enable SPI2 */
SPI2->CR2 = SPI_CR2_RXNEIE | SPI_CR2_FRXTH
            | SPI_CR2_DS_2 | SPI_CR2_DS_1 | SPI_CR2_DS_0; /* (1) */
SPI2->CR1 |= SPI_CR1_SPE; /* (2) */
```

### A.17.3 SPI full duplex communication code example

```
if ((SPI1->SR & SPI_SR_TXE) == SPI_SR_TXE) /* Test Tx empty */
{
    /* Will initiate 8-bit transmission if TXE */
    *(uint8_t *)&(SPI1->DR) = SPI1_DATA;
}
```

### A.17.4 SPI interrupt code example

```
if ((SPI1->SR & SPI_SR_RXNE) == SPI_SR_RXNE)
{
    SPI1_Data = (uint8_t)SPI1->DR; /* receive data, clear flag */
    /* Process */
}
```

### A.17.5 SPI master configuration with DMA code example

```
/* (1) Master selection, BR: Fpclk/256 (due to C27 on the board, SPI_CLK is
   set to the minimum)
   CPOL and CPHA at zero (rising first edge) */
/* (2) TX and RX with DMA,
   enable slave select output,
   enable RXNE interrupt,
   select 8-bit Rx fifo */
/* (3) Enable SPI1 */
SPI1->CR1 = SPI_CR1_MSTR | SPI_CR1_BR; /* (1) */
SPI1->CR2 = SPI_CR2_TXDMAEN | SPI_CR2_RXDMAEN | SPI_CR2_SSOE
            | SPI_CR2_RXNEIE | SPI_CR2_FRXTH
            | SPI_CR2_DS_2 | SPI_CR2_DS_1 | SPI_CR2_DS_0; /* (2) */
SPI1->CR1 |= SPI_CR1_SPE; /* (3) */
```

## A.17.6 SPI slave configuration with DMA code example

```

/* nSS hard, slave, CPOL and CPHA at zero (rising first edge) */
/* (1) Select TX and RX with DMA,
   enable RXNE interrupt,
   select 8-bit Rx fifo */
/* (2) Enable SPI2 */
SPI2->CR2 = SPI_CR2_TXDMAEN | SPI_CR2_RXDMAEN
            | SPI_CR2_RXNEIE | SPI_CR2_FRXTH
            | SPI_CR2_DS_2 | SPI_CR2_DS_1 | SPI_CR2_DS_0; /* (1) */
SPI2->CR1 |= SPI_CR1_SPE; /* (2) */

```

## A.18 TSC code example

### A.18.1 TSC configuration code example

```

/* Configure TCS */
/* With a charge transfer around 2.5 µs */
/* (1) Select fPGCLK = fHCLK/32,
   Set pulse high = 2xtPGCLK,Master
   Set pulse low = 2xtPGCLK
   Set Max count value = 16383 pulses
   Enable TSC */
/* (2) Disable hysteresis */
/* (3) Enable end of acquisition IT */
/* (4) Sampling enabled, G2IO4 */
/* (5) Channel enabled, G2IO3 */
/* (6) Enable group, G2 */
TSC->CR = TSC_CR_PGPSC_2 | TSC_CR_PGPSC_0 | TSC_CR_CTPH_0 | TSC_CR_CTPL_0
          | TSC_CR_MCV_2 | TSC_CR_MCV_1 | TSC_CR_TSCE; /* (1) */
TSC->IOHCR &= (uint32_t) (~(TSC_IOHCR_G2_IO4 | TSC_IOHCR_G2_IO3)); /* (2) */
TSC->IER = TSC_IER_EOAIE; /* (3) */
TSC->IOSCR = TSC_IOSCR_G2_IO4; /* (4) */
TSC->IOCCR = TSC_IOCCR_G2_IO3; /* (5) */
TSC->IOGCSR |= TSC_IOGCSR_G2E; /* (5) */

```

### A.18.2 TSC interrupt code example

```

/* End of acquisition flag */
if ((TSC->ISR & TSC_ISR_EOAF) == TSC_ISR_EOAF)
{
    TSC->ICR = TSC_ICR_EOAIC; /* Clear flag */
    AcquisitionValue = TSC->IOGXCR[1]; /* Get G2 counter value */
}

```

## A.19 USART code example

### A.19.1 USART transmitter configuration code example

```
/* (1) Oversampling by 16, 9600 baud */
/* (2) 8 data bit, 1 start bit, 1 stop bit, no parity */
USART1->BRR = 480000 / 96; /* (1) */
USART1->CR1 = USART_CR1_TE | USART_CR1_UE; /* (2) */
```

### A.19.2 USART transmit byte code example

```
/* Start USART transmission */
USART1->TDR = stringtosend[send++]; /* Will initiate TC if TXE is set*/
```

### A.19.3 USART transfer complete code example

```
if ((USART1->ISR & USART_ISR_TC) == USART_ISR_TC)
{
    if (send == sizeof(stringtosend))
    {
        send=0;
        USART1->ICR |= USART_ICR_TCCF; /* Clear transfer complete flag */
    }
    else
    {
        /* clear transfer complete flag and fill TDR with a new char */
        USART1->TDR = stringtosend[send++];
    }
}
```

### A.19.4 USART receiver configuration code example

```
/* (1) oversampling by 16, 9600 baud */
/* (2) 8 data bit, 1 start bit, 1 stop bit, no parity, reception mode */
USART1->BRR = 480000 / 96; /* (1) */
USART1->CR1 = USART_CR1_RXNEIE | USART_CR1_RE | USART_CR1_UE; /* (2) */
```

### A.19.5 USART receive byte code example

```
if ((USART1->ISR & USART_ISR_RXNE) == USART_ISR_RXNE)
{
    chartoreceive = (uint8_t)(USART1->RDR); /* Receive data, clear flag */
}
```

## A.19.6 USART LIN mode code example

```

/* (1) oversampling by 16, 9600 baud */
/* (2) LIN mode */
/* (3) 8 data bit, 1 start bit, 1 stop bit, no parity,
   reception and transmission enabled */
USART1->BRR = 480000 / 96; /* (1) */
USART1->CR2 = USART_CR2_LINEN | USART_CR2_LBDIE; /* (2) */
USART1->CR1 = USART_CR1_TE | USART_CR1_RXNEIE
              | USART_CR1_RE | USART_CR1_UE; /* (3) */
/* Polling idle frame Transmission */
while ((USART1->ISR & USART_ISR_TC) != USART_ISR_TC)
{
    /* add time out here for a robust application */
}
USART1->ICR |= USART_ICR_TCCF; /* Clear TC flag */
USART1->CR1 |= USART_CR1_TCIE; /* Enable TC interrupt */

```

## A.19.7 USART synchronous mode code example

```

/* (1) Oversampling by 16, 9600 baud */
/* (2) Synchronous mode
   CPOL and CPHA = 0 => rising first edge
   Last bit clock pulse
   Most significant bit first in transmit/receive */
/* (3) 8 data bit, 1 start bit, 1 stop bit, no parity
   Transmission enabled, reception enabled */
USART1->BRR = 480000 / 96; /* (1) */
USART1->CR2 = USART_CR2_MSBFIRST | USART_CR2_CLKEN
              | USART_CR2_LBCL; /* (2) */
USART1->CR1 = USART_CR1_TE | USART_CR1_RXNEIE
              | USART_CR1_RE | USART_CR1_UE; /* (3) */
/* Polling idle frame Transmission w/o clock */
while ((USART1->ISR & USART_ISR_TC) != USART_ISR_TC)
{
    /* add time out here for a robust application */
}
USART1->ICR |= USART_ICR_TCCF; /* Clear TC flag */
USART1->CR1 |= USART_CR1_TCIE; /* Enable TC interrupt */

```

### A.19.8 USART single-wire half-duplex code example

```

/* (1) Oversampling by 16, 9600 baud */
/* (2) Single-wire half-duplex mode */
/* (3) 8 data bit, 1 start bit, 1 stop bit, no parity, reception and
transmission enabled */
USART1->BRR = 480000 / 96; /* (1) */
USART1->CR3 = USART_CR3_HDSEL; /* (2) */
USART1->CR1 = USART_CR1_TE | USART_CR1_RXNEIE
| USART_CR1_RE | USART_CR1_UE; /* (3) */
/* Polling idle frame Transmission */
while ((USART1->ISR & USART_ISR_TC) != USART_ISR_TC)
{
    /* add time out here for a robust application */
}
USART1->ICR |= USART_ICR_TCCF; /* Clear TC flag */
USART1->CR1 |= USART_CR1_TCIE; /* Enable TC interrupt */

```

### A.19.9 USART smartcard mode code example

```

/* (1) Oversampling by 16, 9600 baud */
/* (2) Clock divided by 16 = 3MHz */
/* (3) Smart card mode enable */
/* (4) 1.5 stop bits, clock enable */
/* (5) 8-data bit plus parity, 1 start bit */
USART1->BRR = 480000 / 96; /* (1) */
USART1->GTPR = 16 >> 1; /* (2) */
USART1->CR3 = USART_CR3_SCEN; /* (3) */
USART1->CR2 = USART_CR2_STOP_1 | USART_CR2_STOP_0
| USART_CR2_CLKEN; /* (4) */
USART1->CR1 = USART_CR1_M | USART_CR1_PCE
| USART_CR1_TE | USART_CR1_UE; /* (5) */
/* Polling idle frame transmission transfer complete
(this frame is not sent) */
while ((USART1->ISR & USART_ISR_TC) != USART_ISR_TC)
{
    /* add time out here for a robust application */
}
USART1->ICR |= USART_ICR_TCCF; /* Clear TC flag */
USART1->CR1 |= USART_CR1_TCIE; /* Enable TC interrupt */

```

### A.19.10 USART IrDA mode code example

```
/* (1) Oversampling by 16, 9600 baud */
/* (2) Divide by 24 to achieve the low power frequency */
/* (3) Enable IrDA */
/* (4) 8 data bit, 1 start bit, 1 stop bit, no parity */
USART1->BRR = 480000 / 96; /* (1) */
USART1->GTPR = 24; /* (2) */
USART1->CR3 = USART_CR3_IREN; /* (3) */
USART1->CR1 = USART_CR1_TE | USART_CR1_UE; /* (4) */
/* Polling idle frame Transmission */
while((USART1->ISR & USART_ISR_TC) != USART_ISR_TC)
{
    /* add time out here for a robust application */
}
USART1->ICR |= USART_ICR_TCCF; /* Clear TC flag */
USART1->CR1 |= USART_CR1_TCIE; /* Enable TC interrupt */
```

### A.19.11 USART DMA code example

```
/* (1) Oversampling by 16, 9600 baud */
/* (2) Enable DMA in reception and transmission */
/* (3) 8 data bit, 1 start bit, 1 stop bit, no parity, reception and
transmission enabled */
USART1->BRR = 480000 / 96; /* (1) */
USART1->CR3 = USART_CR3_DMAT | USART_CR3_DMAR; /* (2) */
USART1->CR1 = USART_CR1_TE | USART_CR1_RE | USART_CR1_UE; /* (3) */
/* Polling idle frame Transmission */
while ((USART1->ISR & USART_ISR_TC) != USART_ISR_TC)
{
    /* add time out here for a robust application */
}
USART1->ICR |= USART_ICR_TCCF; /* Clear TC flag */
USART1->CR1 |= USART_CR1_TCIE; /* Enable TC interrupt */
```

### A.19.12 USART hardware flow control code example

```
/* (1) oversampling by 16, 9600 baud */
/* (2) RTS and CTS enabled */
/* (3) 8 data bit, 1 start bit, 1 stop bit, no parity, reception and
transmission enabled */
USART1->BRR = 480000 / 96; /* (1) */
USART1->CR3 = USART_CR3_RTSE | USART_CR3_CTSE; /* (2) */
USART1->CR1 = USART_CR1_TE | USART_CR1_RXNEIE
| USART_CR1_RE | USART_CR1_UE; /* (3) */
/* Polling idle frame Transmission */
while ((USART1->ISR & USART_ISR_TC) != USART_ISR_TC)
{
    /* add time out here for a robust application */
}
USART1->ICR |= USART_ICR_TCCF; /* Clear TC flag */
USART1->CR1 |= USART_CR1_TCIE; /* Enable TC interrupt */
```

## A.20 WWDG code example

### A.20.1 WWDG configuration code example

```
/* (1) Set prescaler to have a roll-over each about 5.5ms,
set window value (about 2.25ms) */
/* (2) Refresh WWDG before activate it */
/* (3) Activate WWDG */
WWDG->CFR = 0x60; /* (1) */
WWDG->CR = WWDG_REFRESH; /* (2) */
WWDG->CR |= WWDG_CR_WDGA; /* (3) */
```

## Important security notice

The STMicroelectronics group of companies (ST) places a high value on product security, which is why the ST product(s) identified in this documentation may be certified by various security certification bodies and/or may implement our own security measures as set forth herein. However, no level of security certification and/or built-in security measures can guarantee that ST products are resistant to all forms of attacks. As such, it is the responsibility of each of ST's customers to determine if the level of security provided in an ST product meets the customer needs both in relation to the ST product alone, as well as when combined with other components and/or software for the customer end product or application. In particular, take note that:

- ST products may have been certified by one or more security certification bodies, such as Platform Security Architecture ([www.psacertified.org](http://www.psacertified.org)) and/or Security Evaluation standard for IoT Platforms ([www.trustcb.com](http://www.trustcb.com)). For details concerning whether the ST product(s) referenced herein have received security certification along with the level and current status of such certification, either visit the relevant certification standards website or go to the relevant product page on [www.st.com](http://www.st.com) for the most up to date information. As the status and/or level of security certification for an ST product can change from time to time, customers should re-check security certification status/level as needed. If an ST product is not shown to be certified under a particular security standard, customers should not assume it is certified.
- Certification bodies have the right to evaluate, grant and revoke security certification in relation to ST products. These certification bodies are therefore independently responsible for granting or revoking security certification for an ST product, and ST does not take any responsibility for mistakes, evaluations, assessments, testing, or other activity carried out by the certification body with respect to any ST product.
- Industry-based cryptographic algorithms (such as AES, DES, or MD5) and other open standard technologies which may be used in conjunction with an ST product are based on standards which were not developed by ST. ST does not take responsibility for any flaws in such cryptographic algorithms or open technologies or for any methods which have been or may be developed to bypass, decrypt or crack such algorithms or technologies.
- While robust security testing may be done, no level of certification can absolutely guarantee protections against all attacks, including, for example, against advanced attacks which have not been tested for, against new or unidentified forms of attack, or against any form of attack when using an ST product outside of its specification or intended use, or in conjunction with other components or software which are used by customer to create their end product or application. ST is not responsible for resistance against such attacks. As such, regardless of the incorporated security features and/or any information or support that may be provided by ST, each customer is solely responsible for determining if the level of attacks tested for meets their needs, both in relation to the ST product alone and when incorporated into a customer end product or application.
- All security features of ST products (inclusive of any hardware, software, documentation, and the like), including but not limited to any enhanced security features added by ST, are provided on an "AS IS" BASIS. AS SUCH, TO THE EXTENT PERMITTED BY APPLICABLE LAW, ST DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, unless the applicable written and signed contract terms specifically provide otherwise.

## Revision history

**Table 146. Document revision history**

Date	Revision	Changes
06-Apr-2012	1	<p>Initial release</p>
08-Aug-2012	2	<p><b>Documentation conventions</b></p> <ul style="list-style-type: none"> <li>– Added ‘Always read...’ in <i>Section 1.1: List of abbreviations for registers</i></li> </ul> <p><b>Memory map</b></p> <ul style="list-style-type: none"> <li>– Added boot configuration link in <i>Table 1: STM32F05xxx memory map and peripheral register boundary addresses</i></li> <li>– Added Footnote under <i>Table 3: Boot modes</i></li> <li>– Remove ‘DCode’ after ‘CPU’ in <i>Section : DMA bus</i></li> <li>– Replaced sentence and added Physical remap paragraph in <i>Section 2.5: Boot configuration</i></li> <li>– Corrected Prefetch buffer and added paragraph in <i>Section 1.2.2: Read operations</i></li> <li>– Removed sentence ‘It is set after the first...’, removed note and added description of bit 5 in <i>Section 1.5.1: Flash access control register (FLASH_ACR)</i></li> </ul> <p><b>CRC</b></p> <ul style="list-style-type: none"> <li>– Replaced references to bits 4:3 and 2:1 in <i>Section 6.4.3: Control register (CRC_CR)</i></li> </ul> <p><b>PWR</b></p> <ul style="list-style-type: none"> <li>– Replaced ‘POR’, ‘PDR’ and ‘PWD’ by ‘VPOR’, ‘VPDR’ and ‘VPVD’ in <i>Figure 2</i> and <i>Figure 3</i></li> <li>– Modified “When VDDA is different from VDD...” paragraph in <i>Section 1.1.1: Independent A/D and D/A converter supply and reference voltage</i></li> <li>– Moved arrow in <i>Figure 2: Power on reset/power down reset waveform</i></li> <li>– Corrected WKUP1 in <i>Section 1.4.2: Power control/status register (PWR_CSR)</i></li> </ul> <p><b>RCC</b></p> <ul style="list-style-type: none"> <li>– Added Power reset in <i>Section 1.1.2: System reset</i></li> <li>– Added paragraph ‘For more details on how to measure..’ in <i>Section 1.2.2: HSI clock</i></li> <li>– Added ‘LSE’ and ‘LSI’ bullets in <i>Section 1.2.12: Clock-out capability</i></li> <li>– Added ‘/1’, ‘LSI’ and ‘LSE’ in <i>Figure 2: Clock tree (STM32F03x and STM32F05x devices)</i></li> <li>– Added the sentence “... drive the HSI clock ...” to <i>Section 1.2.2: HSI clock</i></li> <li>– Added “independent” to the title of <i>Section 1.2.11: Independent watchdog clock</i></li> <li>– Replaced the first sentence after <i>Figure 5: Frequency measurement with TIM14 in capture mode</i></li> <li>– Added the sentence “When the system is in stop mode...” in <i>Section : The input capture channel of the Timer 14 can be a GPIO line or an internal clock of the MCU. This selection is performed through the TI1_RMP [1:0] bits in the TIM14_OR register. The possibilities available are the following ones.</i></li> <li>– Added <i>Section 1.2.13: Internal/external clock measurement with TIM14</i> and <i>Section 1.3: Low-power modes</i></li> </ul>

**Table 146. Document revision history (continued)**

Date	Revision	Changes
08-Aug-2012	2 (cont'd)	<ul style="list-style-type: none"> <li>– Modified description of Bit 19 and Bits 15:8 in <i>Section 1.4.1: Clock control register (RCC_CR)</i></li> <li>– Replaced text under ‘PLLXTPRE’ and modified description of bits 26:24 in <i>Section 1.4.2: Clock configuration register (RCC_CGCR)</i></li> <li>– Modified the formating in the register of <i>Section 1.4.4</i> and <i>Section 1.4.5: APB peripheral reset register 1 (RCC_APB1RSTR)</i></li> <li>– Modified title of Bit 0 in the registers of <i>Section 1.4.7: APB peripheral clock enable register 2 (RCC_APB2ENR)</i></li> <li>– Replaced ‘IWWDRSTF’ by ‘IWDGRSTF’ and added PORV18RSTDF POR/PDR for bit 23 in <i>Section 1.4.10: Control/status register (RCC_CSR)</i></li> <li>– Modified <i>Section 1.4.4: APB peripheral reset register 2 (RCC_APB2RSTR)</i> to <i>Section 1.4.8: APB peripheral clock enable register 1 (RCC_APB1ENR)</i> title name.</li> </ul> <p><b>GPIO</b></p> <ul style="list-style-type: none"> <li>– Replaced SWDAT in <i>Section 9.3.1: General-purpose I/O (GPIO)</i></li> <li>– Added specific reset values in <i>Section 9.4.3: GPIO port output speed register (GPIOx_OSPEEDR)</i> (<math>x = A..F</math>)</li> </ul> <p><b>DMA</b></p> <ul style="list-style-type: none"> <li>– Added ‘I2C1’, ‘TIM15’, ‘TIM16 and ‘TIM17’ to <i>Figure 21: DMA block diagram</i></li> </ul> <p><b>ADC</b></p> <ul style="list-style-type: none"> <li>– Changed JIITOFF_D2 and JIITOFF_D4 to JIITOFF_DIV4 and JIITOFF_DIV2 in <i>Section 13.12.5: ADC configuration register 2 (ADC_CFGR2)</i> and <i>Section 13.12.11: ADC register map</i></li> <li>– Replaced ‘SMPR’ with ‘SMP’ in <i>Section 13.12.11: ADC register map</i></li> </ul> <p><b>DAC</b></p> <ul style="list-style-type: none"> <li>– Replaced note in <i>Section 14.2: DAC1 main features</i></li> </ul> <p><b>COMP</b></p> <ul style="list-style-type: none"> <li>– Replaced ‘bandgap’ with ‘V<sub>REFINT</sub>’ and added PA12 to COMP2_OUT in <i>Figure 92: Comparator 1 and 2 block diagrams</i></li> <li>– Added the sentence “Reset and clock enable bits....” to <i>Section 15.3.3: COMP reset and clocks</i></li> <li>– Modified COMP2MODE and COMP1MODE bit description in <i>Section 15.6.1: COMP control and status register (COMP_CSR)</i></li> </ul> <p><b>Timers</b></p> <ul style="list-style-type: none"> <li>– Added OCCS bit in <i>Section 1.4.3: TIM2 and TIM3 slave mode control register (TIM2_SMCR and TIM3_SMCR)</i> and <i>Section 1.4.3: TIM1 slave mode control register (TIM1_SMCR)</i></li> <li>– Mapped ‘TI1_RMP’ to bit 0 and 1 in <i>Section 1.4.10: TIM14 capture/compare register 1 (TIM14_CCR1)</i></li> </ul> <p><b>Infrared</b></p> <ul style="list-style-type: none"> <li>– Infrared function output changed to IR_OUT</li> </ul> <p><b>RTC</b></p> <ul style="list-style-type: none"> <li>– Removed V<sub>DD</sub> in <i>Section 25.6.19: RTC backup registers (RTC_BKPxR)</i></li> <li>– Removed Tamper3 in RTC ISR and TAFCR registers</li> <li>– RTC ordered before WDG</li> </ul> <p><b>IWDG</b></p> <ul style="list-style-type: none"> <li>– Replaced ‘IWWDG’ occurrences by ‘IWDG’</li> </ul>

**Table 146. Document revision history (continued)**

Date	Revision	Changes
08-Aug-2012	2 (cont'd)	<p><b>I2C</b></p> <ul style="list-style-type: none"> <li>– Modified <i>Section 24.4.5: I2C initialization</i>, <i>Section 24.4.6: Software reset</i></li> <li>– Removed ‘SWRST’ in <i>Section 24.7.1: Control register 1 (I2Cx_CR1)</i> and <i>Section 24.7.7: Interrupt and Status register (I2Cx_ISR)</i></li> </ul> <p><b>USART</b></p> <ul style="list-style-type: none"> <li>– Removed “with a 12-bit mantissa and 4-bit fraction” in <i>Section 25.5: USART functional description</i></li> <li>– Changed Equation in <i>Section 25.5.5: Tolerance of the USART receiver to clock deviation</i></li> <li>– Changed reset values length in <i>Section 25.7: USART registers</i></li> <li>– Corrected reset values for ‘DEP’, ‘TC’, ‘RDR’ and ‘TDR’ in <i>Table 90: USART register map and reset values</i></li> </ul> <p><b>Debug</b></p> <ul style="list-style-type: none"> <li>– Removed ‘Control of the trace..’ bullet in <i>Section 1.9: MCU debug component (DBGMCU)</i></li> <li>– Replaced IDCODE values in <i>Section 1.5.3: SW-DP state machine (reset, idle states, ID code)</i> and <i>Section 1.5.5: SW-DP registers</i></li> <li>– Replaced “APB low” to APB1 in <i>Section 1.9.2: Debug support for timers, watchdog and I<sup>2</sup>C</i></li> <li>– Corrected <i>Figure 1: Block diagram of STM32F0xx MCU and Cortex<sup>®</sup>-M0-level debug support</i></li> <li>– Modified the title of <i>Section 1.3.1: SWD port pins</i></li> <li>– Modified <i>Section 1.3.2: SW-DP pin assignment</i> and <i>Section 1.3.3: Internal pull-up &amp; pull-down on SWD pins</i></li> <li>– Modified titles in <i>Section 1.5: SWD port</i> and replaced ‘SWDAT’</li> <li>– Removed ‘DBG SLEEP’ in <i>Section 1.9.3: Debug MCU configuration register (DBGMCU_CR)</i></li> <li>– Added ‘System’ to window watchdog in <i>Section 1.9.4: Debug MCU APB1 freeze register (DBGMCU_APB1_FZ)</i></li> <li>– Marked bit 0, 5, 6 and 7 as reserved in <i>Section 1.9.6: DBG register map</i></li> </ul> <p><b>Unique ID</b></p> <ul style="list-style-type: none"> <li>– Removed Bold from ‘base address’ in <i>Section 32.1: Unique device ID register (96 bits)</i></li> </ul>

**Table 146. Document revision history (continued)**

Date	Revision	Changes
10-May-2013	3	<ul style="list-style-type: none"> <li>– Added part numbers STM32F06x throughout the document.</li> <li>– Updated document title.</li> </ul> <p><b>Documentation conventions</b></p> <ul style="list-style-type: none"> <li>– Added “(w_r0)” to <i>Section 1.1: List of abbreviations for registers</i>.</li> <li>– Added reference to PM0215 in <i>Related documents</i>.</li> <li>– Updated <i>Glossary</i>.</li> </ul> <p><b>System and memory overview</b></p> <ul style="list-style-type: none"> <li>– Modified Sections <i>BusMatrix</i> and <i>AHB to APB bridge</i>.</li> <li>– Added <i>Note</i>: in <i>Section 2.3: Embedded SRAM</i>.</li> </ul> <p><b>Embedded Flash memory</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Table 4: Access status versus protection level and execution modes</i>.</li> <li>– Modified phrasing of <i>Level 2: no debug</i> paragraph in <i>Section 1.3.1: Read protection</i>.</li> <li>– Added description of information block in <i>Section 1.2.1: Flash memory organization</i>.</li> <li>– Renamed “FKEYR” to “FKEY” and renamed “OPTKEYR” to “OPTKEY”.</li> <li>– Renamed and modified <i>Section 1.5.7: Flash Option byte register (FLASH_OBR)</i>.</li> <li>– Modified information about reset value in <i>Section 1.5.2: Flash key register (FLASH_KEYR)</i>, <i>Section 1.5.3: Flash option key register (FLASH_OPTKEYR)</i>, <i>Section 1.5.7: Flash Option byte register (FLASH_OBR)</i> and <i>Section 1.5.8: Write protection register (FLASH_WRPTR)</i>.</li> <li>– Modified Section <i>Unlocking the Flash memory</i>, item 5. in <i>Main Flash memory programming</i>, procedure in sections <i>Page Erase</i> and <i>Mass Erase</i>.</li> <li>– Added <i>Note</i>: in <i>Page Erase</i> and <i>Mass Erase</i>.</li> <li>– Updated <i>Table 6: Flash interface - register map and reset values</i>.</li> </ul> <p><b>Option bytes</b></p> <ul style="list-style-type: none"> <li>– Reworked <i>Section 2: Option bytes</i>.</li> </ul> <p><b>PWR</b></p> <ul style="list-style-type: none"> <li>– Modified introduction of <i>Section 1.1: Power supplies</i>.</li> <li>– Added bullet “In STM32F06x devices...” in <i>Section 1.1.4: Voltage regulator</i>.</li> <li>– Added section <i>External NPOR signal</i>.</li> <li>– Added <i>Caution</i>: in <i>Section 1.3: Low-power modes</i> and in <i>Section 1.3.5: Standby mode</i>.</li> </ul>

**Table 146. Document revision history (continued)**

Date	Revision	Changes
10-May-2013	3 (cont'd)	<p><b>RCC</b></p> <ul style="list-style-type: none"> <li>– Inverted order of sections <i>Section 1.1.1: Power reset</i> and <i>Section 1.1.2: System reset</i>.</li> <li>– Added <i>Caution:</i> in <i>External crystal/ceramic resonator (HSE crystal)</i> and <i>Caution:</i> in <i>Section 1.2.5: LSE clock</i></li> <li>– Added bullet “The timer clock frequencies are automatically fixed by hardware...” in <i>Section 1.2: Clocks</i>.</li> <li>– Updated <i>Section 1.1.1: Power reset</i>, <i>Section 1.1.2: System reset</i>, <i>Section 1.1.3: RTC domain reset</i>, <i>Calibration of the HSI14</i>, <i>Section 1.2.12: Clock-out capability</i>, <i>Section 1.2.13: Internal/external clock measurement with TIM14</i>, <i>Section 1.4.2: Clock configuration register (RCC_CFGR)</i>, <i>Section 1.4.4: APB peripheral reset register 2 (RCC_APB2RSTR)</i>, <i>Section 1.4.12: Clock configuration register 2 (RCC_CFGR2)</i>, <i>Section 1.4.13: Clock configuration register 3 (RCC_CFGR3)</i>.</li> <li>– Modified <i>Figure 1: Simplified diagram of the reset circuit</i> and <i>Figure 2: Clock tree (STM32F03x and STM32F05x devices)</i>.</li> <li>– Replaced FORCE_OBL with OBL_LAUNCH in <i>Option byte loader reset</i>.</li> <li>– Deleted <i>Section 7.2.13: Clock-independent system clock sources for TIM14</i>.</li> <li>– Added USART3EN, UART4EN and UART5EN in <i>Table 1: RCC register map and reset values</i>.</li> <li>– Modified reset values of RCC_CSR register in <i>Table 1: RCC register map and reset values</i>.</li> <li>– Modified reset value and description of Bit 23 in <i>Section 1.4.10: Control/status register (RCC_CSR)</i>.</li> <li>– Specified Bit 8 as obsolete in <i>Section 1.4.13: Clock configuration register 3 (RCC_CFGR3)</i>.</li> </ul> <p><b>GPIO</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Section 9.1: Introduction</i>.</li> <li>– Removed “Analog” arrows in <i>Figure 17: Basic structure of a five-volt tolerant I/O port bit</i>.</li> <li>– Renamed <i>Section 9.4.11: GPIO port bit reset register (GPIOx_BRR)</i> (<math>x = A..F</math>).</li> </ul> <p><b>SYSCFG</b></p> <ul style="list-style-type: none"> <li>– Modified values of register bits 23, 22 and 20.</li> </ul> <p><b>Interrupts and events</b></p> <ul style="list-style-type: none"> <li>– Added Priority numbers and modified description of EXTI4_15 in <i>Table 1: Vector table</i>.</li> </ul> <p><b>ADC</b></p> <ul style="list-style-type: none"> <li>– Rephrased part of <i>Section 13.4.1: Calibration (ADCAL)</i>.</li> <li>– Replaced AUTDLY with WAIT in sections <i>13.5.5: Example timing diagrams (single/continuous modes hardware/software triggers)</i> and <i>13.12.4: ADC configuration register 1 (ADC_CFGR1)</i>.</li> <li>– Updated <i>Figure 25: ADC block diagram</i>.</li> <li>– Replaced "0x44...0x2FC" with "0x44...0x304" in <i>Table 44: ADC register map and reset values</i>.</li> </ul>

**Table 146. Document revision history (continued)**

Date	Revision	Changes
10-May-2013	3 (cont'd)	<p><b>DAC</b></p> <ul style="list-style-type: none"> <li>– Corrected name of <i>Table 66: External triggers</i>.</li> <li>– Replaced every occurrence of “DAC1” with “DAC”.</li> </ul> <p><b>COMP</b></p> <ul style="list-style-type: none"> <li>– Replaced “COMP1SW1 : Comparator enable” with “COMP1SW1 : Comparator 1 non inverting input DAC switch” in <i>Section 15.6.1: COMP control and status register (COMP_CSR)</i>.</li> </ul> <p><b>TIM2 and TIM3</b></p> <ul style="list-style-type: none"> <li>– Corrected <i>Figure 106: Advanced-control timer block diagram</i>.</li> </ul> <p><b>TIM14</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Figure 17: TIM16 and TIM17 block diagram</i>.</li> </ul> <p><b>TIM15/16/17</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Section 2.1: TIM15/16/17 introduction</i>, <i>Section 2.3: TIM16 and TIM17 main features</i> and <i>Section 2.4.4: Clock sources</i>.</li> <li>– Removed Note in <i>Section 2.6.1: TIM16 and TIM17 control register 1 (TIM16_CR1 and TIM17_CR1)</i>.</li> <li>– Marked as “reserved”: Bits 14 and 6 in <i>Section 2.6.3: TIM16 and TIM17 DMA/interrupt enable register (TIM16_DIER and TIM17_DIER)</i>, bits 6 and 0 in <i>Section 2.6.4: TIM16 and TIM17 status register (TIM16_SR and TIM17_SR)</i>, and bit 6 in <i>Section 2.6.5: TIM16 and TIM17 event generation register (TIM16_EGR and TIM17_EGR)</i>.</li> <li>– Updated <i>Table 7: TIM16 and TIM17 register map and reset values</i>.</li> </ul> <p><b>RTC</b></p> <ul style="list-style-type: none"> <li>– Replaced “power-on reset” with “backup domain reset” throughout <i>Section 25: Real-time clock (RTC)</i>.</li> <li>– Removed “the backup registers are reset when a tamper detection event occurs” in <i>Section 25.2: RTC main features</i>.</li> <li>– Updated <i>RTC backup registers</i>.</li> <li>– Updated <i>RTC_ISR register reset values</i> in <i>Section 25.3.9: Resetting the RTC</i> and <i>Section 25.6.4: RTC initialization and status register (RTC_ISR)</i>.</li> </ul> <p><b>I2C</b></p> <ul style="list-style-type: none"> <li>– Updated case of digital filter enabled and <i>Table 81: Comparison of analog vs. digital filters</i>.</li> <li>– Added note for WUPEN in <i>Section 26.7.1: Control register 1 (I2Cx_CR1)</i>.</li> <li>– Corrected <i>Figure 275: Transfer sequence flowchart for I2C master transmitter for N&gt;255 bytes</i>.</li> <li>– Removed maximum values of parameter “Data hold time” and added row “Data valid time” in <i>Table 90: I2C-SMBUS specification data setup and hold times</i>.</li> <li>– Updated sub-sections <i>I2C timings</i> and <i>Slave clock stretching (NOSTRETCH = 0)</i>.</li> </ul>

**Table 146. Document revision history (continued)**

Date	Revision	Changes
10-May-2013	3 (cont'd)	<p><b>I2C (continued)</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Figure 257: I2C block diagram</i> and <i>Figure 259: Setup and hold timings</i>.</li> <li>– Reclassified section “I2C register map” from 2.8 to 26.7.12.</li> <li>– Added <i>Caution: “If wakeup from Stop is disabled...”</i> in <i>Section 26.4.14: Wakeup from Stop mode on address match</i>.</li> <li>– Added <i>Section 26.5: I2C low-power modes</i>.</li> <li>– Moved <i>Section 24.7: I2C debug mode</i> to 26.4.17 and renamed it <i>Debug mode</i>.</li> <li>– Updated <i>Table 93: Examples of timings settings for fI2CCLK = 8 MHz</i>.</li> </ul> <p><b>USART</b></p> <ul style="list-style-type: none"> <li>– Removed note on bit 19 (RWU) in <i>Section 25.7.8: Interrupt &amp; status register (USART_ISR)</i>.</li> </ul> <p><b>SPI/I2S</b></p> <ul style="list-style-type: none"> <li>– Reorganized SPI initialization and sequence handling sections.</li> <li>– Corrected and updated SPI disabling and DMA handling procedures.</li> <li>– Updated description of NSSP and TI modes.</li> <li>– Corrected DMA CRC management.</li> <li>– Removed CRC interrupt capability.</li> <li>– Changed RXONLY and BIDIMODE bit description.</li> <li>– Modified sections : <i>Simplex communications on page 828</i> and : <i>Data frame format on page 833</i>, <i>28.5.7: Procedure for enabling SPI on page 835</i>, <i>Figure 318: Hardware/software slave select management</i>, <i>Figure 319: Data clock timing diagram</i>, <i>Figure 320: Data alignment when data length is not equal to 8-bit or 16-bit</i>, <i>Figure 327: TI mode transfer</i>.</li> <li>– Modified reset value of SPIx_I2SPR in <i>Section 28.9.9: SPIx_I<sup>2</sup>S prescaler register (SPIx_I2SPR)</i>.</li> <li>– Added <i>Figure 322: Master full duplex communication</i>, <i>Figure 323: Slave full duplex communication</i>, <i>Figure 324: Master full duplex communication with CRC</i>, <i>Figure 325: Master full duplex communication in packed mode</i>.</li> </ul> <p><b>TSC</b></p> <ul style="list-style-type: none"> <li>– Replaced “Power-on reset value” with “Reset value” in <i>Section 17.6.2: TSC interrupt enable register (TSC_IER)</i>.</li> </ul> <p><b>Debug</b></p> <ul style="list-style-type: none"> <li>– Renamed <i>Section 1.3.2: SW-DP pin assignment</i>.</li> <li>– Changed description of Bits 11:0 in <i>Section 1.4.1: MCU device ID code</i>.</li> <li>– Modified notes regarding IDCODE register in <i>Table 5: SW-DP registers</i>.</li> </ul>

**Table 146. Document revision history (continued)**

Date	Revision	Changes
21-May-2013	4	<p><b>ADC</b></p> <ul style="list-style-type: none"> <li>– Added note in <i>Section 13.5.2: Programmable resolution (RES) - fast conversion mode</i>: RES[1:0] can be changed only when ADEN=0.</li> <li>– Added bit CKMODE.</li> <li>– Rewrote <i>Section 13.4.2: ADC on-off control (ADEN, ADDIS, ADRDY)</i> and <i>Section 13.4.3: ADC clock (CKMODE)</i>.</li> <li>– Added <i>Figure 28: ADC clock scheme</i>.</li> <li>– Fixed issues concerning CKMODE &amp; JITOFF bits. Now bits JITOFF_DIV4/DIV2 bits are replaced with bits CKMODE[1:0] at the same position and with same decoding as v1.</li> <li>– Changed note on ADRDY in <i>Section 13.4.2: ADC on-off control (ADEN, ADDIS, ADRDY)</i>.</li> <li>– Extended <i>Section 13.12.8: ADC channel selection register (ADC_CHSELR)</i> (added channel 18).</li> <li>– Removed reference to TSVREFE bit in <i>Section 13.9: Temperature sensor and internal reference voltage</i> and referred to TSEN/VREFEN instead.</li> </ul>
09-Jan-2014	5	<p><b>Cover page</b></p> <ul style="list-style-type: none"> <li>– Changed part numbers in title and introduction on page 1</li> <li>– Renamed low-density to STM32F03x</li> <li>– Renamed medium-density to STM32F05x</li> <li>– Renamed STM32F06x to STM32F08x</li> <li>– Added STM32F07x microcontrollers (STM32F071xB and STM32F072xx). Expanded the Flash memory range to 128 Kbytes.</li> <li>– Added <i>Section 8: Clock recovery system (CRS)</i>, <i>Section 29: Controller area network (bxCAN)</i> and <i>Section 30: Universal serial bus full-speed device interface (USB)</i>.</li> </ul> <p><b>Documentation conventions</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Section 1.2: Glossary</i>.</li> </ul> <p><b>System and memory overview</b></p> <ul style="list-style-type: none"> <li>– Added ‘TIM7, USART3, USART4, CAN and USB’ and ‘7 channels DMA’ in <i>Figure 1: System architecture (all features)</i>.</li> <li>– Added ‘CRS, TIM7, USART3, USART4, CAN and USB’ in <i>Table 1: STM32F0xx peripheral register boundary addresses</i>.</li> <li>– Updated <i>Table 1: STM32F0xx peripheral register boundary addresses</i>.</li> <li>– Added <i>Table 2: STM32F0xx memory boundary addresses</i></li> <li>– Added 16 KB-RAM for STM32F07x devices in <i>Section 2.3: Embedded SRAM</i>.</li> <li>– Added USB in <i>Embedded boot loader</i>.</li> </ul> <p><b>Embedded Flash memory</b></p> <ul style="list-style-type: none"> <li>– Added <i>Table 2: Flash memory organization (STM32F07x devices)</i>.</li> <li>– Added ‘WRP[31:16]’ in <i>Section 1.5.8: Write protection register (FLASH_WRP)</i> and <i>Table 6: Flash interface - register map and reset values</i>.</li> </ul> <p><b>Option bytes</b></p> <ul style="list-style-type: none"> <li>– Added row ‘0x1FFF F80C’ in <i>Table 8: Option byte organization</i>.</li> <li>– Updated <i>Section 2.1.3: Write protection option bytes</i>.</li> <li>– Updated <i>Table 9: Option byte map and ST production values</i>.</li> </ul>

**Table 146. Document revision history (continued)**

Date	Revision	Changes
09-Jan-2014	5 (cont'd)	<p><b>CRC</b></p> <ul style="list-style-type: none"> <li>– Introduced programmable polynomial feature in <i>Section 6.2: CRC main features</i>, <i>Section 6.4.3: Control register (CRC_CR)</i> and <i>Section 6.4.6: CRC register map</i>.</li> <li>– Added <i>Section : Polynomial programmability</i> and <i>Section 6.4.5: CRC polynomial (CRC_POL)</i>.</li> </ul> <p><b>PWR</b></p> <ul style="list-style-type: none"> <li>– Added STM32F07x device features</li> <li>– Updated <i>Figure 1: Power supply overview</i></li> <li>– Added <i>Section 1.1.2: Independent I/O supply rail</i></li> <li>– Added bits 10 to 15 in <i>Section 1.4.2: Power control/status register (PWR_CSR)</i></li> <p><b>RCC</b></p> <ul style="list-style-type: none"> <li>– Renamed “backup domain” to “RTC domain”.</li> <li>– Added HSI48 48 MHz RC oscillator clock and USART2 features for STM32F07x devices:</li> <li>– Modified <i>Section 1.2: Clocks</i> and <i>Section 1.2.2: HSI clock</i>, <i>Section 1.2.4: PLL</i>, <i>Section 1.2.7: System clock (SYSLCK) selection</i>, <i>Section 1.2.12: Clock-out capability</i>, <i>Section 1.3: Low-power modes</i>.</li> <li>– Added <i>Figure 3: Clock tree (STM32F04x and STM32F07x devices)</i> and <i>Section 1.2.3: HSI48 clock</i>.</li> <li>– Updated register descriptions for STM32F07x devices in <i>Section 1.4.2: Clock configuration register (RCC_CFGR)</i>, <i>Section 1.4.3: Clock interrupt register (RCC_CIR)</i>, <i>Section 1.4.5: APB peripheral reset register 1 (RCC_APB1RSTR)</i>, <i>Section 1.4.6: AHB peripheral clock enable register (RCC_AHBENR)</i>, <i>Section 1.4.8: APB peripheral clock enable register 1 (RCC_APB1ENR)</i>, <i>Section 1.4.11: AHB peripheral reset register (RCC_AHBRSTR)</i>, <i>Section 1.4.13: Clock configuration register 3 (RCC_CFGR3)</i>, and <i>Section 1.4.14: Clock control register 2 (RCC_CR2)</i> and updated <i>Table 1: RCC register map and reset values</i>.</li> <p><b>GPIO</b></p> <ul style="list-style-type: none"> <li>– Added port E in ranges of all registers except LCKR.</li> <li>– Modified <i>Section 9.1: Introduction</i>.</li> <li>– Replaced <math>V_{DD}</math> with <math>V_{DDIO}</math> in <i>Figure 17: Basic structure of an I/O port bit</i>, <i>Figure 18: Input floating/pull up/pull down configurations</i>, <i>Figure 19: Output configuration</i>, <i>Figure 20: Alternate function configuration</i> and <i>Figure 21: High impedance-analog configuration</i>.</li> <p><b>SYSCFG</b></p> <ul style="list-style-type: none"> <li>– Added bits [30:24] and 21 for STM32F07x devices in <i>Section 1.1.1: SYSCFG configuration register 1 (SYSCFG_CFGR1)</i>.</li> <li>– Added pin PE[x] for bits [15:0] in <i>Section 1.1.2: SYSCFG external interrupt configuration register 1 (SYSCFG_EXTICR1)</i>, pins PD[x] and PE[x] for bits [15:0] in <i>Section 1.1.3: SYSCFG external interrupt configuration register 2 (SYSCFG_EXTICR2)</i>, pins PD[x], PE[x] and PF[x] for bits [15:0] in <i>Section 1.1.4: SYSCFG external interrupt configuration register 3 (SYSCFG_EXTICR3)</i> and <i>Section 1.1.5: SYSCFG external interrupt configuration register 4 (SYSCFG_EXTICR4)</i>.</li> <li>– Updated <i>Table 1: SYSCFG register map and reset values</i>.</li> </ul> </ul></ul></ul>

**Table 146. Document revision history (continued)**

Date	Revision	Changes
09-Jan-2014	5 (cont'd)	<p><b>DMA</b></p> <ul style="list-style-type: none"> <li>– Change 5 channels to up to 7 channels</li> <li>– Updated <i>Figure 21: DMA block diagram</i></li> <li>– Updated <i>Figure 23: DMA request mapping</i></li> <li>– Added <i>Table 29: Summary of the DMA requests for each channel on STM32F03x, STM32F04x and STM32F05x devices</i> and <i>Table 30: Summary of the DMA requests for each channel on STM32F07x devices</i></li> <li>– Added bits 27 to 20 in <i>Section 11.5.1: DMA interrupt status register (DMA_ISR)</i> and <i>Section 11.5.2: DMA interrupt flag clear register (DMA_IFCR)</i></li> </ul> <p><b>Interrupts and events</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Table 1: Vector table</i></li> </ul> <p><b>ADC</b></p> <ul style="list-style-type: none"> <li>– Updated <i>Section 13.9: Temperature sensor and internal reference voltage</i></li> </ul> <p><b>DAC</b></p> <ul style="list-style-type: none"> <li>– Added second channel DAC_OUT2, Noise-wave generation, Triangular-wave generation and dual mode for STM32F07x devices</li> <li>– Updated <i>Section 14.2: DAC1 main features</i></li> <li>– Added <i>Section 14.4: Dual mode functional description (STM32F07x devices)</i></li> <li>– Added <i>Section 14.4: Noise generation</i></li> <li>– Added <i>Section 14.5: DMA request</i></li> <li>– Added bits 29 to 16 and bits 11 to 6 in <i>Section 14.6.1: DAC control register (DAC_CR)</i></li> <li>– Added bit1 ‘SWTRIG2’ in <i>Section 14.6.2: DAC software trigger register (DAC_SWTRIGR)</i></li> <li>– Added <i>Section 14.8.6: DAC channel2 12-bit right-aligned data holding register (DAC_DHR12R2)</i>, <i>Section 14.8.7: DAC channel2 12-bit left-aligned data holding register (DAC_DHR12L2)</i>, <i>Section 14.8.8: DAC channel2 8-bit right-aligned data holding register (DAC_DHR8R2)</i>, <i>Section 14.8.9: Dual DAC 12-bit right-aligned data holding register (DAC_DHR12RD)</i>, <i>Section 14.8.10: Dual DAC 12-bit left-aligned data holding register (DAC_DHR12LD)</i>, <i>Section 14.8.11: Dual DAC 8-bit right-aligned data holding register (DAC_DHR8RD)</i>, <i>Section 14.8.13: DAC channel2 data output register (DAC_DOR2)</i></li> <li>– Added bit 29 in <i>Section 14.6.7: DAC status register (DAC_SR)</i></li> </ul> <p><b>Basic timer (TIM6/TIM7)</b></p> <ul style="list-style-type: none"> <li>– Added TIM7 for STM32F07x devices in <i>Section 1: Basic timer (TIM6/TIM7)</i></li> </ul> <p><b>RTC</b></p> <ul style="list-style-type: none"> <li>– Renamed “backup domain to “RTC domain”</li> <li>– Added RTC_TAMP3 and wakeup interrupt for STM32F07x devices</li> <li>– Added <i>Table 67: STM32F0xx RTC implementation</i></li> <li>– Added <i>Section 25.3.6: Periodic auto-wakeup, Section : Programming the wakeup timer</i></li> <li>– Corrected bit SHPF read and clear parameters in <i>Section 25.6.4: RTC initialization and status register (RTC_ISR)</i></li> </ul> <p><b>I2C</b></p> <ul style="list-style-type: none"> <li>– Added <i>Table 74: STM32F0xx I2C implementation</i></li> <li>– Updated <i>Table 90: I2C-SMBUS specification data setup and hold times</i></li> </ul>

**Table 146. Document revision history (continued)**

Date	Revision	Changes
09-Jan-2014	5 (cont'd)	<ul style="list-style-type: none"> <li>– Replaced 50ns into tAF(min) and 260ns into tAF(max) in section <i>I2C timings</i></li> <li><b>USART</b> <ul style="list-style-type: none"> <li>– Added number of auto baud rate detection mode in <i>Table 90: STM32F0xx USART implementation</i>.</li> <li>– Removed mentions of mantissa and fraction in <i>Figure 288: USART block diagram</i> and <i>Section 27.7.4: Baud rate register (USARTx_BRR)</i>.</li> <li>– Sections RTS flow control and CTS flow control renamed <i>RS232 RTS flow control</i> and <i>RS232 CTS flow control</i>. Added case of idle character in <i>Section 27.5.1: USART character description</i>. Added recommendation of TE reset during data transmission in <i>Section : Character transmission</i>. Updated <i>Section : Break characters</i>.</li> <li>– Removed note related to USART disabling procedure in <i>Section : Single byte communication</i>.</li> <li>– Added case of oversampling by 8 in <i>Section 27.5.6: Auto baud rate detection</i>.</li> <li>– Renamed <i>Section 27.5.16 RS232 Hardware flow control</i> and <i>RS485 Driver Enable</i>.</li> <li>– Added check that no transfer is ongoing before entering stop mode in <i>Section 27.5.17: Wakeup from Stop mode</i>; added case of RXNE set in <i>Section : Using Mute mode with Stop mode</i>.</li> <li>– Updated <i>Table 111: USART register map and reset values</i>.</li> <li>– Modified <i>Table 90: STM32F0xx USART implementation</i>.</li> <li>– Added note related to smartcard mode in <i>Section 27.7.1: Control register 1 (USARTx_CR1)</i>.</li> <li>– Added sequence to deliver SCLK clock to smartcards, in CLKEN description of <i>Section 27.7.2: Control register 2 (USARTx_CR2)</i>.</li> <li>– Added note related to PSC configuration in <i>Section 27.7.5: Guard time and prescaler register (USARTx_GTPR)</i>.</li> <li>– Added support of 7-bit data word length: added 6 LSB bits in <i>Section 27.5.1: USART character description</i>, <i>Section 27.5.9: Parity control</i>, <i>Section : Even parity</i> and <i>Section : Odd parity</i>, <i>Section 27.5.5: Tolerance of the USART receiver to clock deviation</i>, renamed M bit M0 in <i>Section 27.7.1: Control register 1 (USARTx_CR1)</i>.</li> <li>– Added 2 auto baud rate detection modes: modes 2 and 3 in <i>Section 27.5.5: Tolerance of the USART receiver to clock deviation</i> and <i>Section 27.5.6: Auto baud rate detection</i>.</li> <li>– Replaced in Bit 2 MMRQ <i>Section 25.7.7: Request register (USART_RQR)</i> “resets the RWU flag” by “sets the RWU flag”</li> <li>– Added ‘In Smartcard, LIN and IrDA modes, only Oversampling by 16 is supported’ in <i>Section 25.5.4: Baud rate generation</i></li> <li>– Corrected and updated stop bits in <i>Figure 198: Word length programming</i></li> <li><b>SPI</b> <ul style="list-style-type: none"> <li>– Modified <i>Table 98: STM32F0xx SPI implementation</i></li> </ul> </li> <li><b>TSC</b> <ul style="list-style-type: none"> <li>– Added Groups 7 and 8 and channels 19 to 24 in <i>Section 17: Touch sensing controller (TSC)</i>. Added <i>Table 106: Capacitive sensing GPIOs</i></li> </ul> </li> <li><b>DEBUG</b> <ul style="list-style-type: none"> <li>– Added STM32F07x ID in bits DEV_IDx in <i>Section 1.4.1: MCU device ID code</i></li> <li>– Added bit 5 ‘DBG_TIM7_STOP’ and bit 25 ‘DBG_CAN_STOP’ to <i>Section 1.9.4: Debug MCU APB1 freeze register (DBGMCU_APB1_FZ)</i></li> </ul> </li> </ul> </li></ul>

**Table 146. Document revision history (continued)**

Date	Revision	Changes
06-May-2014	6	<p><b>Cover page</b>  – Updated for STingray adding STM32F042x4/x6 and STM32F048x6 devices.</p> <p><b>MMAP</b>  – updated <i>Table 2: STM32F0xx memory boundary addresses</i>  – updated <i>Section 2.5: Boot configuration</i></p> <p><b>Embedded Flash memory</b>  – Updated <i>Section 1.2.1: Flash memory organization, Table 4: Access status versus protection level and execution modes</i> and <i>Section 1.5: Flash register description</i>  – Option byte : updated <i>Section 2.1: Option byte description</i></p> <p><b>RCC</b>  – Updated with STM32F04x devices : <i>Section 1.2: Clocks</i> and <i>Figure 3: Clock tree (STM32F04x and STM32F07x devices)</i>.</p> <p><b>CRS</b>  – Updated adding STM32F04x devices.</p> <p><b>GPIO</b>  – Updated <i>Figure 17: Basic structure of an I/O port bit</i></p> <p><b>SYSCFG</b>  – Updated SYSCFG Registers</p> <p><b>DMA updated</b></p> <p><b>IRTIM</b>  – Updated <i>Figure 252: IR internal hardware connections with TIM16 and TIM17</i>.</p> <p><b>IWDG</b>  – updated <i>Section 24.3: IWDG functional description</i>.step6 moved to step1.  – updated <i>Section 24.4: IWDG registers</i>.Moved Note From IWDG_WINR to IWDG_SR register.  – Added STM32F04x</p> <p><b>RTC</b>  – Updated <i>Figure 256: RTC block diagram title</i>.  – Updated <i>Table 67: STM32F0xx RTC implementation</i>.  – Updated <i>25.3: RTC functional description</i>.</p> <p><b>I2C</b>  – Added STM32F04x in <i>Table 74: STM32F0xx I2C implementation</i></p> <p><b>USART</b>  – Changed Stingray adding STM32F04x in <i>Table 90: STM32F0xx USART implementation</i>, <i>27.5: USART functional description</i> and <i>27.7: USART registers</i>.  – Updated <i>Table 109: Frame formats</i> in USART IP  – Changed Stingray adding STM32F04x in <i>Table 98.: STM32F0xx SPI implementation</i>.</p>

**Table 146. Document revision history (continued)**

Date	Revision	Changes
06-May-2014	6 (cont'd)	<ul style="list-style-type: none"> <li>– Changed Stingray adding STM32F04x in <a href="#">Table 98.: STM32F0xx SPI implementation</a>.</li> <li>– Updated TSC IP adding STM32F04x.</li> </ul> <p><b>USB</b></p> <ul style="list-style-type: none"> <li>– Changed USB IP Stingray by adding STM32F04x and STM32F078.</li> <li>– Change Stingray and Nemo in <i>USB main features</i> and <i>Description of USB blocks</i>.</li> <li>– Updated <i>Section 32: Device electronic signature</i></li> <li>– Updated <i>Section 1.3.10: One-pulse mode</i> and <i>Section 1.3.15: Timer synchronization</i>.</li> </ul>
29-Oct-2014	7	<p>Extended the applicability to STM32F091xB/xC and STM32F098xC devices.</p> <p>Updated the following sections:</p> <ul style="list-style-type: none"> <li>– cover page</li> </ul> <p><b>System and memory overview</b></p> <ul style="list-style-type: none"> <li>– <i>Figure 1: System architecture</i></li> <li>– <i>Table 1: STM32F0xx peripheral register boundary addresses</i></li> <li>– <i>Section 2.3: Embedded SRAM</i></li> <li>– <i>Section 2.5: Boot configuration</i></li> </ul> <p><b>Embedded Flash memory</b></p> <ul style="list-style-type: none"> <li>– <i>Section 3.1: Flash main features</i></li> <li>– <i>Section 3.2: Flash memory functional description</i></li> <li>– <i>Section 4.1.3: Write protection option byte</i></li> </ul> <p><b>Cyclic redundancy check calculation unit (CRC)</b></p> <ul style="list-style-type: none"> <li>– Introduction</li> </ul> <p><b>Power control (PWR)</b></p> <ul style="list-style-type: none"> <li>– Figure footnote <a href="#">1</a> related to the figure Power supply overview</li> <li>– <i>Section 6.1.2: Independent I/O supply rail</i></li> <li>– <i>Section 6.4.2: Power control/status register (PWR_CSR)</i></li> </ul> <p><b>Reset and clock control (RCC)</b></p> <ul style="list-style-type: none"> <li>– <i>Section 7.2: Clocks</i></li> <li>– <i>Figure 12: Clock tree (STM32F04x, STM32F07x and STM32F09x devices)</i></li> <li>– <i>Section 7.2.3: HSI48 clock</i></li> <li>– <i>Section 7.2.7: System clock (SYSSCLK) selection</i></li> <li>– <i>Section 7.2.12: Clock-out capability</i></li> <li>– <i>Section 7.3: Low-power modes</i></li> <li>– <i>Section 7.4.2: Clock configuration register (RCC_CFGR)</i></li> <li>– <i>Section 7.4.4: APB peripheral reset register 2 (RCC_APB2RSTR)</i></li> <li>– <i>Section 7.4.5: APB peripheral reset register 1 (RCC_APB1RSTR)</i></li> <li>– <i>Section 7.4.6: AHB peripheral clock enable register (RCC_AHBENR)</i></li> <li>– <i>Section 7.4.7: APB peripheral clock enable register 2 (RCC_APB2ENR)</i></li> <li>– <i>Section 7.4.8: APB peripheral clock enable register 1 (RCC_APB1ENR)</i></li> <li>– <i>Section 7.4.13: Clock configuration register 3 (RCC_CFGR3)</i></li> </ul> <p><b>Clock recovery system (CRS)</b></p> <ul style="list-style-type: none"> <li>– Introduction</li> </ul>

**Table 146. Document revision history (continued)**

Date	Revision	Changes
29-Oct-2014	7 (cont'd)	<p><b>General-purpose I/Os (GPIO)</b></p> <ul style="list-style-type: none"> <li>– Introduction</li> </ul> <p><b>System configuration controller (SYSCFG)</b></p> <ul style="list-style-type: none"> <li>– <i>Section 10.1.1: SYSCFG configuration register 1 (SYSCFG_CFGR1)</i></li> <li>– Added <i>Section 10.1.7: SYSCFG interrupt line 0 status register (SYSCFG_ITLINE0)</i> to <i>Section 10.1.37: SYSCFG interrupt line 30 status register (SYSCFG_ITLINE30)</i></li> <li>– Added <i>Table 28: SYSCFG register map and reset values for STM32F09x devices</i></li> </ul> <p><b>Direct memory access controller (DMA)</b></p> <ul style="list-style-type: none"> <li>– <i>Table 32: Summary of the DMA requests for each channel on STM32F07x devices</i></li> <li>– <i>Table 36: DMA register map and reset values (registers available on STM32F07x and STM32F09x devices only)</i></li> </ul> <p><b>Interrupts and events</b></p> <ul style="list-style-type: none"> <li>– <i>Table 38: Vector table</i></li> <li>– <i>Section 12.2.5: External and internal interrupt/event line mapping</i></li> <li>– <i>Section 12.3.1: Interrupt mask register (EXTI_IMR)</i></li> </ul> <p><b>Digital-to-analog converter (DAC)</b></p> <ul style="list-style-type: none"> <li>– Introduction</li> <li>– <i>Section 14.2: DAC main features</i></li> <li>– <i>Section 14.5.2: DAC channel conversion</i></li> <li>– <i>Section 14.6: Dual-mode functional description (STM32F07x and STM32F09x devices)</i></li> <li>– <i>Section 14.7: Noise generation(STM32F07x and STM32F09x devices)</i></li> <li>– <i>Section 14.8: Triangle-wave generation (STM32F07x and STM32F09x devices)</i></li> <li>– <i>Section 14.10.14: DAC status register (DAC_SR)</i></li> </ul> <p><b>Comparator (COMP)</b></p> <ul style="list-style-type: none"> <li>– Introduction</li> <li>– <i>Section 15.1: Introduction</i></li> </ul> <p><b>General purpose timers (TIM15/16)</b></p> <ul style="list-style-type: none"> <li>– <i>Section 19.4.14: TIM15 external trigger synchronization</i></li> <li>– <i>Section 19.4.15: Timer synchronization (TIM15)</i></li> </ul> <p><b>Basic timer (TIM6/TIM7)</b></p> <ul style="list-style-type: none"> <li>– Introduction</li> </ul> <p><b>Infrared interface (IRTIM)</b></p> <ul style="list-style-type: none"> <li>– <i>Figure 202: IR internal hardware connections with TIM16 and TIM17</i></li> </ul> <p><b>Real time clock (RTC)</b></p> <ul style="list-style-type: none"> <li>– <i>Table 71: STM32F0xx RTC implementation</i></li> <li>– <i>Figure 205: RTC block diagram for STM32F07x and STM32F09x devices</i></li> <li>– <i>Table 76: Interrupt control bits</i></li> </ul> <p><b>Inter-Integrated circuit (I2C) interface</b></p> <p><i>Table 79: STM32F0xx I2C implementation</i></p> <p><b>Universal synchronous asynchronous receiver transmitter (USART)</b></p> <ul style="list-style-type: none"> <li>– <i>Table 95: STM32F0xx USART implementation</i></li> </ul>

**Table 146. Document revision history (continued)**

Date	Revision	Changes
29-Oct-2014	7 (cont'd)	<p><b>Serial peripheral interface / inter-IC sound (SPI/I2S)</b>  – <i>Table 104: STM32F0xx SPI implementation</i></p> <p><b>Touch sensing controller (TSC)</b>  – Introduction</p> <p><b>Controller area network (bxCAN)</b>  – Introduction</p> <p><b>HDMI-CEC controller (HDMI-CEC)</b>  – Introduction</p> <p><b>Debug support (DBG)</b>  <i>Section 32.4.1: MCU device ID code</i></p>
29-Jul-2015	8	<p><b>System and memory overview</b>  – Corrected the original memory space value for STM32F09x devices in <i>Section 2.5: Boot configuration</i>.</p> <p><b>CRS</b>  – Added the note in SYNCSRC[1:0] description of CRS configuration register (CRS_CFGR) in <i>Section 8.6.2: CRS configuration register (CRS_CFGR)</i>.</p> <p><b>Interrupts and events</b>  – Corrected the definition for EXTI line 31 in <i>Section 12.2.5: External and internal interrupt/event line mapping</i></p> <p><b>Appendix A Code examples</b>  – Global update of <i>Section Appendix A: Code examples</i>: now displayed with colors.</p>
20-Jan-2017	9	<p>The order of functions (peripherals) changed.  Updated the following sections:</p> <p><b>System and memory overview</b>  – <i>Table 13: STM32L4Rxxx and STM32L4Sxxx peripheral register boundary addresses OrcaZero</i>  – <i>Section 2.3: Embedded SRAM</i>  – <i>Section 2.5: Boot configuration</i></p> <p><b>Embedded Flash memory</b>  – <i>Section 3.5.4: Flash status register (FLASH_SR)</i>  – <i>Section 3.5.7: Flash Option byte register (FLASH_OBR)</i>  – <i>Section 4: Option bytes</i></p> <p><b>Power control (PWR)</b>  – <i>Section 5.4.2: Power control/status register (PWR_CSR)</i></p> <p><b>Reset and clock control (RCC)</b>  – <i>Section 6.2.10: RTC clock</i>  – <i>Section 6.4.9: RTC domain control register (RCC_BDCR)</i>  – <i>Table 19: RCC register map and reset values</i>  – <i>Section 6.4.14: Clock control register 2 (RCC_CR2)</i></p> <p><b>General-purpose I/Os (GPIO)</b>  – <i>Section 8.3.2: I/O pin alternate function multiplexer and mapping</i>  – <i>Section 8.4.12: GPIO register map</i>  – AF Ry field names changed to AF SELy in <i>Section 8.4.9: GPIO alternate function low register (GPIOx_AFRL)</i> (x = A to F), <i>Section 8.4.10: GPIO alternate function high register (GPIOx_AFRH)</i> (x = A to F)</p>

**Table 146. Document revision history (continued)**

Date	Revision	Changes
20-Jan-2017	9 (cont'd)	<ul style="list-style-type: none"> <li>– <a href="#">Figure 17: Input floating / pull up / pull down configurations</a></li> <li>– <a href="#">Figure 18: Output configuration</a></li> <li>– <a href="#">Figure 19: Alternate function configuration</a></li> <li>– <a href="#">Figure 20: High impedance-analog configuration</a></li> <li><b>System configuration controller (SYSCFG)</b></li> <li>– <a href="#">Section 9.1.1: SYSCFG configuration register 1 (SYSCFG_CFGR1)</a></li> <li>– <a href="#">Section 9.1.38: SYSCFG register maps</a></li> <li><b>Direct memory access controller (DMA)</b></li> <li>– <a href="#">Table 30: Summary of the DMA requests for each channel on STM32F07x devices</a></li> <li><b>Interrupts and events</b></li> <li>– <a href="#">Section 11.3.3: Rising trigger selection register (EXTI_RTSR)</a></li> <li>– <a href="#">Section 11.3.4: Falling trigger selection register (EXTI_FTSR)</a></li> <li>– <a href="#">Section 11.3.5: Software interrupt event register (EXTI_SWIER)</a></li> <li>– <a href="#">Section 11.3.6: Pending register (EXTI_PR)</a></li> <li>– <a href="#">Section 11.3.7: EXTI register map</a></li> <li><b>Cyclic redundancy check calculation unit (CRC)</b></li> <li>– Feature list</li> <li>– <a href="#">Figure 25: CRC calculation unit block diagram</a></li> <li>– <a href="#">Table 40: CRC register map and reset values</a></li> <li>– <a href="#">Section 12.4.2: CRC internal signals</a></li> <li>– <a href="#">Section 12.5.3: CRC control register (CRC_CR)</a></li> <li>– <a href="#">Section 12.5.5: CRC polynomial (CRC_POL)</a></li> <li><b>Analog-to-digital converter (ADC)</b></li> <li>– <a href="#">Figure 26: ADC block diagram</a></li> <li>– <a href="#">Section 15.4.2: ADC voltage regulator (ADVREGEN)</a></li> <li>– <a href="#">Section 13.3.3: ADC on-off control (ADEN, ADDIS, ADRDY)</a></li> <li>– <a href="#">Section : Reading the temperature</a></li> <li>– <a href="#">Section 13.11.1: ADC interrupt and status register (ADC_ISR)</a></li> <li>– <a href="#">Section 13.11.3: ADC control register (ADC_CR)</a></li> <li>– <a href="#">Section 13.11.5: ADC configuration register 2 (ADC_CFGR2)</a></li> <li><b>Digital-to-analog converter (DAC)</b></li> <li>– Sections DAC output buffer enable and DAC channel enable moved outside Single and Dual mode functional description sections</li> <li>– Separation of Single mode and Dual mode functional descriptions</li> <li><b>Comparator (COMP)</b></li> <li>– <a href="#">Section 15.5.1: COMP control and status register (COMP_CSR)</a></li> <li><b>All timers</b></li> <li>– Reset value of all TIMx_ARR registers corrected to 0xFFFF</li> <li><b>Advanced control timers (TIM1)</b></li> <li>– <a href="#">Section Figure 63.: Advanced-control timer block diagram</a></li> <li>– <a href="#">Section 17.3.2: Counter modes</a></li> <li>– <a href="#">Section 17.3.12: Using the break function</a></li> <li>– <a href="#">Section 17.4.5: TIM1 status register (TIM1_SR)</a></li> <li><b>General-purpose timers (TIM2 and TIM3)</b></li> <li>– <a href="#">Section 18.3.5: Input capture mode</a></li> </ul>

**Table 146. Document revision history (continued)**

Date	Revision	Changes
20-Jan-2017	9 (cont'd)	<p><b>General-purpose timers (TIM15/16/17)</b></p> <ul style="list-style-type: none"> <li>– <a href="#">Section Figure 172.: TIM15 block diagram</a></li> <li>– <a href="#">Section Figure 173.: TIM16 and TIM17 block diagram</a></li> <li>– <a href="#">Section 20.4.12: Using the break function</a></li> <li>– <a href="#">Section 20.4.13: One-pulse mode</a></li> <li>– <a href="#">Section 20.5.3: TIM15 slave mode control register (TIM15_SMCR)</a></li> <li>– <a href="#">Section 20.5.5: TIM15 status register (TIM15_SR)</a></li> <li>– <a href="#">Section 20.6.4: TIMx status register (TIMx_SR)(x = 16 to 17)</a></li> </ul> <p><b>Infrared interface (IRTIM)</b></p> <ul style="list-style-type: none"> <li>– Introduction</li> </ul> <p><b>Independent watchdog (IWDG)</b></p> <ul style="list-style-type: none"> <li>– Added <a href="#">Section 23.3.4: Behavior in Stop and Standby modes</a></li> <li>– <b>Real time clock (RTC)</b></li> <li>– <a href="#">Figure 211: RTC block diagram in STM32F03x, STM32F04x and STM32F05x devices</a></li> <li>– <a href="#">Figure 212: RTC block diagram for STM32F07x and STM32F09x devices</a></li> <li>– <a href="#">Section 25.4.9: Resetting the RTC</a></li> <li>– <a href="#">Section 25.7.15: RTC tamper and alternate function configuration register (RTC_TAFCR)</a></li> <li>– <a href="#">Section 25.7.18: RTC register map</a></li> <li>– <a href="#">Section 25.4.15: Calibration clock output.</a></li> <li>– Added caution at the end of <a href="#">Section 25.7.3: RTC control register (RTC_CR)</a>.</li> </ul> <p><b>Inter-Integrated circuit (I2C) interface</b></p> <ul style="list-style-type: none"> <li>– <a href="#">Table 86: STM32F0xx I2C implementation</a></li> <li>– <a href="#">Figure 220: Setup and hold timings</a></li> <li>– <a href="#">Section I2C timings</a></li> <li>– <a href="#">Section 26.4.10: I2C master mode</a></li> <li>– <a href="#">Section 26.7.3: Own address 1 register (I2C_OAR1)</a></li> <li>– <a href="#">Section 26.7.4: Own address 2 register (I2C_OAR2)</a></li> <li>– <a href="#">Section 26.7.5: Timing register (I2C_TIMINGR)</a></li> <li>– <a href="#">Figure 224: Slave initialization flow</a></li> </ul> <p><b>Universal synchronous asynchronous receiver transmitter (USART)</b></p> <ul style="list-style-type: none"> <li>– all nCTS / nRTS / SCLK replaced with CTS / RTS / CK</li> <li>– <a href="#">Section 27.5.5: Tolerance of the USART receiver to clock deviation</a></li> <li>– Added description to <a href="#">Section 27.5.17: Wake-up from Stop mode using USART</a></li> <li>– <a href="#">Section 27.8.3: USART control register 3 (USART_CR3)</a></li> <li>– <a href="#">Section 27.8.9: USART interrupt flag clear register (USART_ICR)</a></li> <li>– Adding section <a href="#">Determining the maximum USART baud rate allowing to wake-up correctly from Stop mode when the USART clock source is the HSI clock</a></li> </ul>

**Table 146. Document revision history (continued)**

Date	Revision	Changes
20-Jan-2017	9 (cont'd)	<p><b>Serial peripheral interface / inter-IC sound (SPI/I2S)</b></p> <ul style="list-style-type: none"> <li>– Footnotes of <a href="#">Figure 274: Full-duplex single master/ single slave application</a>, <a href="#">Figure 275: Half-duplex single master/ single slave application</a> and <a href="#">Figure 276: Simplex single master/single slave application (master in transmit-only/ slave in receive-only mode)</a></li> <li>– <a href="#">Figure 290: Full-duplex communication</a></li> <li>– Added <a href="#">Section 28.5.4: Multi-master communication</a></li> <li>– Added <a href="#">Section 28.7.2: I2S full duplex</a></li> </ul> <p><b>Touch sensing controller (TSC)</b></p> <ul style="list-style-type: none"> <li>– <a href="#">Section 16.3.4: Charge transfer acquisition sequence</a> - note added</li> <li>– <a href="#">Section 16.6.1: TSC control register (TSC_CR)</a> - note added</li> <li>– <a href="#">Section 16.6.4: TSC interrupt status register (TSC_ISR)</a></li> <li>– Removed section Capacitive sensing GPIOs</li> </ul> <p><b>HDMI-CEC</b></p> <ul style="list-style-type: none"> <li>– <a href="#">Section Figure 329.: HDMI-CEC block diagram</a></li> </ul> <p><b>DBG</b></p> <ul style="list-style-type: none"> <li>– <a href="#">Section 32.4.1: MCU device ID code</a></li> </ul> <p><b>Appendix A - Code examples</b></p> <ul style="list-style-type: none"> <li>– <a href="#">Section A.2.2: Main Flash programming sequence</a></li> <li>– <a href="#">Section A.2.3: Page erase sequence</a></li> <li>– <a href="#">Section A.2.4: Mass erase sequence</a></li> <li>– <a href="#">Section A.2.6: Option byte programming sequence</a></li> <li>– <a href="#">Section A.4.2: Alternate function selection sequence code example</a></li> <li>– <a href="#">Section A.2.7: Option byte erasing sequence</a></li> <li>– <a href="#">Section A.7.1: ADC calibration code example</a></li> <li>– <a href="#">Section A.7.2: ADC enable sequence code example</a></li> <li>– <a href="#">Section A.7.3: ADC disable sequence code example</a></li> <li>– <a href="#">Section A.16.7: RTC tamper and time stamp code example</a></li> </ul>

**Table 146. Document revision history (continued)**

Date	Revision	Changes
18-May-2022	10	<p>Updated <a href="#">Introduction</a>, <a href="#">Embedded boot loader</a>, <a href="#">Prefetch controller</a>, <a href="#">Section 3.3.1: Read protection</a>, <a href="#">Section 5.2.1: Power on reset (POR) / power down reset (PDR)</a>, <a href="#">Section 5.2.2: Programmable voltage detector (PVD)</a>, <a href="#">Section 8.1: Introduction</a>, <a href="#">Section 8.3.2: I/O pin alternate function multiplexer and mapping</a>, <a href="#">Calculating the actual <math>V_{DDA}</math> voltage using the internal reference voltage</a>, <a href="#">Section 19.4.5: TIM14 capture/compare mode register 1 [alternate] (TIM14_CCMR1)</a>, <a href="#">Section 19.4.6: TIM14 capture/compare mode register 1 [alternate] (TIM14_CCMR1)</a>, <a href="#">Section 20.4.1: Time-base unit</a>, <a href="#">Section 20.4.4: Clock sources</a>, <a href="#">Section 20.4.10: PWM mode</a>, <a href="#">Section 20.4.13: One-pulse mode</a>, <a href="#">Section 20.5.7: TIM15 capture/compare mode register 1 [alternate] (TIM15_CCMR1)</a>, <a href="#">Section 20.6.6: TIMx capture/compare mode register 1 [alternate] (TIMx_CCMR1)(x = 16 to 17)</a>, <a href="#">Section 20.6.7: TIMx capture/compare mode register 1 [alternate] (TIMx_CCMR1)(x = 16 to 17)</a>, <a href="#">Section 25.4.16: Alarm output</a>, <a href="#">Section 26.4.2: I2C2 block diagram</a>, and <a href="#">Hardware transfer management</a>.</p> <p>Added <a href="#">Section 20.5.8: TIM15 capture/compare mode register 1 [alternate] (TIM15_CCMR1)</a>, note in <a href="#">Section 32.9.3: Debug MCU configuration register (DBGMCU_CR)</a>, and <a href="#">Important security notice</a>.</p> <p>Updated <a href="#">Figure 1: System architecture</a>, <a href="#">Figure 2: Memory map</a>, <a href="#">Figure 6: Power supply overview</a>, <a href="#">Figure 16: Basic structure of an I/O port bit</a>, <a href="#">Figure 17: Input floating / pull up / pull down configurations</a>, <a href="#">Figure 18: Output configuration</a>, <a href="#">Figure 19: Alternate function configuration</a>, <a href="#">Figure 63: Advanced-control timer block diagram</a>, <a href="#">Figure 187: Capture/compare channel 1 main circuit</a>, <a href="#">Figure 227: Transfer bus diagrams for I2C slave transmitter (mandatory events only)</a>, <a href="#">Figure 230: Transfer bus diagrams for I2C slave receiver (mandatory events only)</a>, <a href="#">Figure 237: Transfer bus diagrams for I2C master transmitter (mandatory events only)</a>, <a href="#">Figure 240: Transfer bus diagrams for I2C master receiver (mandatory events only)</a>, and footnote 2 of <a href="#">Figure 276: Simplex single master/single slave application (master in transmit-only/slave in receive-only mode)</a>.</p> <p>Updated <a href="#">Table 7: Access status versus protection level and execution modes</a>, <a href="#">Table 36: Vector table</a>, <a href="#">Table 85: RTC register map and reset values</a>, and <a href="#">Table 138: DEV_ID and REV_ID field values</a>.</p> <p>Minor text edits across the whole document.</p>

# Index

## A

ADC_CCR .....	274
ADC_CFGR1 .....	267
ADC_CFGR2 .....	271
ADC_CHSELR .....	272
ADC_CR .....	265
ADC_DR .....	273
ADC_IER .....	263
ADC_ISR .....	262
ADC_SMPR .....	271
ADC_TR .....	272

## C

CAN_BTR .....	854
CAN_ESR .....	853
CAN_FA1R .....	862
CAN_FFA1R .....	861
CAN_FiRx .....	863
CAN_FM1R .....	860
CAN_FMR .....	860
CAN_FS1R .....	861
CAN_IER .....	851
CAN_MCR .....	844
CAN_MSR .....	846
CAN_RDHxR .....	859
CAN_RDLxR .....	859
CAN_RDTxR .....	858
CAN_RF0R .....	850
CAN_RF1R .....	850
CAN_RIxR .....	858
CAN_TDHzR .....	857
CAN_TDLxR .....	856
CAN_TDTxR .....	856
CAN_TIxR .....	855
CAN_TSR .....	847
CEC_CFGR .....	913
CEC_CR .....	912
CEC_IER .....	917
CEC_ISR .....	915
CEC_RXDR .....	915
CEC_TXDR .....	915
COMP_CSR .....	304
CRC_CR .....	230
CRC_DR .....	229
CRC_IDR .....	229
CRC_INIT .....	231
CRC_POL .....	231

CRS_CFGR .....	144
CRS_CR .....	143
CRS_ICR .....	147
CRS_ISR .....	145

## D

DAC_CR .....	289
DAC_DHR12L1 .....	294
DAC_DHR12L2 .....	295
DAC_DHR12LD .....	296
DAC_DHR12R1 .....	293
DAC_DHR12R2 .....	294
DAC_DHR12RD .....	296
DAC_DHR8R1 .....	294
DAC_DHR8R2 .....	295
DAC_DHR8RD .....	296
DAC_DOR1 .....	297
DAC_DOR2 .....	297
DAC_SR .....	297
DAC_SWTRIGR .....	293
DBGMCU_APB1_FZ .....	932
DBGMCU_APB2_FZ .....	934
DBGMCU_CR .....	931
DBGMCU_IDCODE .....	923
DMA_CCRx .....	206
DMA_CMARx .....	210
DMA_CNDTRx .....	209
DMA_CPARx .....	210
DMA_IFCR .....	205
DMA_ISR .....	203
DMA1_CSELR .....	212

## E

EXTI_EMR .....	222
EXTI_FTSR .....	223
EXTI_IMR .....	221
EXTI_PR .....	224
EXTI_RTSR .....	222
EXTI_SWIER .....	223

## F

FLASH_ACR .....	69
FLASH_CR .....	71
FLASH_KEYR .....	70
FLASH_OPTKEYR .....	70
FLASH_SR .....	71

**G**

GPIOx_AFRH .....	163
GPIOx_AFRL .....	162
GPIOx_BRR .....	163
GPIOx_BSRR .....	161
GPIOx_IDR .....	160
GPIOx_LCKR .....	161
GPIOx_MODER .....	158
GPIOx_ODR .....	160
GPIOx_OSPEEDR .....	159
GPIOx_OTYPER .....	158
GPIOx_PUPDR .....	159

**I**

I2C_CR1 .....	684
I2C_CR2 .....	687
I2C_ICR .....	695
I2C_ISR .....	693
I2C_OAR1 .....	689
I2C_OAR2 .....	690
I2C_PECR .....	696
I2C_RXDR .....	697
I2C_TIMOUSR .....	692
I2C_TIMINGR .....	691
I2C_TXDR .....	697
IWDG_KR .....	577
IWDG_PR .....	578
IWDG_RLR .....	579
IWDG_SR .....	580
IWDG_WINR .....	581

**P**

PWR_CR .....	92
PWR_CSR .....	93

**R**

RCC_AHBENR .....	121
RCC_AHBRSTR .....	130
RCC_APB1ENR .....	124
RCC_APB1RSTR .....	119
RCC_APB2ENR .....	123
RCC_APB2RSTR .....	117
RCC_BDCR .....	127
RCC_CFGR .....	111
RCC_CFGR2 .....	131
RCC_CFGR3 .....	132
RCC_CIR .....	114
RCC_CR .....	110
RCC_CR2 .....	133

RCC_CSR .....	128
RTC_ALRMAR .....	617
RTC_ALRMASSR .....	627
RTC_BKPxR .....	628
RTC_CALR .....	623
RTC_CR .....	610
RTC_DR .....	608
RTC_ISR .....	613
RTC_PRER .....	615
RTC_SHIFTR .....	619
RTC_SSR .....	618
RTC_TAFCR .....	624
RTC_TR .....	607
RTC_TSDR .....	621
RTC_TSSSR .....	622
RTC_TSTR .....	620
RTC_WPR .....	618
RTC_WUTR .....	616

**S**

SPIx_CR1 .....	814
SPIx_CR2 .....	816
SPIx_CRCPR .....	820
SPIx_DR .....	820
SPIx_I2SCFGR .....	821
SPIx_I2SPR .....	823
SPIx_RXCRCR .....	820
SPIx_SR .....	818
SPIx_TXCRCR .....	821
SYSCFG_CFGR1 .....	166
SYSCFG_CFGR2 .....	172
SYSCFG_EXTICR1 .....	169
SYSCFG_EXTICR2 .....	170
SYSCFG_EXTICR3 .....	170
SYSCFG_EXTICR4 .....	171
SYSCFG_ITLINE0 .....	172
SYSCFG_ITLINE1 .....	173
SYSCFG_ITLINE10 .....	176
SYSCFG_ITLINE11 .....	177
SYSCFG_ITLINE12 .....	177
SYSCFG_ITLINE13 .....	178
SYSCFG_ITLINE14 .....	178
SYSCFG_ITLINE15 .....	178
SYSCFG_ITLINE16 .....	179
SYSCFG_ITLINE17 .....	179
SYSCFG_ITLINE18 .....	179
SYSCFG_ITLINE19 .....	180
SYSCFG_ITLINE2 .....	173
SYSCFG_ITLINE20 .....	180
SYSCFG_ITLINE21 .....	180
SYSCFG_ITLINE22 .....	181

SYSCFG_ITLINE23 .....	181	TIM15_CCR2 .....	537
SYSCFG_ITLINE24 .....	181	TIM15_CNT .....	536
SYSCFG_ITLINE25 .....	182	TIM15_CR1 .....	522
SYSCFG_ITLINE26 .....	182	TIM15_CR2 .....	523
SYSCFG_ITLINE27 .....	182	TIM15_DCR .....	540
SYSCFG_ITLINE28 .....	183	TIM15_DIER .....	526
SYSCFG_ITLINE29 .....	183	TIM15_DMAR .....	541
SYSCFG_ITLINE3 .....	174	TIM15_EGR .....	529
SYSCFG_ITLINE30 .....	184	TIM15_PSC .....	536
SYSCFG_ITLINE4 .....	174	TIM15_RCR .....	537
SYSCFG_ITLINE5 .....	174	TIM15_SMCR .....	524
SYSCFG_ITLINE6 .....	175	TIM15_SR .....	527
SYSCFG_ITLINE7 .....	175	TIM6_CR2 .....	569
SYSCFG_ITLINE8 .....	176	TIMx_ARR .....	463, 553, 571
SYSCFG_ITLINE9 .....	176	TIMx_BDTR .....	554
<b>T</b>			
TIM1_ARR .....	395	TIMx_CCER .....	460, 550
TIM1_BDTR .....	398	TIMx_CCMR1 .....	386, 456, 547-548
TIM1_CCER .....	391	TIMx_CCMR2 .....	389, 459
TIM1_CCR1 .....	396	TIMx_CCR1 .....	463, 554
TIM1_CCR2 .....	396	TIMx_CCR2 .....	464
TIM1_CCR3 .....	397	TIMx_CCR3 .....	464
TIM1_CCR4 .....	398	TIMx_CCR4 .....	466
TIM1_CNT .....	394	TIMx_CNT .....	462, 553, 570
TIM1_CR1 .....	375	TIMx_CR1 .....	446, 542, 568
TIM1_CR2 .....	376	TIMx_CR2 .....	448, 544
TIM1_DCR .....	400	TIMx_DCR .....	466, 556
TIM1_DIER .....	381	TIMx_DIER .....	451, 544, 569
TIM1_DMAR .....	401	TIMx_DMAR .....	467, 557
TIM1_EGR .....	384	TIMx_EGR .....	455, 546, 570
TIM1_PSC .....	395	TIMx_PSC .....	462, 553, 571
TIM1_RCR .....	395	TIMx_RCR .....	554
TIM1_SMCR .....	378	TIMx_SMCR .....	449
TIM1_SR .....	383	TIMx_SR .....	452, 545, 570
TIM14_ARR .....	489	TSC_CR .....	318
TIM14_CCER .....	487	TSC_ICR .....	321
TIM14_CCMR1 .....	485-486	TSC_IER .....	320
TIM14_CCR1 .....	489	TSC_IOASCR .....	323
TIM14_CNT .....	488	TSC_IOCCR .....	324
TIM14_CR1 .....	482	TSC_IOGCSR .....	324
TIM14_DIER .....	483	TSC_IOGxCR .....	325
TIM14_EGR .....	484	TSC_IOHCR .....	322
TIM14_OR .....	490	TSC_IOSCR .....	323
TIM14_PSC .....	489	TSC_ISR .....	322
TIM14_SR .....	484		
TIM15_ARR .....	536	<b>U</b>	
TIM15_BDTR .....	538	USART_BRR .....	755
TIM15_CCER .....	533	USART_CR1 .....	744
TIM15_CCMR1 .....	530-531	USART_CR2 .....	747
TIM15_CCR1 .....	537	USART_CR3 .....	751
		USART_GTPR .....	755
		USART_ICR .....	763

---

USART_ISR .....	758
USART_RDR .....	764
USART_RQR .....	757
USART_RTOR .....	756
USART_TDR .....	764
USB_ADDRn_RX .....	898
USB_ADDRn_TX .....	897
USB_BCDR .....	891
USB_BTABLE .....	890
USB_CNTR .....	884
USB_COUNTn_RX .....	898
USB_COUNTn_TX .....	897
USB_DADDR .....	889
USB_EPnR .....	892
USB_FNR .....	889
USB_ISTR .....	886
USB_LPMCSR .....	890

**W**

WWDG_CFR .....	587
WWDG_CR .....	586
WWDG_SR .....	587

**IMPORTANT NOTICE – READ CAREFULLY**

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to [www.st.com/trademarks](http://www.st.com/trademarks). All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2022 STMicroelectronics – All rights reserved

