



RM0008

Reference manual

STM32F101xx and STM32F103xx
advanced ARM-based 32-bit MCUs

Introduction

This reference manual targets application developers. It provides complete information on how to use the STM32F101xx and STM32F103xx microcontroller memory and peripherals. The STM32F101xx and STM32F103xx will be referred to as STM32F10xxx throughout the document.

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

For ordering information, mechanical and electrical device characteristics please refer to the *STM32F101xx and STM32F103xx datasheets*.

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

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

Related documents

Available from www.arm.com:

- *Cortex™-M3 Technical Reference Manual*

Available from www.st.com:

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

Contents

1	Documentation conventions	23
1.1	List of abbreviations for registers	23
2	Memory and bus architecture	24
2.1	System architecture	24
2.2	Memory organization	25
2.3	Memory map	26
2.3.1	Peripheral memory map	27
2.3.2	Embedded SRAM	28
2.3.3	Bit banding	28
2.3.4	Embedded Flash memory	29
2.4	Boot configuration	32
3	Power control (PWR)	33
3.1	Power supplies	33
3.1.1	Independent A/D converter supply and reference voltage	33
3.1.2	Battery backup domain	34
3.1.3	Voltage regulator	35
3.2	Power supply supervisor	35
3.2.1	Power on reset (POR)/power down reset (PDR)	35
3.2.2	Programmable voltage detector (PVD)	35
3.3	Low-power modes	37
3.3.1	Slowing down system clocks	37
3.3.2	Peripheral clock gating	37
3.3.3	Sleep mode	38
3.3.4	Stop mode	39
3.3.5	Standby mode	40
3.3.6	Auto Wakeup (AWU) from low-power mode	41
3.4	Power control registers	43
3.4.1	Power control register (PWR_CR)	43
3.4.2	Power control/status register (PWR_CSR)	44
3.5	PWR register map	45

4	Reset and clock control (RCC)	46
4.1	Reset	46
4.1.1	System Reset	46
4.1.2	Power reset	47
4.1.3	Backup domain Reset	47
4.2	Clocks	47
4.2.1	HSE clock	49
4.2.2	HSI clock	50
4.2.3	PLL	50
4.2.4	LSE clock	50
4.2.5	LSI clock	51
4.2.6	System clock (SYSCLK) selection	51
4.2.7	Clock security system (CSS)	52
4.2.8	RTC clock	52
4.2.9	Watchdog clock	52
4.2.10	Clock-out capability	52
4.3	RCC register description	53
4.3.1	Clock control register (RCC_CR)	53
4.3.2	Clock configuration register (RCC_CFGR)	55
4.3.3	Clock interrupt register (RCC_CIR)	58
4.3.4	APB2 Peripheral reset register (RCC_APB2RSTR)	61
4.3.5	APB1 Peripheral reset register (RCC_APB1RSTR)	63
4.3.6	AHB Peripheral Clock enable register (RCC_AHBENR)	65
4.3.7	APB2 Peripheral Clock enable register (RCC_APB2ENR)	66
4.3.8	APB1 Peripheral Clock enable register (RCC_APB1ENR)	68
4.3.9	Backup domain control register (RCC_BDCR)	70
4.3.10	Control/status register (RCC_CSR)	71
4.4	RCC register map	73
5	General-purpose and alternate-function I/Os (GPIOs and AFIOs)	74
5.1	GPIO functional description	74
5.1.1	General-purpose I/O (GPIO)	77
5.1.2	Atomic bit set or bit reset	77
5.1.3	External interrupt/wakeup lines	77
5.1.4	Alternate functions (AF)	77
5.1.5	Software remapping of I/O alternate functions	78

5.1.6	GPIO locking mechanism	78
5.1.7	Input configuration	78
5.1.8	Output configuration	79
5.1.9	Alternate function configuration	79
5.1.10	Analog input configuration	80
5.2	GPIO register description	82
5.2.1	Port configuration register low (GPIOx_CRL) (x=A..E)	82
5.2.2	Port configuration register high (GPIOx_CRH) (x=A..E)	83
5.2.3	Port input data register (GPIOx_IDR) (x=A..E)	84
5.2.4	Port output data register (GPIOx_ODR) (x=A..E)	84
5.2.5	Port bit set/reset register (GPIOx_BSRR) (x=A..E)	85
5.2.6	Port bit reset register (GPIOx_BRR) (x=A..E)	85
5.2.7	Port configuration lock register (GPIOx_LCKR) (x=A..E)	86
5.3	Alternate function I/O and debug configuration (AFIO)	87
5.3.1	Using OSC32_IN/OSC32_OUT pins as GPIO ports PC14/PC15	87
5.3.2	Using OSC_IN/OSC_OUT pins as GPIO ports PD0/PD1	87
5.3.3	BXCAN alternate function remapping	87
5.3.4	JTAG/SWD alternate function remapping	87
5.3.5	Timer alternate function remapping	88
5.3.6	USART Alternate function remapping	90
5.3.7	I2C 1 alternate function remapping	90
5.3.8	SPI 1 alternate function remapping	90
5.4	AFIO register description	91
5.4.1	Event control register (AFIO_EVCR)	91
5.4.2	AF remap and debug I/O configuration register (AFIO_MAPR)	92
5.4.3	External interrupt configuration register 1 (AFIO_EXTICR1)	94
5.4.4	External interrupt configuration register 2 (AFIO_EXTICR2)	94
5.4.5	External interrupt configuration register 3 (AFIO_EXTICR3)	95
5.4.6	External interrupt configuration register 4 (AFIO_EXTICR4)	95
5.5	GPIO and AFIO register maps	96
5.5.1	GPIO register map	96
5.5.2	AFIO register map	96
6	Interrupts and events	97
6.1	Nested vectored interrupt controller (NVIC)	97
6.1.1	SysTick calibration value register	97
6.1.2	Interrupt and exception vectors	97

6.2	External interrupt/event controller (EXTI)	100
6.2.1	Main features	100
6.2.2	Block diagram	100
6.2.3	Wakeup event management	101
6.2.4	Functional description	101
6.2.5	External interrupt/event line mapping	102
6.3	EXTI register description	102
6.3.1	EXTI register map	106
7	DMA controller (DMA)	107
7.1	Introduction	107
7.2	Main features	107
7.3	Functional description	108
7.3.1	DMA transactions	108
7.3.2	Arbiter	109
7.3.3	DMA channels	109
7.3.4	Error management	110
7.3.5	Interrupts	111
7.3.6	DMA request mapping	112
7.4	DMA registers	113
7.4.1	DMA interrupt status register (DMA_ISR)	113
7.4.2	DMA interrupt flag clear register (DMA_IFCR)	115
7.4.3	DMA channel x configuration register (DMA_CCRx) (x = 1 ..7)	116
7.4.4	DMA channel x number of data register (DMA_CNDTRx) (x = 1 ..7)	117
7.4.5	DMA channel x peripheral address register (DMA_CPARx) (x = 1 ..7)	118
7.4.6	DMA channel x memory address register (DMA_CMARx) (x = 1 ..7)	118
7.5	DMA register map	118
8	Real-time clock (RTC)	121
8.1	Introduction	121
8.2	Main features	121
8.3	Functional description	121
8.3.1	Overview	121
8.3.2	Resetting RTC registers	123
8.3.3	Reading RTC registers	123
8.3.4	Configuring RTC registers	124

8.3.5	RTC flag assertion	124
8.4	RTC register description	125
8.4.1	RTC control register High (RTC_CRH)	125
8.4.2	RTC control register low (RTC_CRL)	126
8.4.3	RTC prescaler load register (RTC_PRLH / RTC_PRLL)	128
8.4.4	RTC prescaler divider register (RTC_DIVH / RTC_DIVL)	129
8.4.5	RTC counter register (RTC_CNTH / RTC_CNTL)	130
8.4.6	RTC alarm register high (RTC_ALRH / RTC_ALRL)	131
8.5	RTC register map	132
9	Backup registers (BKP)	133
9.1	Introduction	133
9.2	Features	133
9.3	Tamper detection	133
9.4	RTC calibration	134
9.5	BKP register description	134
9.5.1	Backup data register x (BKP_DRx) (x = 1 ..10)	134
9.5.2	RTC clock calibration register (BKP_RTCCR)	134
9.5.3	Backup control register (BKP_CR)	135
9.5.4	Backup control/status register (BKP_CSR)	136
9.6	BKP register map	137
10	Independent watchdog (IWDG)	138
10.1	Introduction	138
10.1.1	Hardware watchdog	138
10.1.2	Register access protection	138
10.1.3	Debug mode	138
10.2	IWDG register description	140
10.2.1	Key register (IWDG_KR)	140
10.2.2	Prescaler register (IWDG_PR)	140
10.2.3	Reload register (IWDG_RLR)	141
10.2.4	Status register (IWDG_SR)	142
10.3	IWDG register map	143
11	Window watchdog (WWDG)	144
11.1	Introduction	144

11.2	Main features	144
11.3	Functional description	144
11.4	How to program the watchdog timeout	145
11.5	Debug mode	146
11.6	Register description	147
11.6.1	Control Register (WWDG_CR)	147
11.6.2	Configuration register (WWDG_CFR)	147
11.6.3	Status register (WWDG_SR)	148
11.7	WWDG register map	148
12	Advanced control timer (TIM1)	149
12.1	Introduction	149
12.2	Main features	149
12.3	Block diagram	150
12.4	Functional description	151
12.4.1	Time base unit	151
12.4.2	Counter modes	152
12.4.3	Repetition downcounter	160
12.4.4	Clock selection	162
12.4.5	Capture/compare channels	164
12.4.6	Input capture mode	166
12.4.7	PWM input mode	167
12.4.8	Forced output mode	168
12.4.9	Output compare mode	169
12.4.10	PWM mode	170
12.4.11	Complementary outputs and dead-time insertion	173
12.4.12	Using the break function	174
12.4.13	Clearing the OCxREF signal on an external event	177
12.4.14	6-step PWM generation	178
12.4.15	One-pulse mode	179
12.4.16	Encoder interface mode	180
12.4.17	Timer input XOR function	182
12.4.18	Interfacing with Hall sensors	182
12.4.19	TIM1 and external trigger synchronization	185
12.4.20	Timer synchronization	188
12.4.21	Debug mode	188

12.5	TIM1 register description	189
12.5.1	Control register 1 (TIM1_CR1)	189
12.5.2	Control register 2 (TIM1_CR2)	191
12.5.3	Slave mode control register (TIM1_SMCR)	193
12.5.4	DMA/Interrupt enable register (TIM1_DIER)	196
12.5.5	Status register (TIM1_SR)	197
12.5.6	Event generation register (TIM1_EGR)	199
12.5.7	Capture/compare mode register 1 (TIM1_CCMR1)	200
12.5.8	Capture/compare mode register 2 (TIM1_CCMR2)	203
12.5.9	Capture/compare enable register (TIM1_CCER)	204
12.5.10	Counter (TIM1_CNT)	207
12.5.11	Prescaler (TIM1_PSC)	207
12.5.12	Auto-reload register (TIM1_ARR)	207
12.5.13	Repetition counter register (TIM1_RCR)	208
12.5.14	Capture/compare register 1 (TIM1_CCR1)	208
12.5.15	Capture/compare register 2 (TIM1_CCR2)	209
12.5.16	Capture/compare register 3 (TIM1_CCR3)	209
12.5.17	Capture/compare register 4 (TIM1_CCR4)	210
12.5.18	Break and dead-time register (TIM1_BDTR)	210
12.5.19	DMA control register (TIM1_DCR)	212
12.5.20	DMA address for burst mode (TIM1_DMAR)	213
12.6	TIM1 register map	213
13	General purpose timer (TIMx)	215
13.1	Introduction	215
13.2	Main features	215
13.3	Block diagram	216
13.4	Functional description	216
13.4.1	Time base unit	216
13.4.2	Counter modes	218
13.4.3	Clock selection	226
13.4.4	Capture/compare channels	229
13.4.5	Input capture mode	231
13.4.6	PWM input mode	232
13.4.7	Forced output mode	232
13.4.8	Output compare mode	233

13.4.9	PWM mode	234
13.4.10	One pulse mode	237
13.4.11	Clearing the OC _x REF signal on an external event	239
13.4.12	Encoder interface mode	240
13.4.13	Timer input XOR function	242
13.4.14	Timers and external trigger synchronization	243
13.4.15	Timer synchronization	246
13.4.16	Debug mode	251
13.5	TIMx register description	252
13.5.1	Control register 1 (TIMx_CR1)	252
13.5.2	Control register 2 (TIMx_CR2)	254
13.5.3	Slave mode control register (TIMx_SMCR)	255
13.5.4	DMA/Interrupt enable register (TIMx_DIER)	258
13.5.5	Status register (TIMx_SR)	259
13.5.6	Event generation register (TIMx_EGR)	261
13.5.7	Capture/compare mode register 1 (TIMx_CCMR1)	262
13.5.8	Capture/compare mode register 2 (TIMx_CCMR2)	265
13.5.9	Capture/compare enable register (TIMx_CCER)	267
13.5.10	Counter (TIMx_CNT)	268
13.5.11	Prescaler (TIMx_PSC)	268
13.5.12	Auto-reload register (TIMx_ARR)	268
13.5.13	Capture/compare register 1 (TIMx_CCR1)	269
13.5.14	Capture/compare register 2 (TIMx_CCR2)	269
13.5.15	Capture/compare register 3 (TIMx_CCR3)	270
13.5.16	Capture/compare register 4 (TIMx_CCR4)	270
13.5.17	DMA control register (TIMx_DCR)	271
13.5.18	DMA address for burst mode (TIMx_DMAR)	271
13.6	TIMx register map	272
14	Controller area network (bxCAN)	274
14.1	Introduction	274
14.2	Main features	274
14.3	General description	275
14.3.1	CAN 2.0B active core	275
14.3.2	Control, status and configuration registers	275
14.3.3	Tx mailboxes	275
14.3.4	Acceptance filters	276

14.3.5	Receive FIFO	276
14.4	Operating modes	277
14.4.1	Initialization mode	277
14.4.2	Normal mode	277
14.4.3	Sleep mode (low power)	278
14.4.4	Test mode	278
14.4.5	Silent mode	278
14.4.6	Loop back mode	279
14.4.7	Loop back combined with silent mode	279
14.5	Functional description	280
14.5.1	Transmission handling	280
14.5.2	Time triggered communication mode	281
14.5.3	Reception handling	281
14.5.4	Identifier filtering	283
14.5.5	Message storage	287
14.5.6	Error management	289
14.5.7	Bit timing	289
14.6	Interrupts	291
14.7	Register access protection	293
14.8	CAN register description	293
14.8.1	Control and status registers	293
14.8.2	Mailbox registers	304
14.8.3	CAN filter registers	310
14.9	bxCAN register map	314
15	Inter-integrated circuit (I²C) interface	317
15.1	Introduction	317
15.2	Main features	317
15.3	General description	318
15.4	Functional description	321
15.4.1	I ² C slave mode	321
15.4.2	I ² C master mode	324
15.4.3	Error conditions	328
15.4.4	SDA/SCL line control	329
15.4.5	SMBus	329
15.4.6	DMA requests	332

15.4.7	Packet error checking	333
15.5	Interrupt requests	334
15.6	I ² C debug mode	335
15.7	I ² C register description	335
15.7.1	Control register 1(I2C_CR1)	335
15.7.2	Control register 2 (I2C_CR2)	338
15.7.3	Own address register 1 (I2C_OAR1)	339
15.7.4	Own address register 2 (I2C_OAR2)	339
15.7.5	Data register (I2C_DR)	340
15.7.6	Status register 1 (I2C_SR1)	341
15.7.7	Status register 2 (I2C_SR2)	344
15.7.8	Clock control register (I2C_CCR)	345
15.7.9	TRISE Register (I2C_TRISE)	346
15.8	I ² C register map	347
16	Analog-to-digital converter (ADC)	348
16.1	Introduction	348
16.2	Main features	348
16.3	Pin description	350
16.4	Functional description	350
16.4.1	ADC on-off control	350
16.4.2	ADC clock	350
16.4.3	Channel selection	350
16.4.4	Single conversion mode	351
16.4.5	Continuous conversion mode	351
16.4.6	Timing diagram	351
16.4.7	Analog watchdog	352
16.4.8	Scan mode	353
16.4.9	Injected channel management	354
16.4.10	Discontinuous mode	355
16.5	Calibration	356
16.6	Data alignment	357
16.7	Channel-by-channel programmable sample time	357
16.8	Conversion on external trigger	357
16.9	DMA request	358

16.10	Dual ADC mode	359
16.10.1	Injected simultaneous mode	361
16.10.2	Regular simultaneous mode	361
16.10.3	Fast interleaved mode	362
16.10.4	Slow interleaved mode	362
16.10.5	Alternate trigger mode	363
16.10.6	Independent mode	364
16.10.7	Combined regular/injected simultaneous mode	364
16.10.8	Combined regular simultaneous + alternate trigger mode	364
16.10.9	Combined injected simultaneous + interleaved	365
16.11	Temperature sensor	366
16.12	Interrupts	367
16.13	ADC register description	368
16.13.1	ADC status register (ADC_SR)	368
16.13.2	ADC control register 1 (ADC_CR1)	369
16.13.3	ADC control register 2 (ADC_CR2)	371
16.13.4	ADC sample time register 1 (ADC_SMPR1)	374
16.13.5	ADC sample time register 2 (ADC_SMPR2)	375
16.13.6	ADC injected channel data offset register x (ADC_JOFR _x)(x=1..4)	376
16.13.7	ADC watchdog high threshold register (ADC_HTR)	376
16.13.8	ADC watchdog low threshold register (ADC_LTR)	377
16.13.9	ADC regular sequence register 1 (ADC_SQR1)	377
16.13.10	ADC regular sequence register 2 (ADC_SQR2)	378
16.13.11	ADC regular sequence register 3 (ADC_SQR3)	378
16.13.12	ADC injected sequence register (ADC_JSQR)	379
16.13.13	ADC injected data register x (ADC_JDR _x) (x= 1..4)	380
16.13.14	ADC regular data register (ADC_DR)	380
16.14	ADC register map	381
17	USB full speed device interface (USB)	383
17.1	Introduction	383
17.2	Main features	383
17.3	Block diagram	383
17.4	Functional description	384
17.4.1	Description of USB blocks	385
17.5	Programming considerations	386

17.5.1	Generic USB device programming	386
17.5.2	System and power-on reset	386
17.5.3	Double-buffered endpoints	393
17.5.4	Isochronous transfers	395
17.5.5	Suspend/Resume events	396
17.6	USB register description	398
17.6.1	Common registers	398
17.6.2	Endpoint-specific registers	405
17.6.3	Buffer descriptor table	408
17.7	USB Register map	413
18	Serial peripheral interface (SPI)	414
18.1	Introduction	414
18.2	Main features	414
18.2.1	SPI features	414
18.3	SPI functional description	415
18.3.1	General description	415
18.3.2	SPI slave mode	418
18.3.3	SPI master mode	419
18.3.4	Simplex communication	420
18.3.5	Status flags	421
18.3.6	CRC calculation	421
18.3.7	SPI communication using DMA (direct memory addressing)	422
18.3.8	Error flags	423
18.3.9	Disabling the SPI	423
18.3.10	Interrupts	424
18.4	SPI register description	424
18.4.1	SPI Control Register 1 (SPI_CR1)	424
18.4.2	SPI control register 2 (SPI_CR2)	427
18.4.3	SPI status register (SPI_SR)	428
18.4.4	SPI data register (SPI_DR)	429
18.4.5	SPI Rx CRC register (SPI_RXCRCR)	429
18.4.6	SPI Tx CRC register (SPI_TXCRCR)	430
18.5	SPI register map	431
19	Universal synchronous asynchronous receiver transmitter (USART)	432

19.1	Introduction	432
19.2	Main features	432
19.3	General description	433
19.3.1	Block diagram	435
19.3.2	USART character description	436
19.3.3	Transmitter	437
19.3.4	Receiver	439
19.3.5	Fractional baud rate generation	442
19.3.6	Multiprocessor communication	444
19.3.7	Parity control	445
19.3.8	LIN (local interconnection network) mode	446
19.3.9	USART synchronous mode	448
19.3.10	Single wire half duplex communication	450
19.3.11	Smartcard	451
19.3.12	IrDA SIR ENDEC block	453
19.3.13	Continuous communication using DMA	454
19.3.14	Hardware flow control	456
19.4	Interrupt requests	458
19.5	USART register description	459
19.5.1	Status register (USART_SR)	459
19.5.2	Data register (USART_DR)	461
19.5.3	Baud Rate Register (USART_BRR)	461
19.5.4	Control register 1 (USART_CR1)	462
19.5.5	Control register 2 (USART_CR2)	464
19.5.6	Control register 3 (USART_CR3)	466
19.5.7	Guard time and prescaler register (USART_GTPR)	468
19.6	USART register map	469
20	Debug support (DBG)	470
20.1	Overview	470
20.2	Referenced ARM documentation	471
20.3	SWJ debug port (serial wire and JTAG)	471
20.3.1	Mechanism to select the JTAG-DP or the SW-DP	472
20.4	Pinout and debug port pins	473
20.4.1	SWJ debug port pins	473
20.4.2	Flexible SWJ-DP pin assignment	473

20.4.3	Internal pull-up and pull-down on JTAG pins	474
20.4.4	Using serial wire and releasing the unused debug pins as GPIOs	475
20.5	STM32F10xxx JTAG TAP connection	475
20.6	ID codes and locking mechanism	476
20.6.1	MCU device ID code	476
20.6.2	TMC TAP	477
20.6.3	Cortex-M3 TAP	477
20.6.4	Cortex-M3 JEDEC-106 ID code	477
20.7	JTAG debug port	477
20.8	SW debug port	479
20.8.1	SW protocol introduction	479
20.8.2	SW protocol sequence	479
20.8.3	SW-DP state machine (Reset, idle states, ID code)	480
20.8.4	DP and AP read/write accesses	481
20.8.5	SW-DP registers	481
20.8.6	SW-AP registers	482
20.9	AHB-AP (AHB Access Port) - valid for both JTAG-DP or SW-DP	482
20.10	Core debug	483
20.11	Capability of the debugger host to connect under system reset	484
20.12	FPB (Flash patch breakpoint)	484
20.13	DWT (data watchpoint trigger)	484
20.14	ITM (instrumentation trace macrocell)	485
20.14.1	General description	485
20.14.2	Timestamp packets, synchronization and overflow packets	485
20.15	MCU debug component (MCUDBG)	487
20.15.1	Debug support for low-power modes	487
20.15.2	Debug support for timers, watchdog, bxCAN and I ² C	487
20.15.3	Debug MCU configuration register	487
20.16	TPIU (trace port interface unit)	489
20.16.1	Introduction	489
20.16.2	TRACE pin assignment	491
20.16.3	TPUI formatter	492
20.16.4	TPUI frame synchronization packets	493
20.16.5	Emission of synchronization frame packet	493
20.16.6	Synchronous mode	493
20.16.7	Asynchronous mode	494

20.16.8 TRACECLKIN connection inside STM32F10xxx	494
20.16.9 TPIU registers	494
20.16.10 Example of configuration	495
20.17 DBG register map	496
Revision history	497

List of tables

Table 1.	Register boundary addresses	27
Table 2.	Flash module organization	30
Table 3.	Boot modes	32
Table 4.	Low-power mode summary	37
Table 5.	Sleep-now	38
Table 6.	Sleep-on-exit	39
Table 7.	Stop mode	40
Table 8.	Standby mode	41
Table 9.	PWR - register map and reset values	45
Table 10.	RCC - register map and reset values	73
Table 11.	Port bit configuration table	76
Table 12.	Output MODE bits	76
Table 13.	BXCAN alternate function remapping	87
Table 14.	Debug interface signals	87
Table 15.	Debug port mapping	88
Table 16.	Timer 4 alternate function remapping	88
Table 17.	Timer 3 alternate function remapping	89
Table 18.	Timer 2 alternate function remapping	89
Table 19.	Timer 1 alternate function remapping	89
Table 20.	USART3 remapping	90
Table 21.	USART2 remapping	90
Table 22.	USART1 remapping	90
Table 23.	I2C1 remapping	90
Table 24.	SPI1 remapping	91
Table 25.	GPIO register map and reset values	96
Table 26.	AFIO register map and reset values	96
Table 27.	Vector table	97
Table 28.	External interrupt/event controller register map and reset values	106
Table 29.	DMA interrupt requests	111
Table 30.	Summary of DMA requests for each channel	113
Table 31.	DMA - register map and reset values	118
Table 32.	RTC - register map and reset values	132
Table 33.	BKP - register map and reset values	137
Table 34.	Watchdog timeout period (with 40 kHz input clock)	139
Table 35.	IWDG register map and reset values	143
Table 36.	WWDG register map and reset values	148
Table 37.	Counting direction versus encoder signals	181
Table 38.	Output control bits for complementary OCx and OCxN channels with break feature	206
Table 39.	TIM1 - Register map and reset values	213
Table 40.	Counting direction versus encoder signals	240
Table 41.	Output control bit for standard OCx channels	268
Table 42.	TIMx - register map and reset values	272
Table 43.	Transmit mailbox mapping	288
Table 44.	Receive mailbox mapping	288
Table 45.	bxCAN - register map and reset values	314
Table 46.	SMBus vs. I2C	330
Table 47.	I2C Interrupt requests	334
Table 48.	I2C register map and reset values	347

Table 49.	ADC pins	350
Table 50.	Analog watchdog channel selection	352
Table 51.	External trigger for regular channels	358
Table 52.	External trigger for injected channels	358
Table 53.	ADC interrupts	367
Table 54.	ADC - register map and reset values	381
Table 55.	Double-buffering buffer flag definition	394
Table 56.	Bulk double-buffering memory buffers usage	394
Table 57.	Isochronous memory buffers usage	396
Table 58.	Resume event detection	397
Table 59.	Reception status encoding	408
Table 60.	Endpoint type encoding	408
Table 61.	Endpoint kind meaning	408
Table 62.	Transmission status encoding	408
Table 63.	Definition of allocated buffer memory	412
Table 64.	USB register map and reset values	413
Table 65.	SPI interrupt requests	424
Table 66.	SPI register map and reset values	431
Table 67.	Noise detection from sampled data	441
Table 68.	Error calculation for programmed baud rates	443
Table 69.	Frame formats	445
Table 70.	USART interrupt requests	458
Table 71.	USART register map and reset values	469
Table 72.	SWJ debug port pins	473
Table 73.	Flexible SWJ-DP pin assignment	474
Table 74.	JTAG debug port data registers	477
Table 75.	32-bit debug port registers addressed through the shifted value A[3:2]	478
Table 76.	Packet request (8-bits)	479
Table 77.	ACK response (3 bits)	480
Table 78.	DATA transfer (33 bits)	480
Table 79.	SW-DP registers	481
Table 80.	Cortex-M3 AHB-AP registers	482
Table 81.	Core debug registers	483
Table 82.	Main ITM registers	486
Table 83.	Asynchronous TRACE pin assignment	491
Table 84.	Synchronous TRACE pin assignment	491
Table 85.	Flexible TRACE pin assignment	492
Table 86.	Important TPIU registers	495
Table 87.	DBG - register map and reset values	496
Table 88.	Document revision history	497

List of figures

Figure 1.	System architecture	24
Figure 2.	Memory map	26
Figure 3.	Power supply overview	33
Figure 4.	Power on reset/power down reset waveform	35
Figure 5.	PVD thresholds	36
Figure 6.	Reset circuit	47
Figure 7.	Clock tree	48
Figure 8.	HSE/ LSE clock sources	49
Figure 9.	Basic structure of a standard I/O port bit	75
Figure 10.	Basic structure of a five-volt tolerant I/O port bit	75
Figure 11.	Input floating/pull up/pull down configurations	78
Figure 12.	Output configuration	79
Figure 13.	Alternate function configuration	80
Figure 14.	High impedance-analog input configuration	81
Figure 15.	External interrupt/event controller block diagram	100
Figure 16.	External interrupt/event GPIO mapping	102
Figure 17.	DMA block diagram	108
Figure 18.	DMA request mapping	112
Figure 19.	RTC simplified block diagram	122
Figure 20.	RTC second and alarm waveform example with PR=0003, ALARM=00004	124
Figure 21.	RTC Overflow waveform example with PR=0003	125
Figure 22.	Independent watchdog block diagram	139
Figure 23.	Watchdog block diagram	145
Figure 24.	Window watchdog timing diagram	146
Figure 25.	Advanced control timer (TIM1) block diagram	150
Figure 26.	Counter timing diagram with prescaler division change from 1 to 2	152
Figure 27.	Counter timing diagram with prescaler division change from 1 to 4	152
Figure 28.	Counter timing diagram, internal clock divided by 1	153
Figure 29.	Counter timing diagram, internal clock divided by 2	153
Figure 30.	Counter timing diagram, internal clock divided by 4	154
Figure 31.	Counter timing diagram, internal clock divided by N	154
Figure 32.	Counter timing diagram, update event when ARPE=0 (TIM1_ARR not preloaded)	154
Figure 33.	Counter timing diagram, update event when ARPE=1 (TIM1_ARR preloaded)	155
Figure 34.	Counter timing diagram, internal clock divided by 1	156
Figure 35.	Counter timing diagram, internal clock divided by 2	156
Figure 36.	Counter timing diagram, internal clock divided by 4	156
Figure 37.	Counter timing diagram, internal clock divided by N	157
Figure 38.	Counter timing diagram, update event when repetition counter is not used	157
Figure 39.	Counter timing diagram, internal clock divided by 1, TIM1_ARR=0x6	158
Figure 40.	Counter timing diagram, internal clock divided by 2	158
Figure 41.	Counter timing diagram, internal clock divided by 4, TIM1_ARR=0x36	159
Figure 42.	Counter timing diagram, internal clock divided by N	159
Figure 43.	Counter timing diagram, update event with ARPE=1 (counter underflow)	159
Figure 44.	Counter timing diagram, Update event with ARPE=1 (counter overflow)	160
Figure 45.	Update rate examples depending on mode and TIM1_RCR register settings	161
Figure 46.	Control circuit in normal mode, internal clock divided by 1	162
Figure 47.	TI2 external clock connection example	162
Figure 48.	Control circuit in external clock mode 1	163

Figure 49.	External trigger input block	163
Figure 50.	Control circuit in external clock mode 2	164
Figure 51.	Capture/compare channel (example: channel 1 input stage)	165
Figure 52.	Capture/compare channel 1 main circuit	165
Figure 53.	Output stage of capture/compare channel (channel 1 to 3)	166
Figure 54.	Output stage of capture/compare channel (channel 4)	166
Figure 55.	PWM input mode timing	168
Figure 56.	Output compare mode, toggle on OC1.	170
Figure 57.	Edge-aligned PWM waveforms (ARR=8)	171
Figure 58.	Center-aligned PWM waveforms (ARR=8)	172
Figure 59.	Complementary output with dead-time insertion	173
Figure 60.	Dead-time waveforms with delay greater than the negative pulse.	173
Figure 61.	Dead-time waveforms with delay greater than the positive pulse.	174
Figure 62.	Output behavior in response to a break.	176
Figure 63.	Clearing TIM1 OCxREF	177
Figure 64.	6-step generation, COM example (OSSR=1)	178
Figure 65.	Example of one pulse mode	179
Figure 66.	Example of counter operation in encoder interface mode.	181
Figure 67.	Example of encoder interface mode with TI1FP1 polarity inverted.	182
Figure 68.	Example of hall sensor interface.	184
Figure 69.	Control circuit in reset mode	185
Figure 70.	Control circuit in gated mode	186
Figure 71.	Control circuit in trigger mode.	187
Figure 72.	Control circuit in external clock mode 2 + trigger mode	188
Figure 73.	General-purpose timer block diagram	216
Figure 74.	Counter timing diagram with prescaler division change from 1 to 2	217
Figure 75.	Counter timing diagram with prescaler division change from 1 to 4	218
Figure 76.	Counter timing diagram, internal clock divided by 1	219
Figure 77.	Counter timing diagram, internal clock divided by 2	219
Figure 78.	Counter timing diagram, internal clock divided by 4	219
Figure 79.	Counter timing diagram, internal clock divided by N.	220
Figure 80.	Counter timing diagram, Update event when ARPE=0 (TIMx_ARR not preloaded).	220
Figure 81.	Counter timing diagram, Update event when ARPE=1 (TIMx_ARR preloaded).	221
Figure 82.	Counter timing diagram, internal clock divided by 1	222
Figure 83.	Counter timing diagram, internal clock divided by 2	222
Figure 84.	Counter timing diagram, internal clock divided by 4	222
Figure 85.	Counter timing diagram, internal clock divided by N.	223
Figure 86.	Counter timing diagram, Update event when repetition counter is not used	223
Figure 87.	Counter timing diagram, internal clock divided by 1, TIMx_ARR=0x6	224
Figure 88.	Counter timing diagram, internal clock divided by 2	224
Figure 89.	Counter timing diagram, internal clock divided by 4, TIMx_ARR=0x36	225
Figure 90.	Counter timing diagram, internal clock divided by N.	225
Figure 91.	Counter timing diagram, Update event with ARPE=1 (counter underflow).	225
Figure 92.	Counter timing diagram, Update event with ARPE=1 (counter overflow).	226
Figure 93.	Control circuit in normal mode, internal clock divided by 1.	227
Figure 94.	TI2 external clock connection example.	227
Figure 95.	Control circuit in external clock mode 1	228
Figure 96.	External trigger input block	228
Figure 97.	Control circuit in external clock mode 2	229
Figure 98.	Capture/compare channel (example: channel 1 input stage)	229
Figure 99.	Capture/compare channel 1 main circuit	230
Figure 100.	Output stage of capture/compare channel (channel 1).	230

Figure 101. PWM input mode timing.....	232
Figure 102. Output compare mode, toggle on OC1.....	234
Figure 103. Edge-aligned PWM waveforms (ARR=8).....	235
Figure 104. Center-aligned PWM waveforms (ARR=8).....	236
Figure 105. Example of one pulse mode.....	237
Figure 106. Clearing TIMx OCxREF	239
Figure 107. Example of counter operation in encoder interface mode.....	241
Figure 108. Example of encoder interface mode with IC1FP1 polarity inverted.....	241
Figure 109. Control circuit in reset mode	243
Figure 110. Control circuit in gated mode	244
Figure 111. Control circuit in trigger mode.....	245
Figure 112. Control circuit in external clock mode 2 + trigger mode	246
Figure 113. Master/Slave timer example	247
Figure 114. Gating timer 2 with OC1REF of timer 1	248
Figure 115. Gating timer 2 with Enable of timer 1	249
Figure 116. Triggering timer 2 with Update of timer 1	249
Figure 117. Triggering timer 2 with Enable of timer 1	250
Figure 118. Triggering timer 1 and 2 with timer 1 TI1 input.....	251
Figure 119. CAN network topology	275
Figure 120. CAN block diagram.....	276
Figure 121. bxCAN operating modes.....	276
Figure 122. bxCAN in silent mode	279
Figure 123. bxCAN in loop back mode	279
Figure 124. bxCAN in combined mode	280
Figure 125. Transmit mailbox states	281
Figure 126. Receive FIFO states.....	282
Figure 127. Filter bank scale configuration - register organization	285
Figure 128. Example of filter numbering	286
Figure 129. Filtering mechanism - example	287
Figure 130. CAN error state diagram.....	288
Figure 131. Bit timing	290
Figure 132. CAN frames	291
Figure 133. Event flags and interrupt generation.....	292
Figure 134. I2C bus protocol	319
Figure 135. I2C block diagram.....	320
Figure 136. Transfer sequence diagram for slave transmitter.....	322
Figure 137. Transfer sequence diagram for slave receiver	323
Figure 138. Transfer sequence diagram for master transmitter.....	326
Figure 139. Transfer sequence diagram for master receiver	327
Figure 140. I2C interrupt mapping diagram	335
Figure 141. Single ADC block diagram	349
Figure 142. Timing diagram	352
Figure 143. Analog watchdog guarded area	352
Figure 144. Injected conversion latency	354
Figure 145. Calibration timing diagram	356
Figure 146. Right alignment of data	357
Figure 147. Left alignment of data	357
Figure 148. Dual ADC block diagram	360
Figure 149. Injected simultaneous mode on 4 channels	361
Figure 150. Regular simultaneous mode on 16 channels	362
Figure 151. Fast interleaved mode on 1 channel in continuous conversion mode	362
Figure 152. Slow interleaved mode on 1 channel	363

Figure 153. Alternate trigger: injected channel group of each ADC	363
Figure 154. Alternate trigger: 4 injected channels (each ADC) in discontinuous mode	364
Figure 155. Alternate + Regular simultaneous	365
Figure 156. Case of trigger occurring during injected conversion	365
Figure 157. Interleaved single channel with injected sequence CH11, CH12	365
Figure 158. Temperature sensor and VREFINT channel block diagram	366
Figure 159. USB peripheral block diagram	384
Figure 160. Packet buffer areas with examples of buffer description table locations	389
Figure 161. SPI block diagram	415
Figure 162. Single master/ single slave application	416
Figure 163. Hardware/software slave select management	416
Figure 164. Data clock timing diagram	418
Figure 165. USART block diagram	435
Figure 166. Word length programming	436
Figure 167. Configurable stop bits	438
Figure 168. Data sampling for noise detection	441
Figure 169. Mute mode using Idle line detection	444
Figure 170. Mute mode using Address mark detection	445
Figure 171. Break detection in LIN mode (11-bit break length - LBDL bit is set)	447
Figure 172. Break detection in LIN mode vs. Framing error detection	448
Figure 173. USART example of synchronous transmission	449
Figure 174. USART data clock timing diagram (M=0)	449
Figure 175. USART data clock timing diagram (M=1)	450
Figure 176. RX data setup/hold time	450
Figure 177. ISO 7816-3 asynchronous protocol	451
Figure 178. Parity error detection using the 1.5 stop bits	452
Figure 179. IrDA SIR ENDEC- block diagram	454
Figure 180. IrDA data modulation (3/16) -Normal Mode	454
Figure 181. Hardware flow control between 2 USART	456
Figure 182. RTS flow control	456
Figure 183. CTS flow control	457
Figure 184. USART interrupt mapping diagram	458
Figure 185. Block diagram of STM32F10xxx-level and Cortex-M3-level debug support	470
Figure 186. SWJ debug port	472
Figure 187. JTAG TAP connections	476
Figure 188. TPIU block diagram	490

1 Documentation conventions

1.1 List of abbreviations for registers

The following abbreviations are used in register descriptions:

read/write (rw)	Software can read and write to these bits.
read-only (r)	Software can only read these bits.
write-only (w)	Software can only write to this bit. Reading the bit returns the reset value.
read-clear (rc)	The software can only read or clear this bit.
read/clear (rc_w1)	Software can read as well as clear this bit by writing 1. Writing '0' has no effect on the bit value.
read/clear (rc_w0)	Software can read as well as clear this bit by writing 0. Writing '1' has no effect on the bit value.
read/set (rs)	Software can read as well as set this bit. Writing '0' has no effect on the bit value.
toggle (t)	The software can only toggle this bit by writing '1'. Writing '0' has no effect.

2 Memory and bus architecture

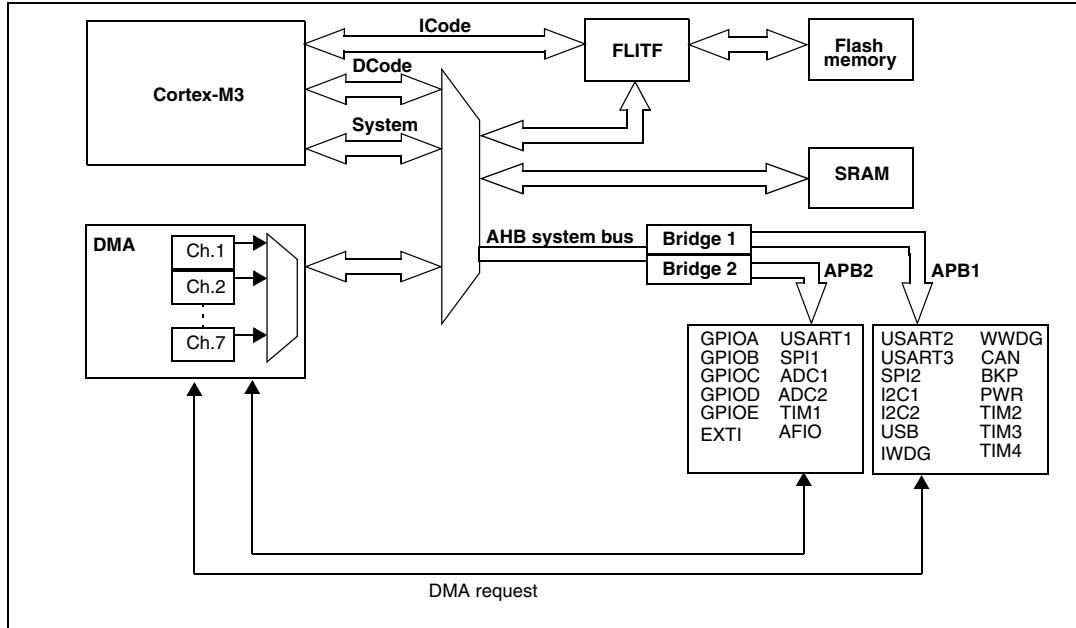
2.1 System architecture

The main system consists of:

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

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

Figure 1. System architecture



ICode bus

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

DCode bus

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

System bus

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

DMA bus

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

BusMatrix

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

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

AHB/APB bridges (APB)

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

Refer to [Table 1 on page 27](#) for the address mapping of the peripherals connected to each bridge.

2.2 Memory organization

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

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

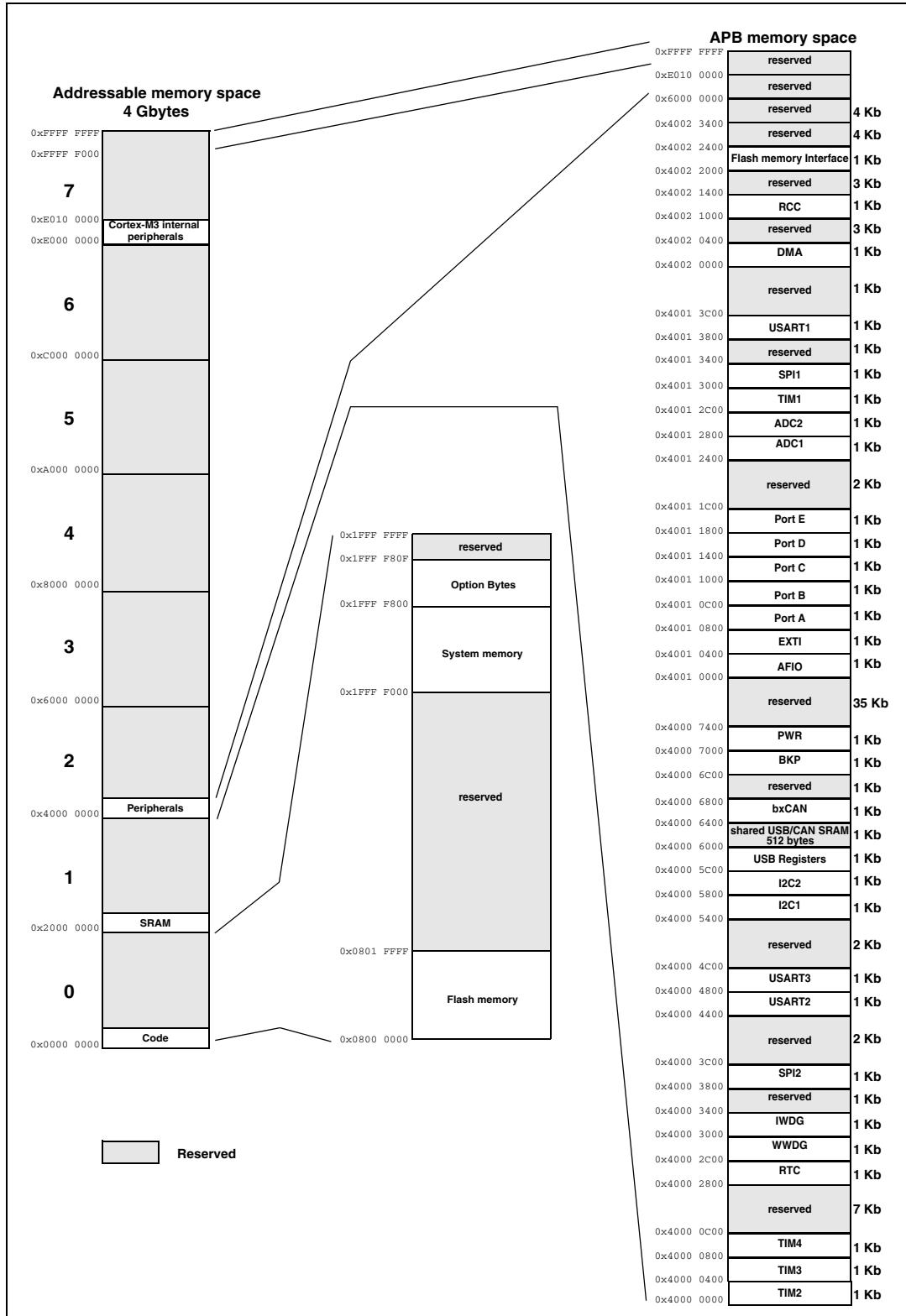
[Figure 2 on page 26](#) shows the STM32F10xxx Memory Map. For the detailed mapping of peripheral registers, please refer to the related chapters.

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

All the memory areas that are not allocated to on-chip memories and peripherals are considered "Reserved" (gray shaded areas in the [Figure 2 on page 26](#)).

2.3 Memory map

Figure 2. Memory map



2.3.1 Peripheral memory map

Table 1. Register boundary addresses

Boundary address	Peripheral	Bus	Register map
0x4002 2400 - 0x4002 3FFF	Reserved	AHB	
0x4002 2000 - 0x4002 23FF	Flash memory interface		
0x4002 1400 - 0x4002 1FFF	Reserved		
0x4002 1000 - 0x4002 13FF	Reset and Clock control RCC		Section 4.4 on page 73
0x4002 0400 - 0x4002 0FFF	Reserved		
0x4002 0000 - 0x4002 03FF	DMA		Section 7.5 on page 118
0x4001 3C00 - 0x4001 3FFF	Reserved	APB2	
0x4001 3800 - 0x4001 3BFF	USART1		Section 19.6 on page 469
0x4001 3400 - 0x4001 37FF	Reserved		
0x4001 3000 - 0x4001 33FF	SPI 1		Section 18.5 on page 431
0x4001 2C00 - 0x4001 2FFF	TIM1 timer		Section 12.6 on page 213
0x4001 2800 - 0x4001 2BFF	ADC2		Section 16.14 on page 381
0x4001 2400 - 0x4001 27FF	ADC1		Section 16.14 on page 381
0x4001 2000 - 0x4001 1FFF	Reserved		
0x4001 1800 - 0x4001 1BFF	GPIO Port E		Section 5.5.1 on page 96
0x4001 1400 - 0x4001 17FF	GPIO Port D		Section 5.5.1 on page 96
0x4001 1000 - 0x4001 13FF	GPIO Port C		Section 5.5.1 on page 96
0X4001 0C00 - 0x4001 0FFF	GPIO Port B		Section 5.5.1 on page 96
0x4001 0800 - 0x4001 0BFF	GPIO Port A		Section 5.5.1 on page 96
0x4001 0400 - 0x4001 07FF	EXTI		Section 6.2 on page 100
0x4001 0000 - 0x4001 03FF	AFIO		Section 5.5 on page 96

Table 1. Register boundary addresses (continued)

Boundary address	Peripheral	Bus	Register map
0x4000 8000 - 0x4000 77FF	Reserved	APB1	
0x4000 7000 - 0x4000 73FF	Power control PWR		Section 3.5 on page 45
0x4000 6C00 - 0x4000 6FFF	Backup registers (BKP)		Section 9.6 on page 137
0x4000 6800 - 0x4000 6BFF	Reserved		
0x4000 6400 - 0x4000 67FF	bxCAN		Section 14.9 on page 314
0x4000 6000 - 0x4000 63FF	shared USB/CAN SRAM 512 bytes		
0x4000 5C00 - 0x4000 5FFF	USB Registers		Section 17.7 on page 413
0x4000 5800 - 0x4000 5BFF	I2C2		Section 15.8 on page 347
0x4000 5400 - 0x4000 57FF	I2C1		Section 15.8 on page 347
0x4000 5000 - 0x4000 4FFF	Reserved		
0x4000 4800 - 0x4000 4BFF	USART3		Section 19.6 on page 469
0x4000 4400 - 0x4000 47FF	USART2		Section 19.6 on page 469
0x4000 4000 - 0x4000 3FFF	Reserved		
0x4000 3800 - 0x4000 3BFF	SPI2		Section 18.5 on page 431
0x4000 3400 - 0x4000 37FF	Reserved		
0x4000 3000 - 0x4000 33FF	Independent watchdog (IWDG)		Section 10.3 on page 143
0x4000 2C00 - 0x4000 2FFF	Window watchdog (WWDG)		
0x4000 2800 - 0x4000 2BFF	RTC		Section 8.5 on page 132
0x4000 2400 - 0x4000 0FFF	Reserved		
0x4000 0800 - 0x4000 0BFF	TIM4 timer		Section 13.6 on page 272
0x4000 0400 - 0x4000 07FF	TIM3 timer		Section 13.6 on page 272
0x4000 0000 - 0x4000 03FF	TIM2 timer		Section 13.6 on page 272

2.3.2 Embedded SRAM

The STM32F10xxx features 20 Kbytes of static SRAM. It can be accessed as bytes, half-words (16 bits) or full words (32 bits). The SRAM start address is 0x2000 0000.

2.3.3 Bit banding

The Cortex™-M3 memory map includes two bit-band regions. These regions map each word in an alias region of memory to a bit in a bit-band region of memory. Writing to a word in the alias region has the same effect as a read-modify-write operation on the targeted bit in the bit-band region.

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

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

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

where:

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

bit_band_base is the starting address of the alias region

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

bit_number is the bit position (0-31) of the targeted bit.

Example:

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

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

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

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

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

2.3.4 Embedded Flash memory

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

- Density of 128 Kbytes
- Memory organization: the Flash memory is organized as a main block and an information block:
 - Main memory block of size 16 Kb × 64 bits. The main block is divided into 128 pages of 1 Kbyte each (see [Table 2](#)).
 - Information block of size 258 × 64 bits. The information block is divided into 2 pages of 2 Kbytes and 16 bytes, respectively (see [Table 2](#)).

The Flash memory interface features:

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

Table 2. Flash module organization

Block	Name	Addresses	Size (bytes)
Main memory	Page 0	0x0800 0000 - 0x0800 03FF	1 Kbyte
	Page 1	0x0800 0400 - 0x0800 07FF	1 Kbyte
	Page 2	0x0800 0800 - 0x0800 0BFF	1 Kbyte
	Page 3	0x0800 0C00 - 0x0800 0FFF	1 Kbyte
	Page 4	0x0800 1000 - 0x0800 13FF	1 Kbyte
	.	.	.
	.	.	.
	.	.	.
Information block	Page 127	0x0801 FC00 - 0x0801 FFFF	1 Kbyte
	System memory	0x1FFF F000 - 0x1FFF F7FF	2 Kbytes
Flash memory registers	Option Bytes	0x1FFF F800 - 0x1FFF F80F	16
	FLASH_ACR	0x4002 2000 - 0x4002 2003	4
	FLASH_KEYR	0x4002 2004 - 0x4002 2007	4
	FLASH_OPTKEYR	0x4002 2008 - 0x4002 200B	4
	FLASH_SR	0x4002 200C - 0x4002 200F	4
	FLASH_CR	0x4002 2010 - 0x4002 2013	4
	FLASH_AR	0x4002 2014 - 0x4002 2017	4
	Reserved	0x4002 2018 - 0x4002 201B	4
	FLASH_OBR	0x4002 201C - 0x4002 201F	4
	FLASH_WRPTR	0x4002 2020 - 0x4002 2023	4
	Reserved	0x4002 2024 - 0x4002 2087	100

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

Reading Flash memory

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

Read accesses can be performed with the following configuration options:

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

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

*zero wait state, if $0 < \text{SYSCLK} \leq 24 \text{ MHz}$
one wait state, if $24 \text{ MHz} < \text{SYSCLK} \leq 48 \text{ MHz}$
two wait states, if $48 \text{ MHz} < \text{SYSCLK} \leq 72 \text{ MHz}$*

- 2 *Half cycle configuration is not available in combination with a prescaler on the AHB. The clock system should be equal to the HCLK clock. This feature can therefore be used only with a direct clock from the internal, 8 MHz RC (HSI) oscillator or with the HSE oscillator.*
- 3 *The prefetch buffer must be kept on when using a prescaler different from 1 on the AHB clock*
- 4 *Using DMA: DMA accesses Flash memory on the DCode bus and has priority over ICode instructions. The DMA provides one free cycle after each transfer. Some instructions can be performed together with DMA transfer.*

Programming and erasing Flash memory

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

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

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

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

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

2.4 Boot configuration

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

Table 3. Boot modes

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

This aliases the physical memory associated with each boot mode to Block 000 (boot memory). The values on the BOOT pins are latched on the 4th rising edge of SYSCLK after a Reset. It is up to the user to set the BOOT1 and BOOT0 pins after Reset to select the required boot mode.

The BOOT pins are also re-sampled when exiting from Standby mode. Consequently they must be kept in the required Boot mode configuration in Standby mode.

Even when aliased in the boot memory space, the related memory (Flash memory or SRAM) is still accessible at its original memory space.

After this startup delay has elapsed, the CPU starts code execution from the boot memory, located at the bottom of the memory address space starting from 0x0000 0000.

Embedded boot loader

The embedded boot loader is used to reprogram the Flash memory using the USART1 serial interface. This program is located in the SystemMemory and is programmed by ST during production.

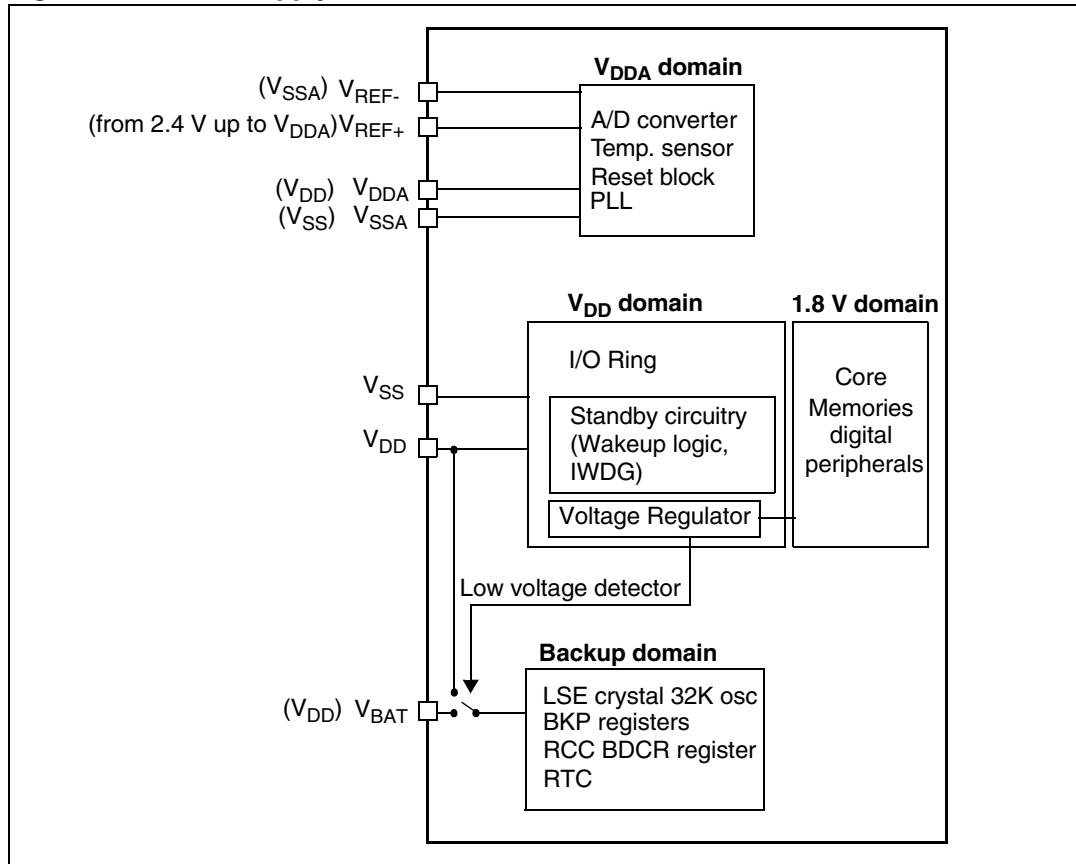
3 Power control (PWR)

3.1 Power supplies

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

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

Figure 3. Power supply overview



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

3.1.1 Independent A/D converter supply and reference voltage

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

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

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

On 100-pin packages

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

On 64-pin packages

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

3.1.2 Battery backup domain

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

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

Warning: During the $t_{RSTTEMPO}$ temporization at V_{DD} startup, the power switch between V_{BAT} and V_{DD} remains connected to V_{BAT} . As V_{DD} rises fast and may become established during this time, a current may be injected into V_{BAT} through a diode connected between V_{DD} and V_{BAT} when V_{BAT} is lower than $V_{DD}-0.6V$. Refer to the datasheet for the value of $t_{RSTTEMPO}$.

If no external battery is used in the application, V_{BAT} must be connected externally to V_{DD} .

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

- PC14 and PC15 can be used as either GPIO or LSE pins
- PC13 can be used as GPIO, TAMPER pin, RTC Calibration Clock, RTC Alarm or second output (refer to [Section 9: Backup registers \(BKP\) on page 133](#))

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

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

- PC14 and PC15 can be used as LSE pins only
- PC13 can be used as TAMPER pin, RTC Alarm or Second output (refer to section [Section 9.5.2: RTC clock calibration register \(BKP_RTCCCR\) on page 134](#)).

3.1.3 Voltage regulator

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

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

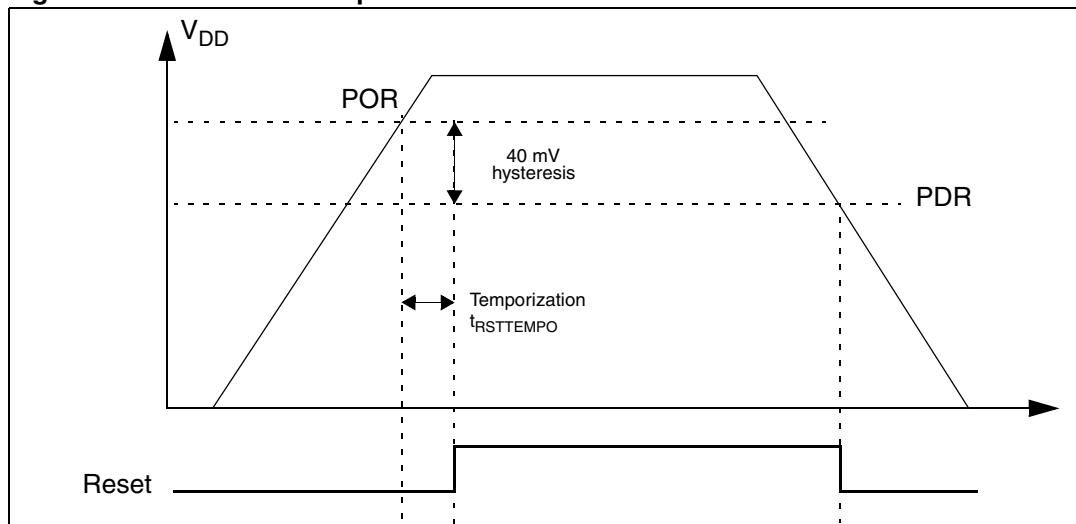
3.2 Power supply supervisor

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

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

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

Figure 4. Power on reset/power down reset waveform



3.2.2 Programmable voltage detector (PVD)

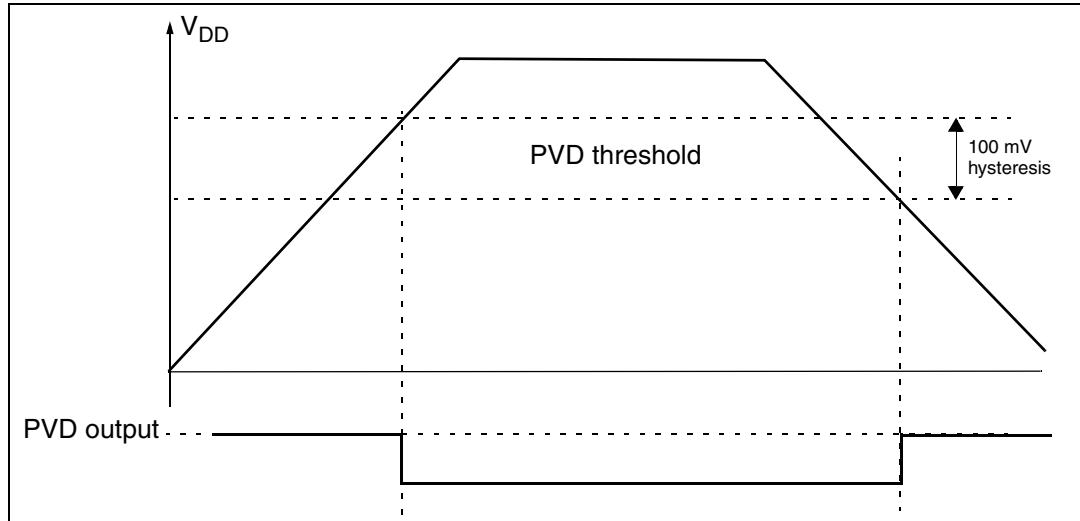
You can use the PVD to monitor the V_{DD} power supply by comparing it to a threshold selected by the PLS[2:0] bits in the [Power control register \(PWR_CR\)](#).

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 5. PVD thresholds



3.3 Low-power modes

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

The STM32F10xxx devices feature three low-power modes:

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

In addition, the power consumption in Run mode can be 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 4. Low-power mode summary

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

3.3.1 Slowing down system clocks

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

For more details refer to [Section 4.3.2: Clock configuration register \(RCC_CFGR\)](#).

3.3.2 Peripheral clock gating

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

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

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

3.3.3 Sleep mode

Entering Sleep mode

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

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

Refer to [Table 5](#) and [Table 6](#) 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. This event can be either an interrupt enabled in the peripheral control register but not in the NVIC, or an EXTI line configured in event mode.

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

Refer to [Table 5](#) and [Table 6](#) for more details on how to exit Sleep mode.

Table 5. Sleep-now

Sleep-now mode	Description
Mode entry	WFI (Wait for Interrupt) or WFE (Wait for Event) while: – SLEEPDEEP = 0 and – SLEEPONEXIT = 0 Refer to the Cortex™-M3 System Control register.
Mode exit	If WFI was used for entry: Interrupt: Refer to Table 27: Vector table If WFE was used for entry Wakeup event: Refer to Section 6.2.3: Wakeup event management
Wakeup latency	None

Table 6. Sleep-on-exit

Sleep-on-exit	Description
Mode entry	WFI (wait for interrupt) while: – SLEEPDEEP = 0 and – SLEEPONEXIT = 1 Refer to the Cortex™-M3 System Control register.
Mode exit	Interrupt: refer to Table 27: Vector table .
Wakeup latency	None

3.3.4 Stop mode

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

Entering Stop mode

Refer to [Table 7](#) for details on how to enter 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 entry is delayed until the memory access is finished.

If an access to APB domain is ongoing, 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 10.1 in Section 10: Independent watchdog \(IWDG\)](#).
- real-time clock (RTC): this is configured by the RTCEN bit in the [Backup domain control register \(RCC_BDCR\)](#)
- Internal RC oscillator (LSI RC): this is configured by the LSION bit in the [Control/status register \(RCC_CSR\)](#).
- External 32.768 kHz oscillator (LSE OSC): this is configured by the LSEON bit in the [Backup domain control register \(RCC_BDCR\)](#).

Exiting Stop mode

Refer to [Table 7](#) for more details on how to exit Stop mode.

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

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

Table 7. 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"> – Set SLEEPDEEP bit in Cortex™-M3 System Control register – Clear PDDS bit in Power Control register (PWR_CR) – Select the voltage regulator mode by configuring LPDS bit in PWR_CR <p>Note: To enter Stop mode, all EXTI Line pending bits (in Pending register (EXTI_PR)) and RTC Alarm flag must be reset. Otherwise, the Stop mode entry procedure is ignored and program execution continues.</p>
Mode exit	<p>If WFI was used for entry:</p> <p>Any EXTI Line configured in Interrupt mode (the corresponding EXTI Interrupt vector must be enabled in the NVIC). Refer to Table 27: Vector table on page 97</p> <p>If WFE was used for entry:</p> <p>Any EXTI Line configured in event mode. Refer to Section 6.2.3: Wakeup event management on page 101</p>
Wakeup latency	HSI RC wakeup time + regulator wakeup time from Low-power mode

3.3.5 Standby mode

The Standby mode allows to achieve the lowest power consumption. It is based on the Cortex-M3 deepsleep mode, with the voltage regulator disabled. The 1.8 V domain is consequently powered off. The PLL, the HSI oscillator and the HSE oscillator are also switched off. SRAM and register contents are lost except for registers in the Backup domain and Standby circuitry (see [Figure 3](#)).

Entering Standby mode

Refer to [Table 8](#) 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 10.1 in Section 10: Independent watchdog \(IWDG\)](#).
- real-time clock (RTC): this is configured by the RTCEN bit in the Backup domain control register (RCC_BDCR)
- Internal RC oscillator (LSI RC): this is configured by the LSION bit in the Control/status register (RCC_CSR).
- External 32.768 kHz oscillator (LSE OSC): this is configured by the LSEON bit in the Backup domain control register (RCC_BDCR)

Exiting Standby mode

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

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

Refer to [Table 8](#) for more details on how to exit Standby mode.

Table 8. Standby mode

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

I/O states in Standby mode

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

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

Debug mode

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

However, by setting some configuration bits in the DBGMCU_CR register, the software can be debugged even when using the low-power modes extensively. For more details, refer to [Section 20.15.1: Debug support for low-power modes](#).

3.3.6

Auto Wakeup (AWU) from low-power mode

The RTC can be used to wakeup the MCU from low-power mode without depending on an external interrupt (Auto Wakeup mode). The RTC provides a programmable time base for waking-up from Stop or Standby mode at regular intervals. For this purpose, two of the three

alternative RTC clock sources can be selected by programming the RTCSEL[1:0] bits in the *Backup domain control register (RCC_BDCR)*:

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

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

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

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

3.4 Power control registers

3.4.1 Power control register (PWR_CR)

Address offset: 0x00

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

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

Bits 31:9 Reserved, always read as 0.

Bit 8 **DBP: Disable Backup Domain write protection.**

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

0: Access to RTC and Backup registers disabled

1: Access to RTC and Backup registers enabled

Bits 7:5 **PLS[2:0]: PVD Level Selection.**

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

000: 2.2V

001: 2.3V

010: 2.4V

011: 2.5V

100: 2.6V

101: 2.7V

110: 2.8V

111: 2.9V

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

Bit 4 **PVDE: Power Voltage Detector Enable.**

This bit is set and cleared by software.

0: PVD disabled

1: PVD enabled

Bit 3 **CSBF: Clear Standby Flag.**

This bit is always read as 0.

0: No effect

1: Clear the SBF Standby Flag (write).

Bit 2 **CWUF: Clear Wakeup Flag.**

This bit is always read as 0.

0: No effect

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

Bit 1 **PDDS: Power Down Deepsleep.**

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

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

1: Enter Standby mode when the CPU enters Deepsleep.

Bit 0 **LPDS: Low-Power Deepsleep.**

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

0: Voltage regulator on during Stop mode

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

3.4.2 Power control/status register (PWR_CSR)

Address offset: 0x04

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

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

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

Bits 31:9 Reserved, always read as 0.

Bit 8 **EWUP: Enable WKUP pin**

This bit is set and cleared by software.

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

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

Note: This bit is reset by a system Reset.

Bits 7:3 Reserved, always read as 0.

Bit 2 **PVDO: PVD Output**

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

0: V_{DD} 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.

Note: The PVD is stopped by Standby mode. For this reason, this bit is equal to 0 after Standby or reset until the PVDE bit is set.

Bit 1 **SBF: Standby Flag**

This bit is set by hardware and cleared only by a POR/PDR (Power On Reset/Power Down Reset) or by setting the CSBF bit in the [Power control register \(PWR_CR\)](#)

0: Device has not been in Standby mode

1: Device has been in Standby mode

Bit 0 **WUF: Wakeup Flag**

This bit is set by hardware and cleared only by a POR/PDR (Power On Reset/Power Down Reset) or by setting the CWUF bit in the *Power control register (PWR_CR)*

0: No wakeup event occurred

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

Note: An additional wakeup event is detected if the WKUP pin is enabled (by setting the EWUP bit) when the WKUP pin level is already high.

3.5 PWR register map

The following table summarizes the PWR registers.

Table 9. 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
000h	PWR_CR																								DBP	PLS[2:0]	PVDE	CSBF	3	CWUF	2	1	0
	Reset value																								0	0 0 0	0	0	0	0	0	0	0
004h	PWR_CSR																								EWUP	Reserved	PVDO	SBF	PDSS	1	WUF	LPDS	0
	Reset value																								0	0	0	0	0	0	0	0	0

Refer to [Table 1 on page 27](#) for the register boundary addresses.

4 Reset and clock control (RCC)

4.1 Reset

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

4.1.1 System Reset

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

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

1. A low level on the NRST pin (External Reset)
2. Window Watchdog end of count condition (WWDG Reset)
3. Independent Watchdog end of count condition (IWDG Reset)
4. A software Reset (SW Reset) (see [Section : Software Reset](#))
5. Low-power management Reset (see [Section : Low-power management Reset](#))

The reset source can be identified by checking the Reset flags in the Control/Status register, RCC_CSR (see [Section 4.3.10: Control/status register \(RCC_CSR\)](#)).

Software Reset

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

Low-power management Reset

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

1. Reset generated when entering Standby mode:

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

2. Reset when entering Stop mode:

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

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

4.1.2 Power reset

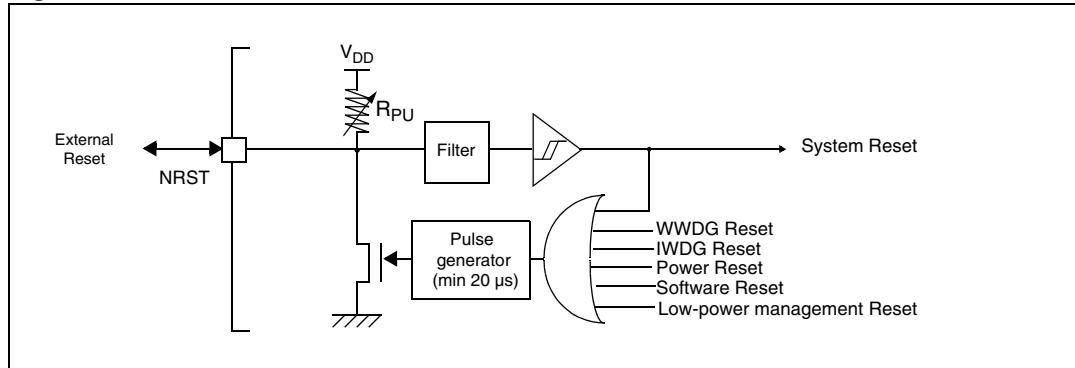
A power Reset is generated when one of the following events occurs:

1. Power On/Power down Reset (POR/PDR Reset)
2. When exiting Standby mode

A power Reset sets all registers to their reset values except the Backup domain (see [Figure 3](#))

These sources act on the NRST pin and it is always kept low during the delay phase. The RESET service routine vector is fixed at address 0x0000_0004 in the memory map. For more details, refer to [Table 27: Vector table on page 97](#).

Figure 6. Reset circuit



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

4.1.3 Backup domain Reset

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

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

4.2 Clocks

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

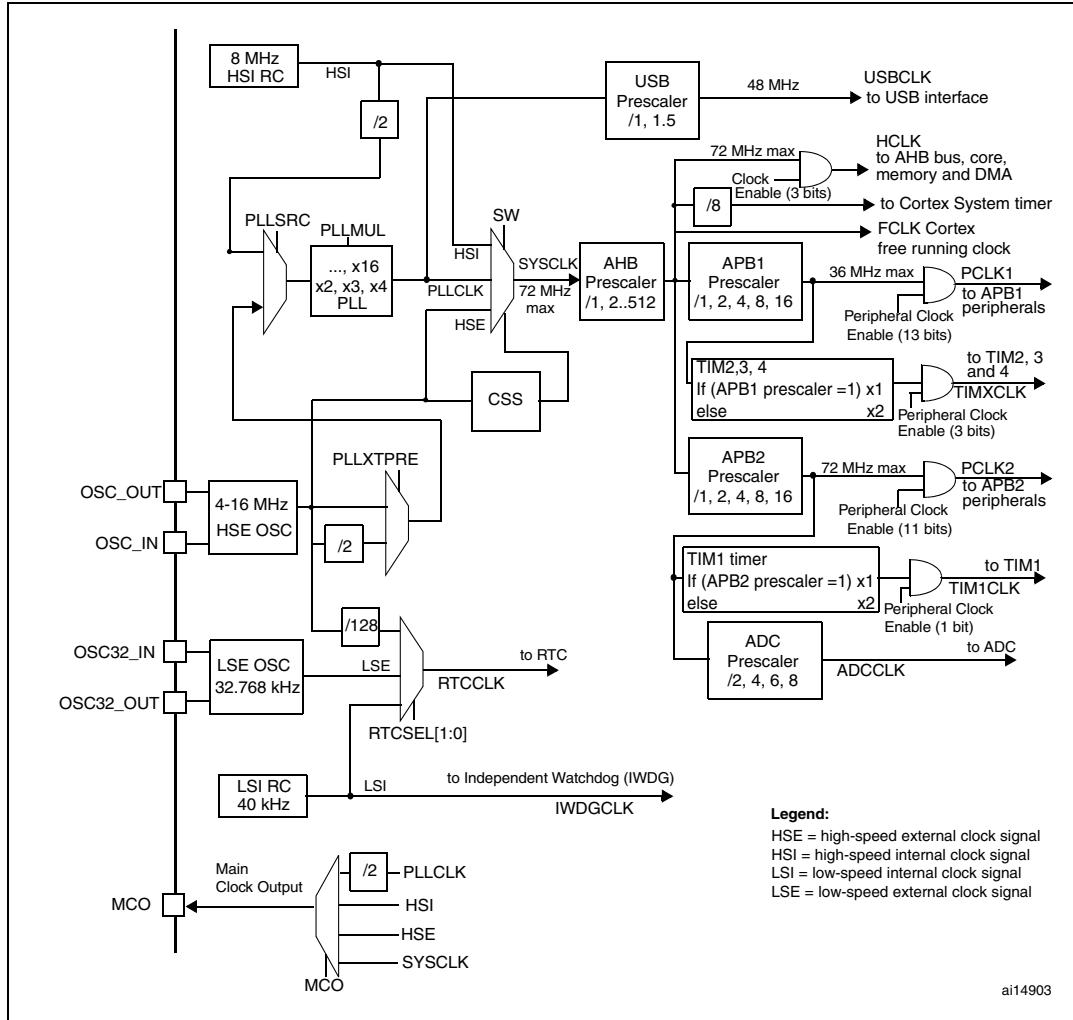
- HSI oscillator clock
- HSE oscillator clock
- PLL clock

The devices have the following two secondary clock sources:

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

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

Figure 7. Clock tree



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

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

The RCC feeds the Cortex System Timer (SysTick) external clock with the AHB clock divided by 8. The SysTick can work either with this clock or with the Cortex clock (AHB), configurable in the SysTick Control and Status Register. The ADCs are clocked by the clock of the High Speed domain (APB2) divided by 2, 4, 6 or 8.

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

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

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

4.2.1 HSE clock

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

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

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

Figure 8. HSE/ LSE clock sources

Hardware configuration	
External Clock	
Crystal/Ceramic Resonators	

External source (HSE bypass)

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

External crystal/ceramic resonator (HSE crystal)

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

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

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

hardware. An interrupt can be generated if enabled in the [*Clock interrupt register \(RCC_CIR\)*](#).

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

4.2.2 HSI clock

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

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

Calibration

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

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 \(RCC_CR\)*](#).

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 4.2.7: Clock security system \(CSS\) on page 52*](#).

4.2.3 PLL

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

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

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

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

4.2.4 LSE clock

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

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

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

External source (LSE bypass)

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

4.2.5 LSI clock

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

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

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

4.2.6 System clock (SYSCLK) selection

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

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

4.2.7 Clock security system (CSS)

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

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

Note:

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

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

4.2.8 RTC clock

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

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

Consequently:

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

4.2.9 Watchdog clock

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

4.2.10 Clock-out capability

The microcontroller clock output (MCO) capability allows the clock to be output onto the external MCO pin. The configuration registers of the corresponding GPIO port must be

programmed in alternate function mode. One of 4 clock signals can be selected as the MCO clock.

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

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

4.3 RCC register description

Refer to [Section 1.1 on page 23](#) for a list of abbreviations used in register descriptions.

4.3.1 Clock control register (RCC_CR)

Address offset: 0x00

Reset value: 0x0000 XX83 where X is undefined.

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				PLL RDY	PL隆ON	Reserved				CSS ON	HSE BYP	HSE RDY	HSE ON		
Res.				r	rw	Res.				rw	rw	r	rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]				Res.	HSI RDY	HSION	
r	r	r	r	r	r	r	r	rw	rw	rw	rw		r	rw	

Bits 31:26 Reserved, always read as 0.

Bit 25 **PLL RDY** *PLL clock ready flag*

Set by hardware to indicate that the PLL is locked.

0: PLL unlocked

1: PLL locked

Bit 24 **PL隆ON** *PLL enable*

Set and reset by software to enable PLL.

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

0: PLL OFF

1: PLL ON

Bits 23:20 Reserved, always read as 0.

Bit 19 **CSS ON** *Clock Security System enable*

Set and reset by software to enable clock detector.

0: Clock detector OFF

1: Clock detector ON if external 1-25 MHz oscillator is ready.

Bit 18 HSEBYP External High Speed clock Bypass

Set and reset by software in debug for bypassing oscillator with external clock. This bit can be written only if the external 1-25 MHz oscillator is disabled.

0: external 1-25 MHz oscillator not bypassed

1: external 1-25 MHz oscillator bypassed with external clock

Bit 17 HSERDY External High Speed clock ready flag

Set by hardware to indicate that external 1-25 MHz oscillator is stable. This bit needs 6 cycles of external 1-25 MHz oscillator clock to fall down after HSEON reset.

0: external 1-25 MHz oscillator not ready

1: external 1-25 MHz oscillator ready

Bit 16 HSEON External High Speed clock enable

Set and reset by software.

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

0: HSE oscillator OFF

1: HSE oscillator ON

Bits 15:8 HSICAL[7:0] Internal High Speed clock Calibration

These bits are initialized automatically at startup.

Bits 7:3 HSITRIM[4:0] Internal High Speed clock trimming

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

The default value is 16, which, when added to the HSICAL value, should trim the HSI to 8 MHz at $T_A = 25^\circ\text{C}$. The HSI RC frequency increases when the HSICAL value increases and decreases when the HSICAL value decreases. The trimming step is 40 kHz between two consecutive HSICAL steps.

Bit 2 Reserved, always read as 0.

Bit 1 HSIRDY Internal High Speed clock ready flag

Set by hardware to indicate that internal 8 MHz RC oscillator is stable. This bit needs 6 cycles of the internal 8 MHz RC oscillator clock to fall down after HSION reset.

0: internal 8 MHz RC oscillator not ready

1: internal 8 MHz RC oscillator ready

Bit 0 HSION Internal High Speed clock enable

Set and reset by software.

Set by hardware to force the internal 8 MHz RC oscillator ON when leaving Stop and Standby mode or in case of failure of the external 1-25 MHz oscillator used directly or indirectly as system clock. This bit can not be reset if the internal 8 MHz RC is used directly or indirectly as system clock or is selected to become the system clock.

0: internal 8 MHz RC oscillator OFF

1: internal 8 MHz RC oscillator ON

4.3.2 Clock configuration register (RCC_CFGR)

Address offset: 0x04

Reset value: 0x0000 0000

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

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				MCO[2:0]			Res.	USB PRE	PLLMUL[3:0]				PLL XTPRE	PLL SRC	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC PRE[1:0]		PPRE2[2:0]			PPRE1[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

Bits 31:26 Reserved, always read as 0.

Bits 26:24 **MCO Microcontroller Clock Output**

Set and reset by software.

0xx: No clock

100: System clock selected

101: Internal 8 MHz RC oscillator clock selected

110: External 1-25 MHz oscillator clock selected

111: PLL clock divided by 2 selected

Notes:

- This clock output could have some truncated cycle at start-up or during MCO clock source switching.
- When the System Clock is selected to output onto MCO, make sure that this clock does not exceed 50 MHz (the maximum I/O speed).

Bit 22 **USBPRE USB prescaler**

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

0: PLL clock is divided by 1.5

1: PLL clock is not divided

Bits 21:18 PLLMUL PLL Multiplication Factor

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

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

0000: PLL input clock x 2

0001: PLL input clock x 3

0010: PLL input clock x 4

0011: PLL input clock x 5

0100: PLL input clock x 6

0101: PLL input clock x 7

0110: PLL input clock x 8

0111: PLL input clock x 9

1000: PLL input clock x 10

1001: PLL input clock x 11

1010: PLL input clock x 12

1011: PLL input clock x 13

1100: PLL input clock x 14

1101: PLL input clock x 15

1110: PLL input clock x 16

1111: PLL input clock x 16

Bit 17 PLLXTPRE HSE divider for PLL entry

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

0: HSE clock not divided

1: HSE clock divided by 2

Bit 16 PLLSRC PLL entry clock source

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

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

1: HSE oscillator clock selected as PLL input clock

Bits 14:14 ADCPRE ADC prescaler

Set and reset by software to select the frequency of the clock to the ADCs.

00: PLCK2 divided by 2

01: PLCK2 divided by 4

10: PLCK2 divided by 6

11: PLCK2 divided by 8

Bits 13:11 PPREG APB High speed prescaler (APB2)

Set and reset by software to control APB High speed clocks division factor.

0xx: HCLK not divided

100: HCLK divided by 2

101: HCLK divided by 4

110: HCLK divided by 8

111: HCLK divided by 16

Bits 10:8 PPREG APB Low speed prescaler (APB1)

Set and reset by software to control APB Low speed clocks division factor.

Warning: the software has to set correctly these bits to not exceed 36 MHz on this domain.

0xx: HCLK not divided

100: HCLK divided by 2

101: HCLK divided by 4

110: HCLK divided by 8

111: HCLK divided by 16

Bits 7:4 HPREG AHB prescaler

Set and reset by software to control AHB clock division factor.

0xxx: SYSCLK not divided

1000: SYSCLK divided by 2

1001: SYSCLK divided by 4

1010: SYSCLK divided by 8

1011: SYSCLK divided by 16

1100: SYSCLK divided by 64

1101: SYSCLK divided by 128

1110: SYSCLK divided by 256

1111: SYSCLK divided by 512

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

Refer to [Reading Flash memory on page 31](#) section for more details.

Bits 3:2 SWS System Clock Switch Status

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

00: HSI oscillator used as system clock

01: HSE oscillator used as system clock

10: PLL used as system clock

11: not applicable

Bits 1:0 SW System clock Switch

Set and reset by software to select SYSCLK source.

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

00: HSI selected as system clock

01: HSE selected as system clock

10: PLL selected as system clock

11: not allowed

4.3.3 Clock interrupt register (RCC_CIR)

Address offset: 0x08

Reset value: 0x0000 0000

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

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

Bits 31:24 Reserved, always read as 0.

Bit 23 **CSSC Clock Security System Interrupt Clear**

Set by software to clear CSSF.

Reset by hardware when clear done.

0: CSSF not cleared

1: CSSF cleared

Bits 22:21 Reserved, always read as 0.

Bit 20 **PLLRDYC PLL Ready Interrupt Clear**

Set by software to clear PLLRDYF.

Reset by hardware when clear done.

0: PLLRDYF not cleared

1: PLLRDYF cleared

Bit 19 **HSERDYC HSE Ready Interrupt Clear**

Set by software to clear HSERDYF.

Reset by hardware when clear done.

0: HSERDYF not cleared

1: HSERDYF cleared

Bit 18 **HSIRDYC HSI Ready Interrupt Clear**

Set by software to clear HSIRDYF.

Reset by hardware when clear done.

0: HSIRDYF not cleared

1: HSIRDYF cleared

Bit 17 **LSERDYC LSE Ready Interrupt Clear**

Set by software to clear LSERDYF.

Reset by hardware when clear done.

0: LSERDYF not cleared

1: LSERDYF cleared

Bit 16 LSIRDYC LSI Ready Interrupt Clear

Set by software to clear LSIRDYF.
 Reset by hardware when clear done.
 0: LSIRDYF not cleared
 1: LSIRDYF cleared

Bits 15:13 Reserved, always read as 0.

Bit 12 PLLRDYIE PLL Ready Interrupt Enable

Set and reset 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 reset by software to enable/disable interrupt caused by the external 1-25 MHz oscillator stabilization.
 0: HSE ready interrupt disabled
 1: HSE ready interrupt enabled

Bit 10 HSIRDYIE HSI Ready Interrupt Enable

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

Bit 9 LSERDYIE LSE Ready Interrupt Enable

Set and reset by software to enable/disable interrupt caused by the external 40 kHz oscillator stabilization.
 0: LSE ready interrupt disabled
 1: LSE ready interrupt enabled

Bit 8 LSIRDYIE LSI Ready Interrupt Enable

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

Bit 7 CSSF Clock Security System Interrupt flag

Reset by software by writing CSSC.
 Set by hardware when a failure is detected in the external 1-25 MHz oscillator.
 0: No clock security interrupt caused by HSE clock failure
 1: Clock security interrupt caused by HSE clock failure

Bits 6:5 Reserved, always read as 0.

Bit 4 PLLRDYF PLL Ready Interrupt flag

Reset by software by writing PLLRDYC.
 Set by hardware when the PLL locks and PLLRDYDIE is set.
 0: No clock ready interrupt caused by PLL lock
 1: Clock ready interrupt caused by PLL lock

Bit3 HSERDYF HSE Ready Interrupt flag

Reset by software by writing HSERDYC.
 Set by hardware when External Low Speed clock becomes stable and HSERDYDIE is set.
 0: No clock ready interrupt caused by the external 1-25 MHz oscillator
 1: Clock ready interrupt caused by the external 1-25 MHz oscillator

Bit 2 HSIRDYF HSI Ready Interrupt flag

Reset by software by writing HSIRDYC.

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

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

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

Bit 1 LSERDYF LSE Ready Interrupt flag

Reset by software by writing LSERDYC.

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

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

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

Bit 0 LSIRDYF LSI Ready Interrupt flag

Reset by software by writing LSIRDYC.

Set by hardware when Internal Low Speed clock becomes stable and LSIRDYDIE is set.

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

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

4.3.4 APB2 Peripheral reset register (RCC_APB2RSTR)

Address offset: 0x0C

Reset value: 0x00000 0000

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

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

Bits 31:15 Reserved, always read as 0.

Bit 14 **USART1RST USART1 reset**

Set and reset by software.

0: No effect

1: Reset USART1

Bit 13 Reserved, always read as 0.

Bit 12 **SPI1RST SPI 1 reset**

Set and reset by software.

0: No effect

1: Reset SPI 1

Bit 11 **TIM1RST TIM1 Timer reset**

Set and reset by software.

0: No effect

1: Reset TIM1 timer

Bit 10 **ADC2RST ADC 2 interface reset**

Set and reset by software.

0: No effect

1: Reset ADC 2 interface

Bit 9 **ADC1RST ADC 1 interface reset**

Set and reset by software.

0: No effect

1: Reset ADC 1 interface

Bits 8:7 Reserved, always read as 0.

Bit 6 **IOPERST IO port E reset**

Set and reset by software.

0: No effect

1: Reset IO port E

Bit 5 IOPDRST *IO port D reset*

Set and reset by software.
0: No effect
1: Reset I/O port D

Bit 4 IOPCRST *IO port C reset*

Set and reset by software.
0: No effect
1: Reset I/O port C

Bit 3 IOPBRST *IO port B reset*

Set and reset by software.
0: No effect
1: Reset I/O port B

Bit 2 IOPARST *I/O port A reset*

Set and reset by software.
0: No effect
1: Reset I/O port A

Bit 1 Reserved, always read as 0.**Bit 0 AFIORST** *Alternate Function I/O reset*

Set and reset by software.
0: No effect
1: Reset Alternate Function

4.3.5 APB1 Peripheral reset register (RCC_APB1RSTR)

Address offset: 0x10

Reset value: 0x0000 0000

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved	PWR RST	BKP RST	Res.	CAN RST	Res.	USB RST	I2C2 RST	I2C1 RST	Reserved	USART 3 RST	USART 2 RST	Res.			
Res.	rw	rw	Res.	rw	Res.	rw	rw	rw	Res.	rw	rw	res.			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SPI2 RST	Reserved	WWD GRST	Reserved								TIM4 RST	TIM3 RST	TIM2 RST	
Res.	rw	Res.	rw	Res.								rw	rw	rw	

Bits 31:29 Reserved, always read as 0.

Bit 28 **PWRRST** Power interface reset

Set and reset by software.

0: No effect

1: Reset power interface

Bit 27 **BKPRST** Backup interface reset

Set and reset by software.

0: No effect

1: Reset backup interface

Bit 26 Reserved, always read as 0.

Bit 25 **CANRST** CAN reset

Set and reset by software.

0: No effect

1: Reset CAN

Bit 24 Reserved, always read as 0.

Bit 23 **USBRST** USB reset

Set and reset by software.

0: No effect

1: Reset USB

Bit 22 **I2C2RST** I2C 2 reset

Set and reset by software.

0: No effect

1: Reset I2C 2

Bit 21 **I2C1RST** I2C 1 reset

Set and reset by software.

0: No effect

1: Reset I2C 1

Bits 20:19 Reserved, always read as 0.

Bit 18 USART3RST *USART 3 reset*

Set and reset by software.

0: No effect

1: Reset USART 3

Bit 17 USART2RST *USART 2 reset*

Set and reset by software.

0: No effect

1: Reset USART 2

Bits 16:15 Reserved, always read as 0.

Bit 14 SPI2RST *SPI 2 reset*

Set and reset by software.

0: No effect

1: Reset SPI 2

Bits 13:12 Reserved, always read as 0.

Bit 11 WWDGRST *Window Watchdog reset*

Set and reset by software.

0: No effect

1: Reset window watchdog

Bits 10:3 Reserved, always read as 0.

Bit 2 TIM4RST *Timer 4 reset*

Set and reset by software.

0: No effect

1: Reset timer 4

Bit 1 TIM3RST *Timer 3 reset*

Set and reset by software.

0: No effect

1: Reset timer 3

Bit 0 TIM2RST *Timer 2 reset*

Set and reset by software.

0: No effect

1: Reset timer 2

4.3.6 AHB Peripheral Clock enable register (RCC_AHBENR)

Address offset: 0x14

Reset value: 0x0000 0014

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										FLITF EN	Res.	SRAM EN	Res.	DMA EN	

Bits 31:5 Reserved, always read as 0.

Bit 4 **FLITFEN** *FLITF clock enable*

Set and reset by software to disable/enable FLITF clock during sleep mode.

0: FLITF clock disabled during Sleep mode

1: FLITF clock enabled during Sleep mode

Bit 3 Reserved, always read as 0.

Bit 2 **SRAMEN** *SRAM interface clock enable*

Set and reset 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 Reserved, always read as 0.

Bit 0 **DMAEN** *DMA clock enable*

Set and reset by software.

0: DMA clock disabled

1: DMA clock enabled

4.3.7 APB2 Peripheral Clock enable register (RCC_APB2ENR)

Address: 0x18

Reset value: 0x0000 0000

Access: word, half-word and byte access

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res;	USART T1 EN	Res;	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	Reserved	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN	
Res.	rw	Res.	rw	rw	rw	rw	Res.	rw	rw	rw	rw	rw	Res.	rw	

Bits 31:15 Reserved, always read as 0.

Bit 14 **USART1EN USART1 clock enable**

Set and reset by software.

0: USART1 clock disabled

1: USART1 clock enabled

Bit 13 Reserved, always read as 0.

Bit 12 **SPI1EN SPI 1 clock enable**

Set and reset by software.

0: SPI 1 clock disabled

1: SPI 1 clock enabled

Bit 11 **TIM1EN TIM1 Timer clock enable**

Set and reset by software.

0: TIM1 timer clock disabled

1: TIM1 timer clock enabled

Bit 10 **ADC2EN ADC 2 interface clock enable**

Set and reset by software.

0: ADC 2 interface clock disabled

1: ADC 2 interface clock enabled

Bit 9 **ADC1EN ADC 1 interface clock enable**

Set and reset by software.

0: ADC 1 interface disabled

1: ADC 1 interface clock enabled

Bits 8:7 Reserved, always read as 0.

Bit 6 IOPEEN I/O port E clock enable

Set and reset by software.

0: I/O port E clock disabled

1: I/O port E clock enabled

Bit 5 IOPDEN I/O port D clock enable

Set and reset by software.

0: I/O port D clock disabled

1: I/O port D clock enabled

Bit 4 IOPCEN I/O port C clock enable

Set and reset by software.

0: I/O port C clock disabled

1:I/O port C clock enabled

Bit 3 IOPBEN I/O port B clock enable

Set and reset by software.

0: I/O port B clock disabled

1:I/O port B clock enabled

Bit 2 IOPAEN I/O port A clock enable

Set and reset by software.

0: I/O port A clock disabled

1:I/O port A clock enabled

Bit 1 Reserved, always read as 0.**Bit 0 AFIOEN Alternate Function I/O clock enable**

Set and reset by software.

0: Alternate Function I/O clock disabled

1:Alternate Function I/O clock enabled

4.3.8 APB1 Peripheral Clock enable register (RCC_APB1ENR)

Address: 0x1C

Reset value: 0x0000 0000

Access: word, half-word and byte access

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved	PWR EN	BKP EN	Res.	CAN EN	Res.	USB EN	I2C2 EN	I2C1 EN	Reserved	USART3 EN	USART2 EN	Res.			
Res.	rw	rw		rw		rw	rw	rw		rw	rw				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SPI2 EN	Reserved	WWD GEN										TIM4 EN	TIM3 EN	TIM2 EN
Res.	rw	Res.	rw										rw	rw	rw

Bits 31:29 Reserved, always read as 0.

Bit 28 **PWREN** Power interface clock enable

Set and reset by software.

0: Power interface clock disabled

1: Power interface clock enable

Bit 27 **BKPN** Backup interface clock enable

Set and reset by software.

0: Backup interface clock disabled

1: Backup interface clock enabled

Bit 26 Reserved, always read as 0.

Bit 25 **CANEN** CAN clock enable

Set and reset by software.

0: CAN clock disabled

1: CAN clock enabled

Bit 24 Reserved, always read as 0.

Bit 23 **USBEN** USB clock enable

Set and reset by software.

0: USB clock disabled

1: USB clock enabled

Bit 22 **I2C2EN** I2C 2 clock enable

Set and reset by software.

0: I2C 2 clock disabled

1: I2C 2 clock enabled

Bit 21 **I2C1EN** I2C 1 clock enable

Set and reset by software.

0: I2C 1 clock disabled

1: I2C 1 clock enabled

Bits 20:19 Reserved, always read as 0.

Bit 18 **USART3EN** *USART 3 clock enable*

Set and reset by software.
0: USART 3 clock disabled
1: USART 3 clock enabled

Bit 17 **USART2EN** *USART 2 clock enable*

Set and reset by software.
0: USART 2 clock disabled
1: USART 2 clock enabled

Bits 16:15 Reserved, always read as 0.

Bit 14 **SPI2EN** *SPI 2 clock enable*

Set and reset by software.
0: SPI 2 clock disabled
1: SPI 2 clock enabled

Bits 13:12 Reserved, always read as 0.

Bit 11 **WWDGGEN** *Window Watchdog clock enable*

Set and reset by software.
0: Window watchdog clock disabled
1: Window watchdog clock enabled

Bits 10:3 Reserved, always read as 0.

Bit 2 **TIM4EN** *Timer 4 clock enable*

Set and reset by software.
0: Timer 4 clock disabled
1: Timer 4 clock enabled

Bit 1 **TIM3EN** *Timer 3 clock enable*

Set and reset by software.
0: Timer 3 clock disabled
1: Timer 3 clock enabled

Bit 0 **TIM2EN** *Timer 2 clock enable*

Set and reset by software.
0: Timer 2 clock disabled
1: Timer 2 clock enabled

4.3.9 Backup domain control register (RCC_BDCR)

Address: 0x20

Reset value: 0x0000 0000, reset by Backup domain Reset.

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

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

Note: LSEON, LSEBYP, RTCSEL and RTCEN bits of the *Backup domain control register (RCC_BDCR)* are in the Backup domain. As a result, after Reset, these bits are write protected and the DBP bit in the *Power control register (PWR_CR)* has to be set before these can be modified. Refer to [Section 9.1 on page 133](#) for further information. These bits are only reset after a Backup domain Reset and V_{BAT} power on. Any internal or external Reset will not have any effect on these bits.

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

Bits 31:17 Reserved, always read as 0.

Bit 16 **BDRST** Backup domain software reset

Set and reset by software.

0: Reset not activated

1: Resets the entire Backup domain

Bit 15 **RTCEN** RTC clock enable

Set and reset by software.

0: RTC clock disabled

1: RTC clock enabled

Bits 14:10 Reserved, always read as 0.

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

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

00: No clock

01: LSE oscillator clock used as RTC clock

10: LSI oscillator clock used as RTC clock

11: HSE oscillator clock divided by 128 used as RTC clock

Bits 7:3 Reserved, always read as 0.

Bit 2 **LSEBYP** External Low Speed oscillator Bypass

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

0: LSE oscillator not bypassed

1: LSE oscillator bypassed

Bit 1 **LSERDY External Low Speed oscillator Ready**

Set and reset by hardware to indicate when the external 32 kHz oscillator is stable. This bit needs 6 cycles of external Low Speed oscillator clock to fall down after LSEON reset.

- 0: External 32 kHz oscillator not ready
- 1: External 32 kHz oscillator ready

Bit 0 **LSEON External Low Speed oscillator enable**

Set and reset by software.

- 0: External 32 kHz oscillator OFF
- 1: External 32 kHz oscillator ON

4.3.10 Control/status register (RCC_CSR)

Address: 0x24

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

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

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

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

Bit 31 **LPWRRSTF Low-Power reset flag**

Reset by software by writing the RMVF bit.

Set by hardware when a Low-power management reset occurs.

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

For further information on Low-power management reset, refer to [Section : Low-power management Reset](#).

Bit 30 **WWDGRSTF Window watchdog reset flag**

Reset by software by writing the RMVF bit.

Set by hardware when a window watchdog reset occurs.

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

Bit 29 **IWDGRSTF Independent Watchdog reset flag**

Reset by software by writing the RMVF bit.

Set by hardware when a watchdog reset from V_{DD} domain occurs.

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

Bit 28 SFTRSTF Software Reset flag

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

Bit 27 PORRSTF POR/PDR reset flag

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

Bit 26 PINRSTF PIN reset flag

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

Bit 25 Reserved, always read as 0.

Bit 24 RMVF Remove reset flag

Set and reset by software to reset the value of the reset flags.
0: Reset of the reset flags not activated
1: Reset the value of the reset flags

Bits 23:2 Reserved, always read as 0.

Bit 1 LSIRDY Internal Low Speed oscillator Ready

Set and reset by hardware to indicate when the internal RC 40 kHz oscillator is stable. This bit needs 3 cycles of internal RC 40 kHz oscillator to fall down after LSION reset.
0: Internal RC 40 kHz oscillator not ready
1: Internal RC 40 kHz oscillator ready

Bit 0 LSION Internal Low Speed oscillator enable

Set and reset by software.
0: Internal RC 40 kHz oscillator OFF
1: Internal RC 40 kHz oscillator ON

4.4 RCC register map

Table 10. RCC - register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	HSICAL[7:0]	HSITRIM[4:0]	SWS [1:0]	SW [1:0]						
000h	RCC_CR	Reserved				PLL RDY 0 0		PLL ON 0 0		Reserved				HSICAL[7:0]									
		Reset value																					
004h	RCC_CFGR	Reserved		MCO [2:0]		Reserved		PLL MUL[3:0]		Reserved		USBPRE 0 0		HSICAL[7:0]									
		Reset value		0 0 0								PLL MUL[3:0]		HSICAL[7:0]									
008h	RCC_CIR	Reserved				CSSC 0 0		PLL RDYC 0 0		Reserved		USBPRE 0 0		HSICAL[7:0]									
		Reset value										PLL RDYC 0 0		HSICAL[7:0]									
00Ch	RCC_APB2RSTR	Reserved				Reserved				HSICAL[7:0]				HSICAL[7:0]									
		Reset value																					
010h	RCC_APB1RSTR	Reserved		PWRSTF 0 0		BKRST 0 0		Reserved		CANRST 0 0		USBRST 0 0		HSICAL[7:0]									
		Reset value										PLL RDYC 0 0		HSICAL[7:0]									
014h	RCC_AHBENR	Reserved				Reserved				HSICAL[7:0]				HSICAL[7:0]									
		Reset value																					
018h	RCC_APB2ENR	Reserved				Reserved				HSICAL[7:0]				HSICAL[7:0]									
		Reset value																					
01Ch	RCC_APB1ENR	Reserved		PWREN 0 0		BKPN 0 0		Reserved		CANEN 0 0		USBEN 0 0		HSICAL[7:0]									
		Reset value										PLL RDYC 0 0											
020h	RCC_BDCR	Reserved				Reserved				HSICAL[7:0]				HSICAL[7:0]									
		Reset value																					
024h	RCC_CSR	LPWRSTF 0 0		WWDRSTF 0 0		IWDRSTF 0 0		SFTRSTF 0 0		PORRSTF 1 1		PINRSTF 1 1		HSICAL[7:0]									
		Reset value										PLL RDYC 0 0											

Refer to [Table 1 on page 27](#) for the register boundary addresses.

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

5.1 GPIO functional description

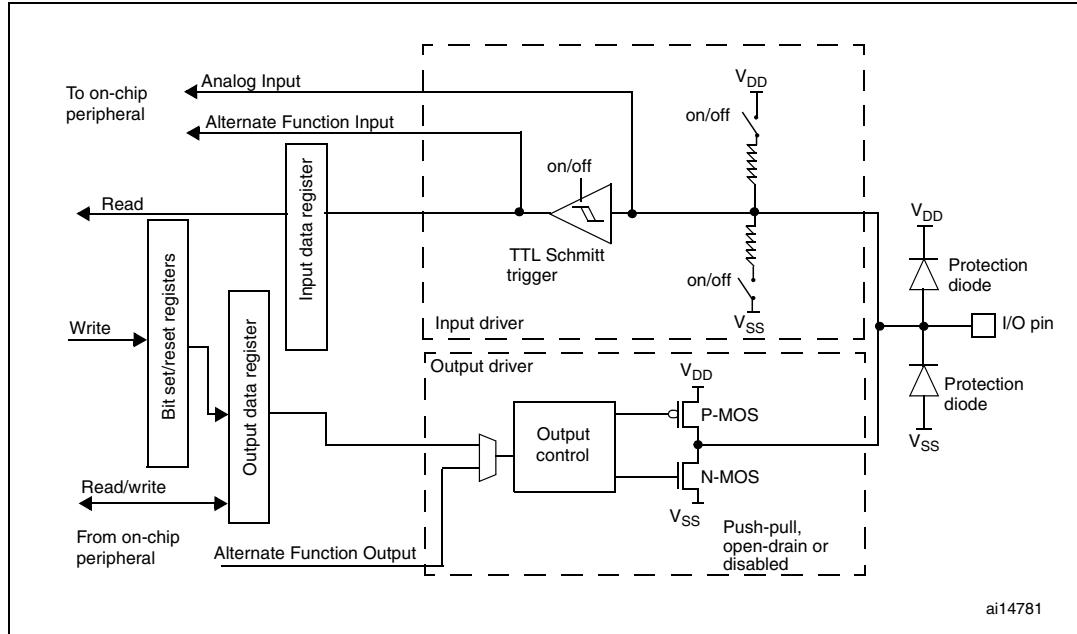
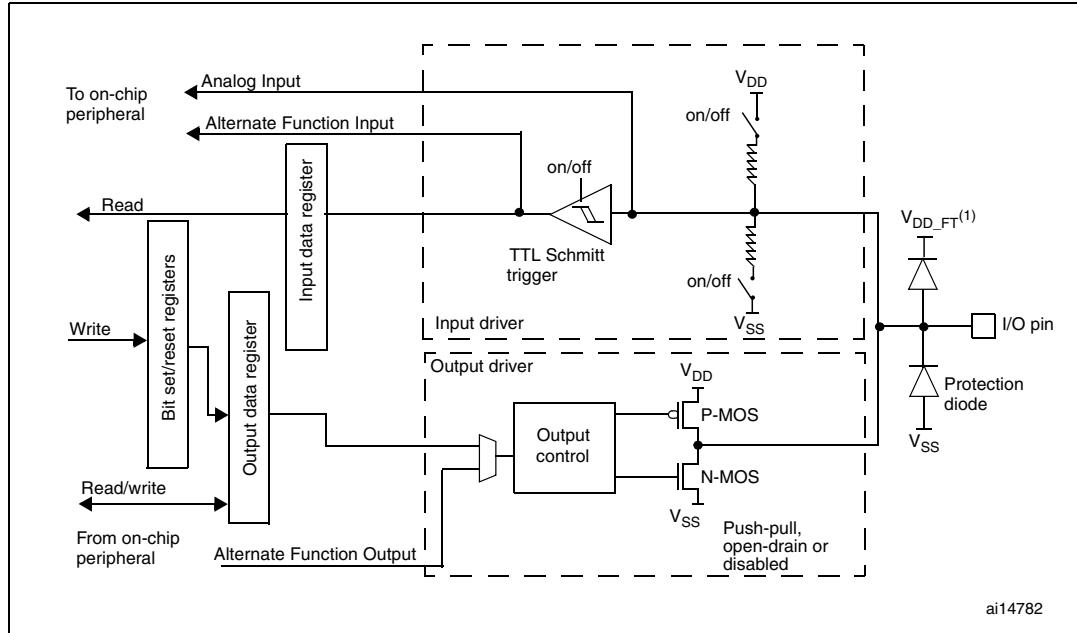
Each of the general-purpose I/O ports has two 32-bit configuration registers (GPIOx_CRL, GPIOx_CRH), two 32-bit data registers (GPIOx_IDR, GPIOx_ODR), a 32-bit set/reset register (GPIOx_BSRR), a 16-bit reset register (GPIOx_BRR) and a 32-bit locking register (GPIOx_LCKR).

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

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

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

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

Figure 9. Basic structure of a standard I/O port bit**Figure 10. Basic structure of a five-volt tolerant I/O port bit**

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

Table 11. Port bit configuration table

Configuration mode		CNF1	CNF0	MODE1	MODE0	PxODR Register
General purpose output	Push-pull	0	0	see <i>Table 12</i>	01 10 11	0 or 1
	Open-drain		1			0 or 1
Alternate Function output	Push-pull	1	0			don't care
	Open-drain		1			don't care
Input	Analog input	0	0	00	00	don't care
	Input floating		1			don't care
	Input pull-down	1	0			0
	Input pull-up					1

Table 12. Output MODE bits

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

5.1.1 General-purpose I/O (GPIO)

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

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

- PA15: JTDI in PU
- PA14: JTCK in PD
- PA13: JTMS in PU
- PB4: JNTRST in PU

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

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

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

5.1.2 Atomic bit set or bit reset

There is no need for the software to disable interrupts when programming the GPIOx_ODR at bit level: it is possible to modify only one or several bits in a single atomic APB2 write access.

This is achieved by programming to ‘1’ the Bit Set/Reset Register (GPIOx_BSRR, or for reset only GPIOx_BRR) to select the bits you want to modify. The unselected bits will not be modified.

5.1.3 External interrupt/wakeup lines

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

- [Section 6.2: External interrupt/event controller \(EXTI\) on page 100](#) and
- [Section 6.2.3: Wakeup event management on page 101](#).

5.1.4 Alternate functions (AF)

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

- For alternate function inputs, the port can be configured either:
 - in Input mode (floating, pull-up or pull-down)
 - in Alternate Function Output mode. In this case the input driver is configured in input floating mode
- For Alternate Function Outputs, the port must be configured in Alternate Function Output mode (Push-Pull or Open-Drain).
- For bidirectional Alternate Functions, the port bit must be configured in Alternate Function Output mode (Push-Pull or Open-Drain). In this case the input driver is configured in input floating mode

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

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

5.1.5 Software remapping of I/O alternate functions

To optimize the number of peripheral I/O functions for different device packages, it is possible to remap some alternate functions to some other pins. This is achieved by software, by programming the corresponding registers (refer to [AFIO register description on page 91](#)). In that case, the alternate functions are no longer mapped to their original assignations.

5.1.6 GPIO locking mechanism

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

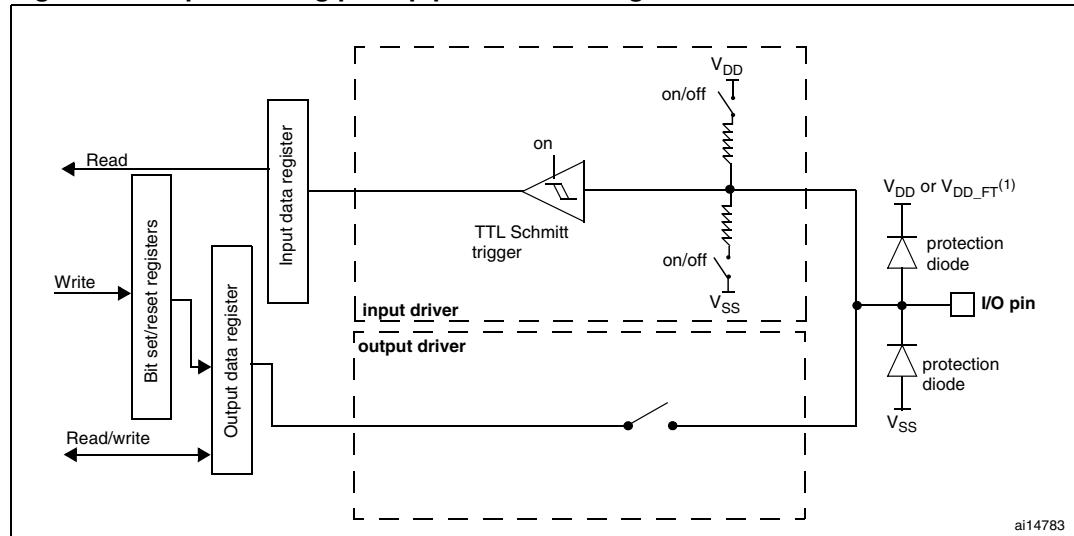
5.1.7 Input configuration

When the I/O Port is programmed as Input:

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

The [Figure 11 on page 78](#) shows the Input Configuration of the I/O Port bit.

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



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

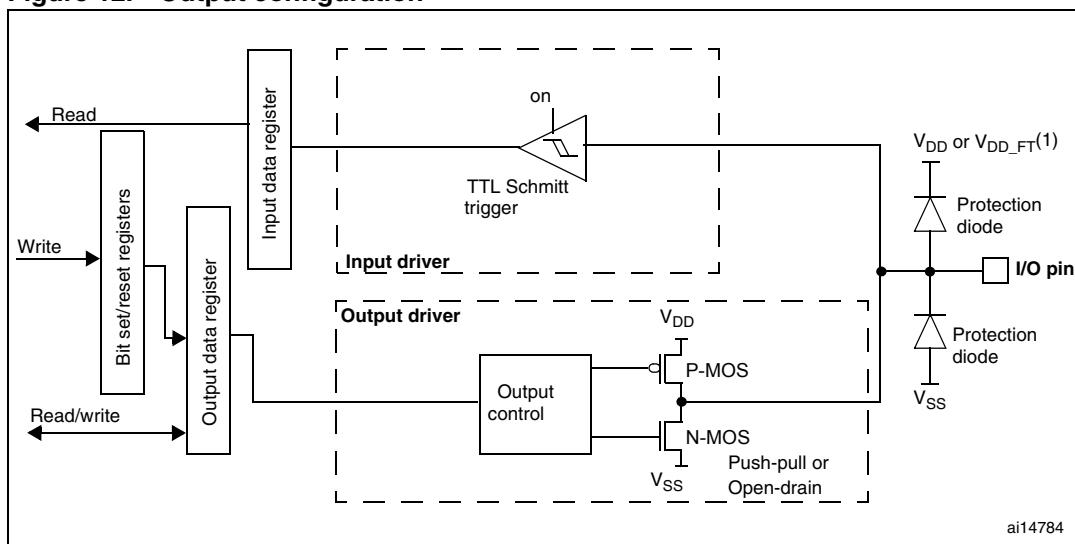
5.1.8 Output configuration

When the I/O Port is programmed as Output:

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

The [Figure 12 on page 79](#) shows the Output configuration of the I/O Port bit.

Figure 12. Output configuration



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

5.1.9 Alternate function configuration

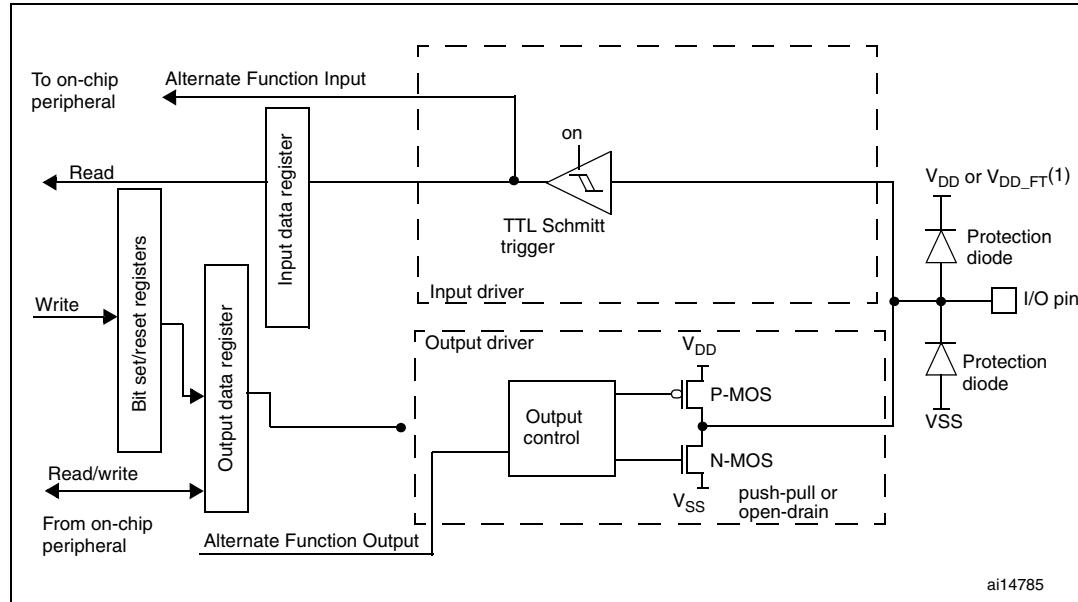
When the I/O Port is programmed as Alternate Function:

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

The [Figure 13 on page 80](#) shows the Alternate Function Configuration of the I/O Port bit. Also, refer to [Section 5.4: AFIO register description on page 91](#) for further information.

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

Figure 13. Alternate function configuration



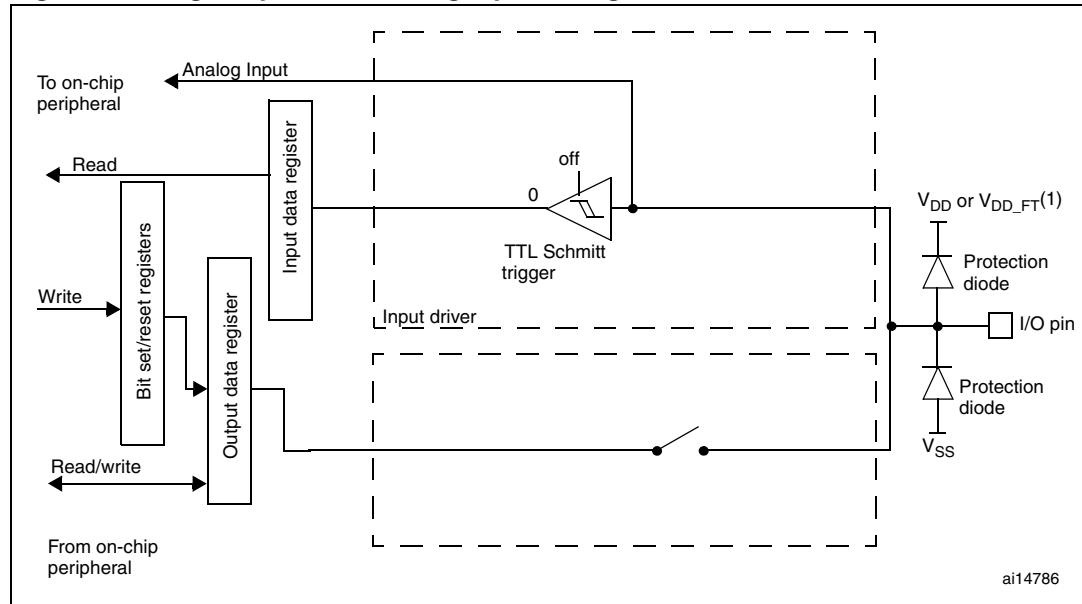
1. V_{DD_FT} is a potential specific to five-volt tolerant I/Os and different from V_{DD} .

5.1.10 Analog input configuration

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

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

The [Figure 14 on page 81](#) shows the High impedance-Analog Input Configuration of the I/O Port bit.

Figure 14. High impedance-analog input configuration

5.2 GPIO register description

Refer to [Section 1.1 on page 23](#) for a list of abbreviations used in register descriptions.

5.2.1 Port configuration register low (GPIOx_CRL) (x=A..E)

Address offset: 0x00

Reset value: 0x4444 4444

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

Bits 31:30, 27:26, 23:22, **CNFy[1:0]: Port x configuration bits (y= 0 .. 7)**

19:18, 15:14, 11:10, 7:6, 3:2 These bits are written by software to configure the corresponding I/O port.

Refer to [Table 11: Port bit configuration table on page 76](#).

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

00: Analog input mode

01: Floating input (reset state)

10: Input with pull-up / pull-down

11: Reserved

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

00: General purpose output push-pull

01: General purpose output Open-drain

10: Alternate function output Push-pull

11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, **MODEy[1:0]: Port x mode bits (y= 0 .. 7)**

17:16, 13:12, 9:8, 5:4, 1:0 These bits are written by software to configure the corresponding I/O port.

Refer to [Table 11: Port bit configuration table on page 76](#).

00: Input mode (reset state)

01: Output mode, max speed 10 MHz.

10: Output mode, max speed 2 MHz.

11: Output mode, max speed 50 MHz.

5.2.2 Port configuration register high (GPIOx_CRH) (x=A..E)

Address offset: 0x04

Reset value: 0x4444 4444

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

Bits 31:30, 27:26, 23:22, **CNFy[1:0]: Port x configuration bits (y= 8 .. 15)**

19:18, 15:14, 11:10, 7:6, 3:2 These bits are written by software to configure the corresponding I/O port.

Refer to [Table 11: Port bit configuration table on page 76](#).

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

- 00: Analog input mode
- 01: Floating input (reset state)
- 10: Input with pull-up / pull-down
- 11: Reserved

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

- 00: General purpose output push-pull
- 01: General purpose output Open-drain
- 10: Alternate function output Push-pull
- 11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, **MODEy[1:0]: Port x mode bits (y= 8 .. 15)**

17:16, 13:12, 9:8, 5:4, 1:0 These bits are written by software to configure the corresponding I/O port.

Refer to [Table 11: Port bit configuration table on page 76](#).

- 00: Input mode (reset state)
- 01: Output mode, max speed 10 MHz.
- 10: Output mode, max speed 2 MHz.
- 11: Output mode, max speed 50 MHz.

5.2.3 Port input data register (GPIO_x_IDR) (x=A..E)

Address offset: 0x08h

Reset value: 0x0000 0000

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

Bits 31:16 Reserved, always read as 0.

Bits 31:0 **IDR_y[15:0]: Port input data (y= 0 .. 15)**

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

5.2.4 Port output data register (GPIO_x_ODR) (x=A..E)

Address offset: 0x0C

Reset value: 0x0000 0000

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

rw rw

Bits 31:16 Reserved, always read as 0.

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

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

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

5.2.5 Port bit set/reset register (GPIOx_BSRR) (x=A..E)

Address offset: 0x10

Reset value: 0x0000 0000

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

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

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

0: No action on the corresponding ODRx bit

1: Reset the corresponding ODRx bit

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

Bits 15:0 **BSy**: Port x Set bit y ($y=0 \dots 15$)

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

0: No action on the corresponding ODRx bit

1: Set the corresponding ODRx bit

5.2.6 Port bit reset register (GPIOx_BRR) (x=A..E)

Address offset: 0x14

Reset value: 0x0000 0000

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

Bits 31:16 Reserved

Bits 15:0 **BRy**: Port x Reset bit y ($y=0 \dots 15$)

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

0: No action on the corresponding ODRx bit

1: Reset the corresponding ODRx bit

5.2.7 Port configuration lock register (GPIOx_LCKR) (x=A..E)

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

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

Address offset: 0x18

Reset value: 0x0000 0000

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

Bits 31:17 Reserved

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

This bit can be read anytime. It can only be modified using the Lock Key Writing Sequence.

0: Port configuration lock key not active

1: Port configuration lock key active. GPIOx_LCKR register is locked until an MCU reset occurs.

LOCK Key Writing Sequence:

Write 1

Write 0

Write 1

Read 0

Read 1 (this read is optional but confirms that the lock is active)

Notes:

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

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

These bits are read write but can only be written when the LCKK bit is 0.

0: Port configuration not locked

1: Port configuration locked.

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

To optimize the number of peripherals available for the 64-pin or the 100-pin package, it is possible to remap some alternate functions to some other pins. This is achieved by software, by programming the *AF remap and debug I/O configuration register (AFIO_MAPR)* on page 92. In this case, the alternate functions are no longer mapped to their original assignments.

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

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

- Note:*
- 1 *The PC14/PC15 GPIO functionality is lost when the 1.8 V domain is powered off (by entering standby mode) or when the backup domain is supplied by V_{BAT} (V_{DD} no more supplied). In this case the IOs are set in analog input mode.*
 - 2 *Refer to the note on IO usage restrictions in Section 3.1.2 on page 34.*

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

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

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

- Note:*
- The use of PD0 and PD1 in output mode is limited since PD0 and PD1 can only be used in output mode at 50 MHz.*

5.3.3 BXCAN alternate function remapping

The BXCAN signal can be mapped on Port A, Port B or Port D as shown in *Table 13*.

Table 13. BXCAN alternate function remapping

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

1. Remap not available on 36-pin package

5.3.4 JTAG/SWD alternate function remapping

The debug interface signals are mapped on the GPIO ports as shown in *Table 14*.

Table 14. Debug interface signals

Alternate function	GPIO port
JTMS / SWDIO	PA13
JTCK / SWCLK	PA14

Table 14. Debug interface signals (continued)

Alternate function	GPIO port
JTDI	PA15
JTDO / TRACESWO	PB3
JNTRST	PB4
TRACECK	PE2
TRACED0	PE3
TRACED1	PE4
TRACED2	PE5
TRACED3	PE6

To optimize the number of free GPIOs during debugging, this mapping can be configured in different ways by programming the SWJ_CFG[1:0] bits in the *AF remap and debug I/O configuration register (AFIO_MAPR)*. Refer to [Table 15](#)

Table 15. Debug port mapping

SWJ_CFG [2:0]	Available debug ports	SWJ I/O pin assigned				
		PA.13 / JTMS/SWDIO	PA.14 / JTCK/S WCLK	PA.15 / JTDI	PB.3 / JTDO/TRACE SWO	PB.4/ JNTRST
000	Full SWJ (JTAG-DP + SW-DP) (Reset state)	X	X	X	X	X
001	Full SWJ (JTAG-DP + SW-DP) but without JNTRST	X	X	X	x	free
010	JTAG-DP Disabled and SW-DP Enabled	X	X	free	free ⁽¹⁾	free
100	JTAG-DP Disabled and SW-DP Disabled	free	free	free	free	free
Other	Forbidden					

1. Released only if not using asynchronous trace.

5.3.5 Timer alternate function remapping

Timer 4 channels 1 to 4 can be remapped from Port B to Port D.

Other timer remapping possibilities are listed in [Table 17](#) to [Table 19](#).

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

Table 16. Timer 4 alternate function remapping

Alternate function	TIM4_REMAP = 0	TIM4_REMAP = 1 ⁽¹⁾
TIM4_CH1	PB6	PD12
TIM4_CH2	PB7	PD13

Table 16. Timer 4 alternate function remapping

Alternate function	TIM4_REMAP = 0	TIM4_REMAP = 1 ⁽¹⁾
TIM4_CH3	PB8	PD14
TIM4_CH4	PB9	PD15

1. Remap available only for 100-pin package.

Table 17. Timer 3 alternate function remapping

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

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

Table 18. Timer 2 alternate function remapping

Alternate function	TIM2_REMAP[1:0] = "00" (no remap)	TIM2_REMAP[1:0] = "01" (partial remap)	TIM2_REMAP[1:0] = "10" (partial remap) ⁽¹⁾	TIM2_REMAP[1:0] = "11" (full remap) ⁽¹⁾
TIM2_CH1/ETR	PA0	PA15	PA0	PA15
TIM2_CH2	PA1	PB3	PA1	PB3
TIM2_CH3	PA2		PB10	
TIM2_CH4	PA3		PB11	

1. Remap not available on 36-pin package.

Table 19. Timer 1 alternate function remapping

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

1. Remap available only for 100-pin packages.

2. Remap not available on 36-pin package.

5.3.6 USART Alternate function remapping

Refer to [AF remap and debug I/O configuration register \(AFIO_MAPR\)](#)

Table 20. USART3 remapping

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

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

2. Remap available only for 100-pin packages.

Table 21. USART2 remapping

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

1. Remap available only for 100-pin packages.

Table 22. USART1 remapping

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

5.3.7 I2C 1 alternate function remapping

Refer to [AF remap and debug I/O configuration register \(AFIO_MAPR\)](#)

Table 23. I2C1 remapping

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

1. Remap not available on 36-pin package.

5.3.8 SPI 1 alternate function remapping

Refer to [AF remap and debug I/O configuration register \(AFIO_MAPR\)](#)

Table 24. SPI1 remapping

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

5.4 AFIO register description

Refer to [Section 1.1 on page 23](#) for a list of abbreviations used in register descriptions.

5.4.1 Event control register (AFIO_EVCR)

Address offset: 0x00

Reset value: 0x0000 0000

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

Bits 31:8 Reserved

Bit 7 **EVOE** *Event Output Enable*

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

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

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

000: PA selected

001: PB selected

010: PC selected

011: PD selected

100: PE selected

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

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

0000: Px0 selected

0001: Px1 selected

0010: Px2 selected

0011: Px3 selected

...

1111: Px15 selected

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

Address offset: 0x04

Reset value: 0x0000 0000

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

Bits 31:27 Reserved

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

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

000: Full SWJ (JTAG-DP + SW-DP): Reset State

001: Full SWJ (JTAG-DP + SW-DP) but without JNTRST

010: JTAG-DP Disabled and SW-DP Enabled

100: JTAG-DP Disabled and SW-DP Disabled

Other combinations: no effect

Bits 23:16 Reserved

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

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

0: No remapping of PD0 and PD1

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

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

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

00: CANRX mapped to PA11, CANTX mapped to PA12

01: Not used

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

11: CANRX mapped to PD0, CANTX mapped to PD1

Bit 12 **TIM4_REMAP** *TIM4 remapping*

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

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

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

Note: TIM4_ETR on PE0 is not re-mapped.

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

These bits are set and cleared by software. They control the mapping of TIM3 channels 1 to 4 on the GPIO ports.

00: No remap (CH1/PA6, CH2/PA7, CH3/PB0, CH4/PB1)

01: Not used

10: Partial remap (CH1/PB4, CH2/PB5, CH3/PB0, CH4/PB1)

11: Full remap (CH1/PC6, CH2/PC7, CH3/PC8, CH4/PC9)

Note: TIM3_ETR on PE0 is not re-mapped.

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

These bits are set and cleared by software. They control the mapping of TIM2 channels 1 to 4 and external trigger (ETR) on the GPIO ports.

00: No remap (CH1/ETR/PA0, CH2/PA1, CH3/PA2, CH4/PA3)

01: Partial remap (CH1/ETR/PA15, CH2/PB3, CH3/PA2, CH4/PA3)

10: Partial remap (CH1/ETR/PA0, CH2/PA1, CH3/PB10, CH4/PB11)

11: Full remap (CH1/ETR/PA15, CH2/PB3, CH3/PB10, CH4/PB11)

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

These bits are set and cleared by software. They control the mapping of TIM2 channels 1 to 4, 1N to 3N, external trigger (ETR) and Break input (BKIN) on the GPIO ports.

00: No remap (ETR/PA12, CH1/PA8, CH2/PA9, CH3/PA10, CH4/PA11, BKIN/PB12, CH1N/PB13, CH2N/PB14, CH3N/PB15)

01: Partial remap (ETR/PA12, CH1/PA8, CH2/PA9, CH3/PA10, CH4/PA11, BKIN/PA6, CH1N/PA7, CH2N/PB0, CH3N/PB1)

10: not used

11: Full remap (ETR/PE7, CH1/PE9, CH2/PE11, CH3/PE13, CH4/PE14, BKIN/PE15, CH1N/PE8, CH2N/PE10, CH3N/PE12)

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

These bits are set and cleared by software. They control the mapping of USART3 CTS, RTS,CK,TX and RX alternate functions on the GPIO ports.

00: No remap (TX/PB10, RX/PB11, CK/PB12, CTS/PB13, RTS/PB14)

01: Partial remap (TX/PC10, RX/PC11, CK/PC12, CTS/PB13, RTS/PB14)

10: not used

11: Full remap (TX/PD8, RX/PD9, CK/PD10, CTS/PD11, RTS/PD12)

Bit 3 **USART2_REMAP** *USART2 remapping*

This bit is set and cleared by software. It controls the mapping of USART2 CTS, RTS,CK,TX and RX alternate functions on the GPIO ports.

0: No remap (CTS/PA0, RTS/PA1, TX/PA2, RX/PA3, CK/PA4)

1: Remap (CTS/PD3, RTS/PD4, TX/PD5, RX/PD6, CK/PD7)

Bit 2 **USART1_REMAP** *USART1 remapping*

This bit is set and cleared by software. It controls the mapping of USART1 TX and RX alternate functions on the GPIO ports.

0: No remap (TX/PA9, RX/PA10)

1: Remap (TX/PB6, RX/PB7)

Bit 1 **I2C1_REMAP** *I2C1 remapping*

This bit is set and cleared by software. It controls the mapping of I2C1 SCL and SDA alternate functions on the GPIO ports.

0: No remap (SCL/PB6, SDA/PB7)

1: Remap (SCL/PB8, SDA/PB9)

Bit 0 **SPI1_REMAP** *SPI1 remapping*

This bit is set and cleared by software. It controls the mapping of SPI1 NSS, SCK, MISO, MOSI alternate functions on the GPIO ports.

- 0: No remap (NSS/PA4, SCK/PA5, MISO/PA6, MOSI/PA7)
- 1: Remap (NSS/PA15, SCK/PB3, MISO/PB4, MOSI/PB5)

5.4.3 External interrupt configuration register 1 (AFIO_EXTICR1)

Address offset: 0x08

Reset value: 0x0000

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

Bits 31:16 Reserved

Bits 15:0 **EXTIx[3:0]**: *EXTIx x configuration (x= 0 to 3)*

These bits are written by software to select the source input for EXTIx external interrupt. Refer to [Section 6.2.5: External interrupt/event line mapping on page 102](#)

- 0000: PA[x] pin
- 0001: PB[x] pin
- 0010: PC[x] pin
- 0011: PD[x] pin
- 0100: PE[x] pin

5.4.4 External interrupt configuration register 2 (AFIO_EXTICR2)

Address offset: 0x0C

Reset value: 0x0000

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

Bits 31:16 Reserved

Bits 15:0 **EXTIx[3:0]**: *EXTIx x configuration (x= 4 to 7)*

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

- 0000: PA[x] pin
- 0001: PB[x] pin
- 0010: PC[x] pin
- 0011: PD[x] pin
- 0100: PE[x] pin

5.4.5 External interrupt configuration register 3 (AFIO_EXTICR3)

Address offset: 0x10

Reset value: 0x0000

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

Bits 31:16 Reserved

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

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

0000: PA[x] pin

0001: PB[x] pin

0010: PC[x] pin

0011: PD[x] pin

0100: PE[x] pin

5.4.6 External interrupt configuration register 4 (AFIO_EXTICR4)

Address offset: 0x14

Reset value: 0x0000

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

Bits 31:16 Reserved

Bits 15:0 **EXTIx[3:0]**: *EXTIx x configuration (x= 12 to 15)*

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

0000: PA[x] pin

0001: PB[x] pin

0010: PC[x] pin

0011: PD[x] pin

0100: PE[x] pin

5.5 GPIO and AFIO register maps

Refer to [Table 1 on page 27](#) for the register boundary addresses.

5.5.1 GPIO register map

Table 25. GPIO register map and reset values

5.5.2 AFIO register map

Table 26. AFIO register map and reset values

6 Interrupts and events

6.1 Nested vectored interrupt controller (NVIC)

Features

- 43 maskable interrupt channels (not including the 16 interrupt lines of Cortex™-M3)
- 16 programmable priority levels (4 bits of interrupt priority are used)
- Low-latency exception and interrupt handling
- Power management control
- Implementation of System Control Registers

The NVIC and the processor core interface are closely coupled, which enables low latency interrupt processing and efficient processing of late arriving interrupts.

All interrupts including the core exceptions are managed by the NVIC. For more information on exceptions and NVIC programming see Chap 5 Exceptions & Chap 8 Nested Vectored Interrupt Controller of the ARM *Cortex™-M3 Technical Reference Manual*.

6.1.1 SysTick calibration value register

The SysTick calibration value is fixed to 9000 which allows the generation of a time base of 1ms with the SysTick clock set to 9 MHz (max HCLK/8).

6.1.2 Interrupt and exception vectors

Table 27. 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) is linked to the NMI vector.	0x0000_0008
-1	fixed	HardFault		All class of fault	0x0000_000C
0	settable	MemManage		Memory management	0x0000_0010
1	settable	BusFault		Pre-fetch fault, memory access fault	0x0000_0014
2	settable	UsageFault		Undefined instruction or illegal state	0x0000_0018
-	-	-		Reserved	0x0000_001C - 0x0000_002B
3	settable	SVCAll		System service call via SWI instruction	0x0000_002C
4	settable	Debug Monitor		Debug Monitor	0x0000_0030
-	-	-		Reserved	0x0000_0034
5	settable	PendSV		Pendable request for system service	0x0000_0038

Table 27. Vector table (continued)

Position	Priority	Type of priority	Acronym	Description	Address
	6	settable	SysTick	System tick timer	0x0000_003C
0	7	settable	WWDG	Window Watchdog interrupt	0x0000_0040
1	8	settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044
2	9	settable	TAMPER	Tamper interrupt	0x0000_0048
3	10	settable	RTC	RTC global interrupt	0x0000_004C
4	11	settable	FLASH	Flash global interrupt	0x0000_0050
5	12	settable	RCC	RCC global interrupt	0x0000_0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000_0068
11	18	settable	DMA_Channel1	DMA Channel1 global interrupt	0x0000_006C
12	19	settable	DMA_Channel2	DMA Channel2 global interrupt	0x0000_0070
13	20	settable	DMA_Channel3	DMA Channel3 global interrupt	0x0000_0074
14	21	settable	DMA_Channel4	DMA Channel4 global interrupt	0x0000_0078
15	22	settable	DMA_Channel5	DMA Channel5 global interrupt	0x0000_007C
16	23	settable	DMA_Channel6	DMA Channel6 global interrupt	0x0000_0080
17	24	settable	DMA_Channel7	DMA Channel7 global interrupt	0x0000_0084
18	25	settable	ADC	ADC global interrupt	0x0000_0088
19	26	settable	USB_HP_CAN_TX	USB High Priority or CAN TX interrupts	0x0000_008C
20	27	settable	USB_LP_CAN_RX0	USB Low Priority or CAN RX0 interrupts	0x0000_0090
21	28	settable	CAN_RX1	CAN RX1 interrupt	0x0000_0094
22	29	settable	CAN_SCE	CAN SCE interrupt	0x0000_0098
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000_009C
24	31	settable	TIM1_BRK	TIM1 Break interrupt	0x0000_00A0
25	32	settable	TIM1_UP	TIM1 Update interrupt	0x0000_00A4
26	33	settable	TIM1_TRG_COM	TIM1 Trigger and Commutation interrupts	0x0000_00A8
27	34	settable	TIM1_CC	TIM1 Capture Compare interrupt	0x0000_00AC
28	35	settable	TIM2	TIM2 global interrupt	0x0000_00B0
29	36	settable	TIM3	TIM3 global interrupt	0x0000_00B4

Table 27. Vector table (continued)

Position	Priority	Type of priority	Acronym	Description	Address
30	37	settable	TIM4	TIM4 global interrupt	0x0000_00B8
31	38	settable	I2C1_EV	I ² C1 event interrupt	0x0000_00BC
32	39	settable	I2C1_ER	I ² C1 error interrupt	0x0000_00C0
33	40	settable	I2C2_EV	I ² C2 event interrupt	0x0000_00C4
34	41	settable	I2C2_ER	I ² C2 error interrupt	0x0000_00C8
35	42	settable	SPI1	SPI1 global interrupt	0x0000_00CC
36	43	settable	SPI2	SPI2 global interrupt	0x0000_00D0
37	44	settable	USART1	USART1 global interrupt	0x0000_00D4
38	45	settable	USART2	USART2 global interrupt	0x0000_00D8
39	46	settable	USART3	USART3 global interrupt	0x0000_00DC
40	47	settable	EXTI15_10	EXTI Line[15:10] interrupts	0x0000_00E0
41	48	settable	RTCAlarm	RTC alarm through EXTI line interrupt	0x0000_00E4
42	49	settable	USBWakeup	USB wakeup from suspend through EXTI line interrupt	0x0000_00E8

6.2 External interrupt/event controller (EXTI)

The external interrupt/event controller consists of up to 19 edge detectors for generating event/interrupt requests. Each input line can be independently configured to select the type (pulse or pending) and the corresponding trigger event (rising or falling or both). Each line can also be masked independently. A pending register maintains the status line of the interrupt requests.

6.2.1 Main features

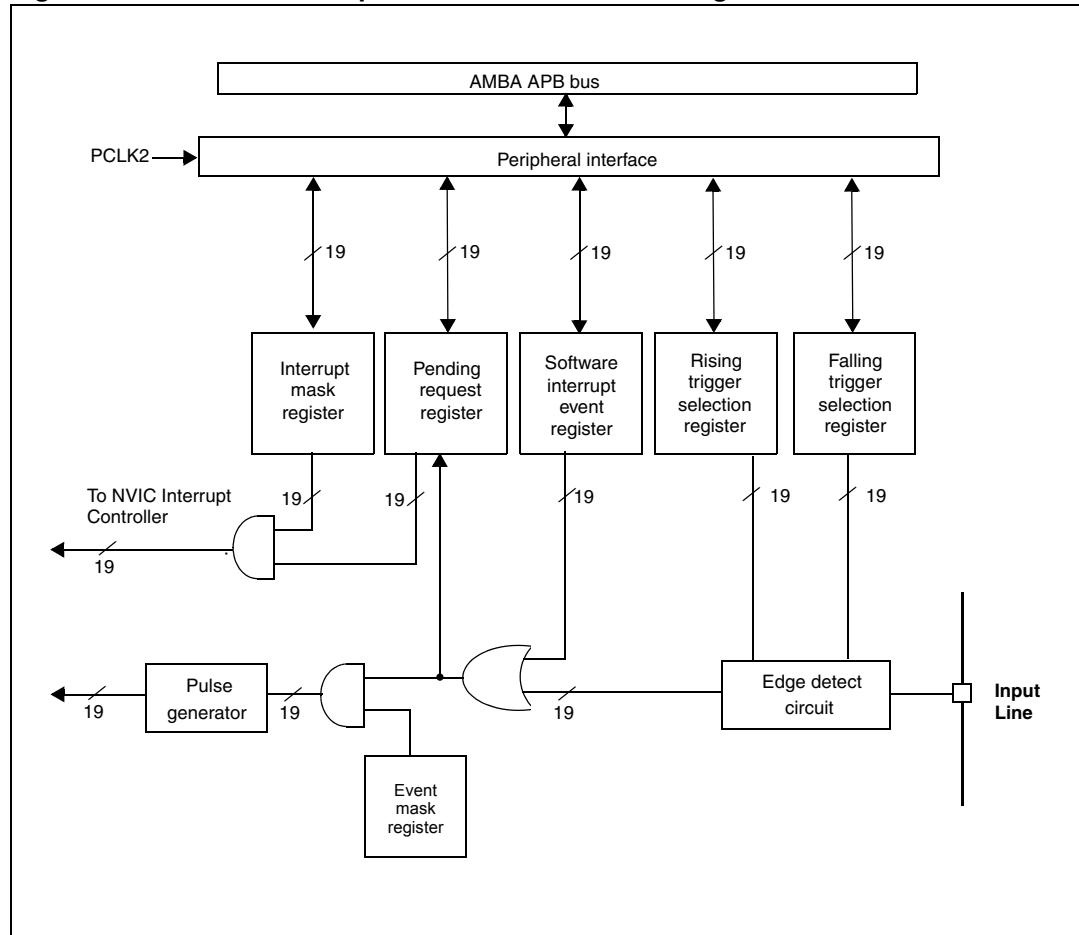
The EXTI controller main features are the following:

- Independent trigger and mask on each interrupt/event line
- Dedicated status bit for each interrupt line
- Generation of up to 19 software event/interrupt requests
- Detection of external signal with pulse width lower than APB2 clock period. Refer to the electrical characteristics section of the datasheet for details on this parameter.

6.2.2 Block diagram

The block diagram is shown in [Figure 15](#).

Figure 15. External interrupt/event controller block diagram



6.2.3 Wakeup event management

Cortex™-M3 is able to handle external events or internal events in order to wake up the core (WFE). By configuring the external lines any I/O port, RTC Alarm and USB Wakeup Events can be used to wake up the CPU (exit from WFE).

To use an external line as a wakeup event, refer to [Section 6.2.4: Functional description](#).

6.2.4 Functional description

To generate the interrupt, the interrupt line should be configured and enabled. This is done by programming the two trigger registers with the desired edge detection and by enabling the interrupt request by writing a ‘1’ to the corresponding bit in the interrupt mask register. When the selected edge occurs on the external interrupt line, an interrupt request is generated. The pending bit corresponding to the interrupt line is also set. This request is reset by writing a ‘1’ in the pending register.

To generate the event, the event line should be configured and enabled. This is done by programming the two trigger registers with the desired edge detection and by enabling the event request by writing a ‘1’ to the corresponding bit in the event mask register. When the selected edge occurs on the event line, an event pulse is generated. The pending bit corresponding to the event line is not set

An interrupt/event request can also be generated by software by writing a ‘1’ in the software interrupt/event register.

Hardware interrupt selection

To configure the 19 lines as interrupt sources, use the following procedure:

- Configure the mask bits of the 19 Interrupt lines (EXTI_IMR)
- Configure the Trigger Selection bits of the Interrupt lines (EXTI_RTSR and EXTI_FTSR)
- Configure the enable and mask bits that control the NVIC IRQ channel mapped to the External Interrupt Controller (EXTI) so that an interrupt coming from one of the 19 lines can be correctly acknowledged.

Hardware event selection

To configure the 19 lines as event sources, use the following procedure:

- Configure the mask bits of the 19 Event lines (EXTI_EMR)
- Configure the Trigger Selection bits of the Event lines (EXTI_RTSR and EXTI_FTSR)

Software interrupt/event selection

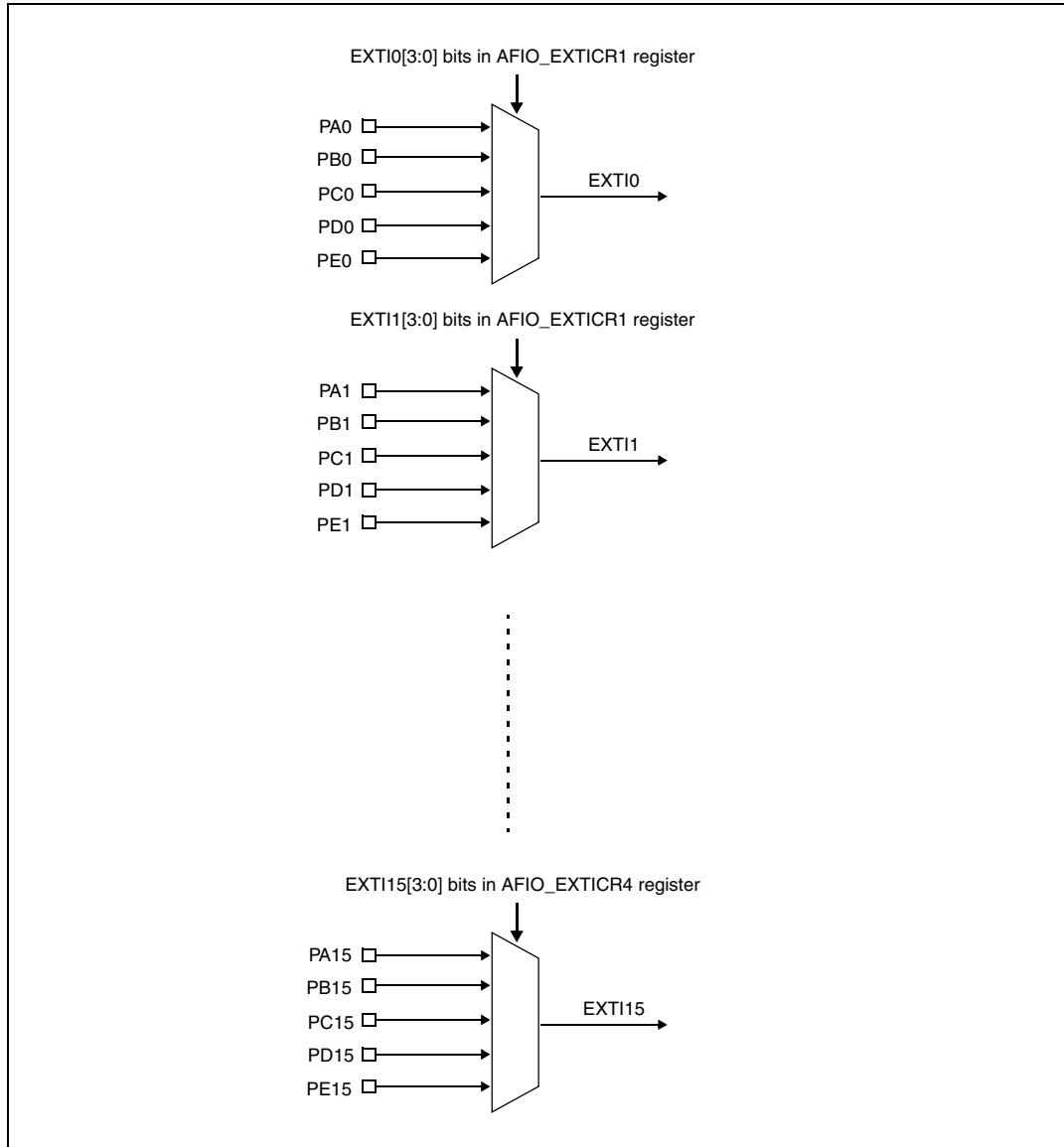
The 19 lines can be configured as software interrupt/event lines. The following is the procedure to generate a software interrupt.

- Configure the mask bits of the 19 Interrupt/Event lines (EXTI_IMR, EXTI_EMR)
- Set the required bit of the software interrupt register (EXTI_SWIER)

6.2.5 External interrupt/event line mapping

The 80 GPIOs are connected to the 16 external interrupt/event lines in the following manner:

Figure 16. External interrupt/event GPIO mapping



The three other EXTI lines are connected as follows:

- EXTI line 16 is connected to the PVD output
- EXTI line 17 is connected to the RTC Alarm event
- EXTI line 18 is connected to the USB Wakeup event

6.3 EXTI register description

Refer to [Section 1.1 on page 23](#) for a list of abbreviations used in register descriptions.

Interrupt mask register (EXTI_IMR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reserved														MR18	MR17	MR16
														rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:19 Reserved, must be kept at reset value (0).

Bits 18:0 **MRx: Interrupt Mask on line x**

0: Interrupt request from Line x is masked

1: Interrupt request from Line x is not masked

Event mask register (EXTI_EMR)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reserved														MR18	MR17	MR16
														rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:19 Reserved, must be kept at reset value (0).

Bits 18:0 **MRx: Event Mask on line x**

0: Event request from Line x is masked

1: Event request from Line x is not masked

Rising Trigger selection register (EXTI_RTSR)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reserved														TR18	TR17	TR16
rw														rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bits 31:19 Reserved, must be kept at reset value (0).

Bits 18:0 **TRx**: Rising trigger event configuration bit of line x

0: Rising trigger disabled (for Event and Interrupt) for input line

1: Rising trigger enabled (for Event and Interrupt) for input line.

Note: The external wakeup lines are edge triggered, no glitches must be generated on these lines. If a rising edge on external interrupt line occurs during writing of EXTI_RTSR register, the pending bit will not be set.

Rising and Falling edge triggers can be set for the same interrupt line. In this configuration, both generate a trigger condition.

Falling Trigger selection register (EXTI_FTSR)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reserved														TR18	TR17	TR16
rw														rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bits 31:19 Reserved, must be kept at reset value (0).

Bits 18:0 **TRx**: Falling trigger event configuration bit of line x

0: Falling trigger disabled (for Event and Interrupt) for input line

1: Falling trigger enabled (for Event and Interrupt) for input line.

Note: The external wakeup lines are edge triggered, no glitches must be generated on these lines. If a falling edge on external interrupt line occurs during writing of EXTI_FTSR register, the pending bit will not be set.

Rising and Falling edge triggers can be set for the same interrupt line. In this configuration, both generate a trigger condition.

Software interrupt event register (EXTI_SWIER)

Address offset: 0x10

Reset value: 0x0000 0000

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

Bits 31:19 Reserved, must be kept at reset value (0).

Bits 18:0 **SWIERx:** Software Interrupt on line x

Writing a 1 to this bit when it is at 0 sets the corresponding pending bit in EXTI_PR. If the interrupt is enabled on this line on the EXTI_IMR and EXTI_EMR, an interrupt request is generated.

This bit is cleared by clearing the corresponding bit of EXTI_PR (by writing a 1 into the bit).

Pending register (EXTI_PR)

Address offset: 0x14

Reset value: 0xxxxx xxxx

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reserved														PR18	PR17	PR16
														rc_w1	rc_w1	rc_w1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0	
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	

Bits 31:19 Reserved, must be kept at reset value (0).

Bits 18:0 **PRx:** Pending bit

0: No trigger request occurred

1: selected trigger request occurred

This bit is set when the selected edge event arrives on the external interrupt line. This bit is cleared by writing a 1 into the bit or by changing the sensitivity of the edge detector.

Note: If an interrupt request occurs one cycle before entering Stop mode, then the EXTI_PR register will be updated only after exit from Stop mode, generating an interrupt request if the corresponding bit in the EXTI_IMR register is set

6.3.1 EXTI register map

Table 28. External interrupt/event controller register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	EXTI_IMR																																
	Reset value																																
0x04	EXTI_EMR																																
	Reset value																																
0x08	EXTI_RTSR																																
	Reset value																																
0x0C	EXTI_FTSR																																
	Reset value																																
0x10	EXTI_SWIER																																
	Reset value																																
0x14	EXTI_PR																																
	Reset value															x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	

Refer to [Table 1 on page 27](#) for the register boundary addresses.

7 DMA controller (DMA)

7.1 Introduction

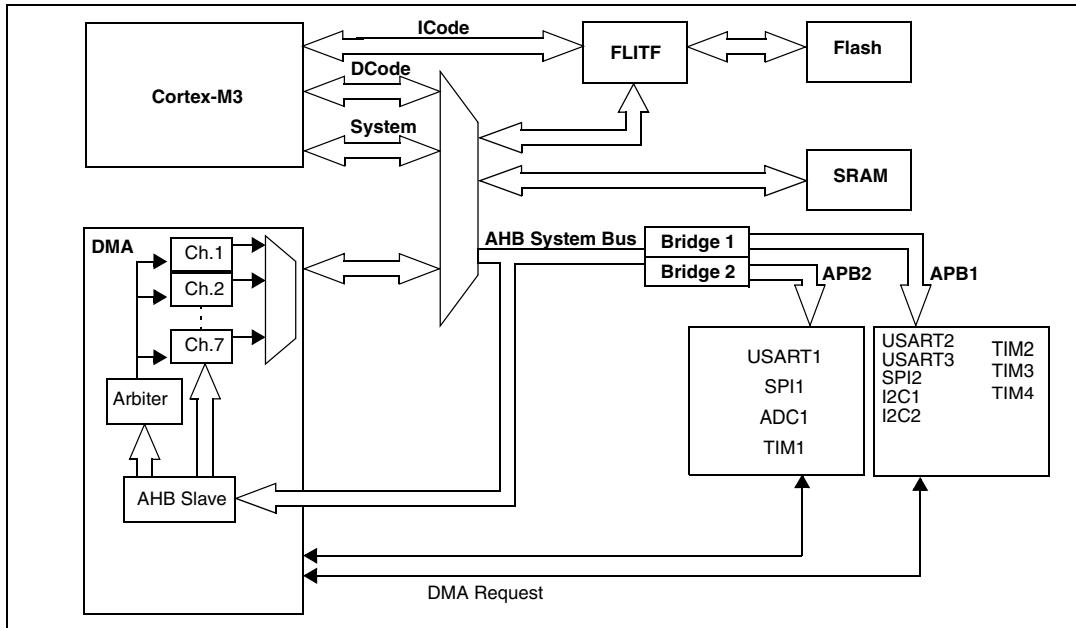
Direct memory access (DMA) is used in order to provide high-speed data transfer between peripherals and memory as well as memory to memory. Data can be quickly moved by DMA without any CPU actions. This keeps CPU resources free for other operations.

The DMA controller has 7 channels, each dedicated to managing memory access requests from one or more peripherals. It has an arbiter for handling the priority between DMA requests.

7.2 Main features

- 7 independently configurable channels (requests)
- Each of the 7 channels is connected to dedicated hardware DMA requests, software trigger is also supported on each channel. This configuration is done by software.
- Priorities between the seven requests are software programmable (4 levels consisting of *very high, high, medium, low*) or hardware in case of equality (request 1 has priority over request 2, etc.)
- Independent source and destination transfer size (byte, half word, word), emulating packing and unpacking.
- Support for circular buffer management
- 3 event flags (DMA Half Transfer, DMA Transfer complete and DMA Transfer Error) logically ORed together in a single interrupt request for each channel
- Memory-to-memory transfer
- Peripheral-to-memory and memory-to-peripheral, and peripheral-to-peripheral transfers
- Access to Flash, SRAM, peripheral SRAM, APB1 and APB2 peripherals as source and destination
- Programmable number of data to be transferred: up to 65536

The block diagram is shown in [Figure 17](#).

Figure 17. DMA block diagram

7.3 Functional description

The DMA controller performs direct memory transfer by sharing the system bus with the Cortex™-M3 core. The DMA request may stop the CPU access to the system bus for some bus cycles, when the CPU and DMA are targeting the same destination (RAM or peripheral). The bus matrix implements round-robin scheduling, thus ensuring at least half of the system bus bandwidth (both to RAM and peripheral) for the CPU.

7.3.1 DMA transactions

After an event, the peripheral sends a request signal to the DMA Controller. The DMA controller serves the request depending on the channel priorities. As soon as the DMA Controller accesses the peripheral, an Acknowledge is sent to the peripheral by the DMA Controller. The peripheral releases its request as soon as it gets the Acknowledge from the DMA Controller. Once the request is deasserted by the peripheral, the DMA Controller releases the Acknowledge. If there are more requests, the peripheral can initiate the next transaction.

In summary, each DMA transfer consists of three operations:

- A load from the peripheral data register or a location in memory addressed through the DMA_CMARx register
- A store of the data loaded to the peripheral data register or a location in memory addressed through the DMA_CMARx register
- A post-decrement of the DMA_CNDTRx register, which contains the number of transactions that have still to be performed.

7.3.2 Arbiter

The arbiter manages the channel requests based on their priority and launches the peripheral/memory access sequences.

The priorities are managed in two stages:

- Software: each channel priority can be configured in the DMA_CCRx register. There are four levels:
 - Very high priority
 - High priority
 - Medium priority
 - Low priority
- Hardware: if 2 requests have the same software priority level, the channel with the lowest number will get priority versus the channel with the highest number. For example, channel 2 gets priority over channel 4.

7.3.3 DMA channels

Each channel can handle DMA transfer between a peripheral register located at a fixed address and a memory address. The amount of data to be transferred (up to 65535) is programmable. The register which contains the amount of data items to be transferred is decremented after each transaction.

Programmable data sizes

Transfer data sizes of the peripheral and memory are fully programmable through the PSIZE and MSIZE bits in the DMA_CCRx register.

Pointer incrementation

Peripheral and memory pointers can optionally be automatically post-incremented after each transaction depending on the PINC and MINC bits in the DMA_CCRx register. If incremented mode is enabled, the address of the next transfer will be the address of the previous one incremented by 1, 2 or 4 depending on the chosen data size. The first transfer address will be the one programmed in the DMA_CPARx/DMA_CMARx registers.

If the channel is configured in non-circular mode, no DMA requests are served after the end of the transfer (i.e. once the number of data to be transferred reaches zero).

Channel configuration procedure

The following sequence should be followed to configure a DMA channelx (where x is the channel number).

1. Set the peripheral register address in the DMA_CPARx register. The data will be moved from/ to this address to/ from the memory after the peripheral event.
2. Set the memory address in the DMA_CMARx register. The data will be written to or read from this memory after the peripheral event.
3. Configure the total number of data to be transferred in the DMA_CNDTRx register. After each peripheral event, this value will be decremented.
4. Configure the channel priority using the PL[1:0] bits in the DMA_CCRx register
5. Configure data transfer direction, circular mode, peripheral & memory incremented mode, peripheral & memory data size, and interrupt after half and/or full transfer in the DMA_CCRx register
6. Activate the channel by setting the ENABLE bit in the DMA_CCRx register.

As soon as the channel is enabled, it can serve any DMA request from the peripheral connected on the channel.

Once half of the bytes are transferred, the Half-Transfer Flag (HTIF) is set and an interrupt is generated if the Half-Transfer Interrupt Enable bit (HTIE) is set. At the end of the transfer, the Transfer Complete Flag (TCIF) is set and an interrupt is generated if the Transfer Complete Interrupt Enable bit (TCIE) is set.

Circular mode

Circular mode is available to handle circular buffers and continuous data flows (e.g. ADC scan mode). This feature can be enabled using the CIRC bit in the DMA_CCRx register. When circular mode is activated, the number of data to be transferred is automatically reloaded with the initial value programmed during the channel configuration phase, and the DMA requests continue to be served.

Memory-to-memory mode

The DMA channels can also work without being triggered by a request from a peripheral. This mode is called Memory to Memory mode.

If the MEM2MEM bit in the DMA_CCRx register is set, then the channel initiates transfers as soon as it is enabled by software by setting the Enable bit (EN) in the DMA_CCRx register. The transfer stops once the DMA_CNDTRx register reaches zero. Memory to Memory mode may not be used at the same time as Circular mode.

7.3.4 Error management

In case of bus error during a DMA read or a write access, the faulty channel is automatically disabled with through a hardware clear of its EN bit in the corresponding Channel Configuration Register (DMA_CCRx). The channel's Transfer Error Interrupt Flag (TEIF) in the DMA_IFR register is set and an interrupt is generated if the Transfer Error Interrupt Enable bit (TEIE) in the DMA_CCRx register is set.

7.3.5 Interrupts

An interrupt can be produced on a Half-transfer, Transfer complete or Transfer error for each DMA channel. Separate interrupt enable bits are available for flexibility.

Table 29. DMA interrupt requests

Interrupt event	Event flag	Enable Control bit
Half-transfer	HTIF	HTIE
Transfer complete	TCIF	TCIE
Transfer error	TEIF	TEIE

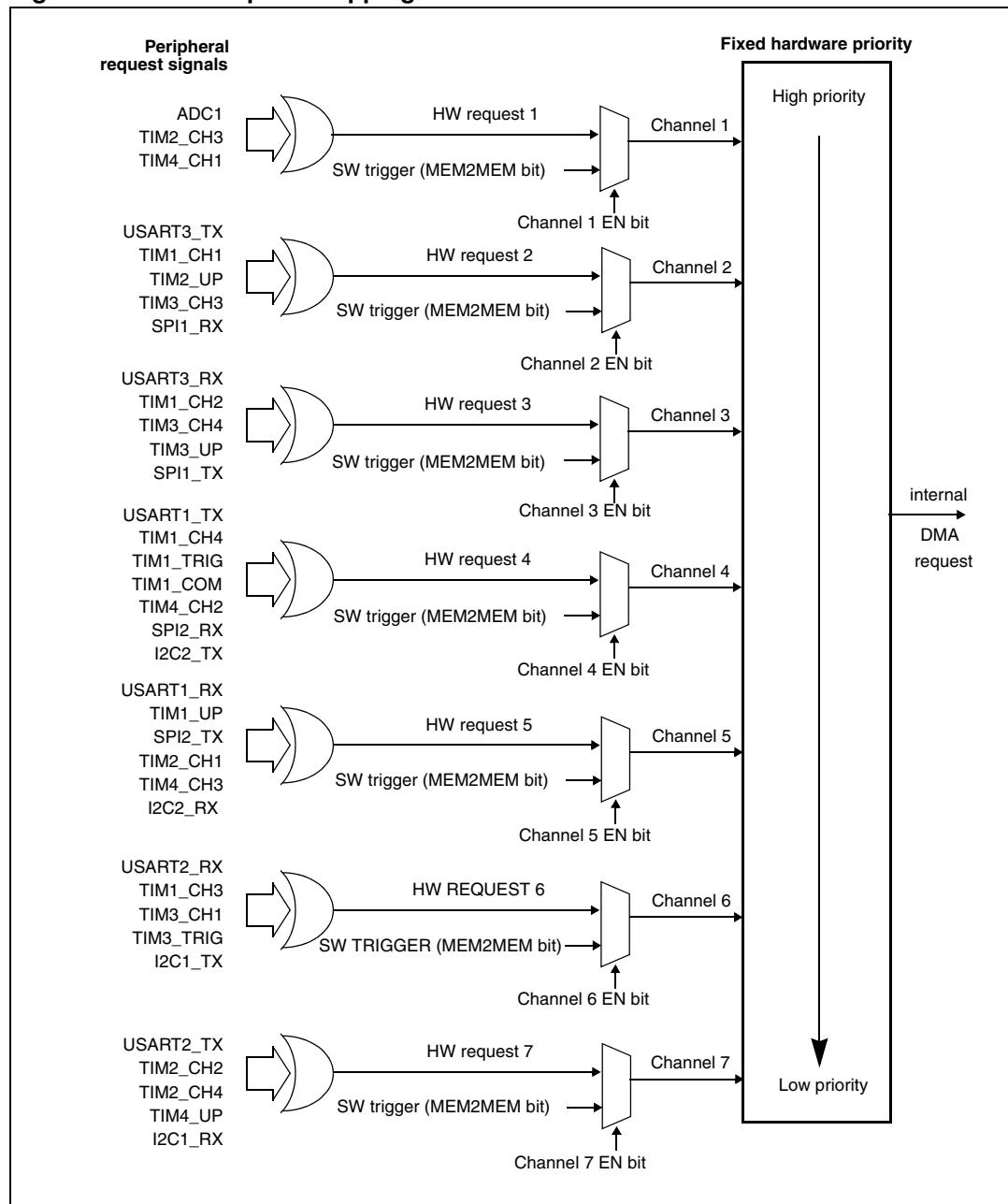
Note: DMA2 Channel4 and DMA2 Channel5 interrupts are mapped onto the same interrupt vector. All other DMA1 and DMA2 Channel interrupts have their own interrupt vector.

7.3.6 DMA request mapping

The 7 requests from the peripherals (TIMx, ADC, SPIx, I²Cx and USARTx) are simply logically ORed before entering DMA, this means that only one request must be enabled at a time. Refer to [Figure 18: DMA request mapping](#).

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

Figure 18. DMA request mapping



[Table 30](#) lists the DMA requests for each channel.

Table 30. Summary of DMA requests for each channel

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
ADC	ADC1						
SPI/I ² S		SPI1_RX	SPI1_TX	SPI/I2S2_RX	SPI/I2S2_TX		
USART		USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I ² C				I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
TIM1		TIM1_CH1	TIM1_CH2	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
TIM2	TIM2_CH3	TIM2_UP			TIM2_CH1		TIM2_CH2 TIM2_CH4
TIM3		TIM3_CH3	TIM3_CH4 TIM3_UP			TIM3_CH1 TIM3_TRIG	
TIM4	TIM4_CH1			TIM4_CH2	TIM4_CH3		TIM4_UP

7.4 DMA registers

Refer to [Section 1.1 on page 23](#) for a list of abbreviations used in the register descriptions.

7.4.1 DMA interrupt status register (DMA_ISR)

Address offset: 0x000

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				TEIF7	HTIF7	TCIF7	GIF7	TEIF6	HTIF6	TCIF6	GIF6	TEIF5	HTIF5	TCIF5	GIF5
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TEIF4	HTIF4	TCIF4	GIF4	TEIF3	HTIF3	TCIF3	GIF3	TEIF2	HTIF2	TCIF2	GIF2	TEIF1	HTIF1	TCIF1	GIF1
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:28 Reserved, always read as 0.

Bits 27, 23, **TEIF_x:** Channel x Transfer Error flag ($x = 1 \dots 7$)

19, 15, 11, This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the 7, 3 DMA_IFCR register.

0: No transfer error (TE) on channel x

1: A transfer error (TE) occurred on channel x

Bits 26, 22, **HTIF_x:** Channel x Half Transfer flag ($x = 1 \dots 7$)

18, 14, 10, This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the 6, 2 DMA_IFCR register.

0: No half transfer (HT) event on channel x

1: A half transfer (HT) event occurred on channel x

Bits 25, 21, **TCIF_x**: *Channel x Transfer Complete flag (x = 1 ..7)*

17, 13, 9, 5, This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the
1 DMA_IFCR register.

- 0: No transfer complete (TC) event on channel x
- 1: A transfer complete (TC) event occurred on channel x

Bits 24, 20, **GIF_x**: *Channel x Global interrupt flag (x = 1 ..7)*

16, 12, 8, 4, This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the
0 DMA_IFCR register.

- 0: No TE, HT or TC event on channel x
- 1: A TE, HT or TC event occurred on channel x

7.4.2 DMA interrupt flag clear register (DMA_IFCR)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				CTEIF 7	CHTIF 7	CTCIF 7	CGIF 7	CTEIF 6	CHTIF 6	CTCIF 6	CGIF 6	CTEIF 5	CHTIF 5	CTCIF 5	CGIF 5
				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
CTEIF 4	CHTIF 4	CTCIF 4	CGIF 4	CTEIF 3	CHTIF 3	CTCIF 3	CGIF 3	CTEIF 2	CHTIF 2	CTCIF 2	CGIF 2	CTEIF 1	CHTIF 1	CTCIF 1	CGIF 1
rw	rw	rw	rw												

Bits 31:28 Reserved, always read as 0.

Bits 27, 23, **CTEIF_x**: Channel x Transfer Error clear ($x = 1 \dots 7$)

19, 15, 11, This bit is set and cleared by software.

7, 3 0: No effect

1: Clears the corresponding TEIF flag in the DMA_ISR register

Bits 26, 22, **CHTIF_x**: Channel x Half Transfer clear ($x = 1 \dots 7$)

18, 14, 10, This bit is set and cleared by software.

6, 2 0: No effect

1: Clears the corresponding HTIF flag in the DMA_ISR register

Bits 25, 21, **CTCIF_x**: Channel x Transfer Complete clear ($x = 1 \dots 7$)

17, 13, 9, 5, This bit is set and cleared by software.

1 0: No effect

1: Clears the corresponding TCIF flag in the DMA_ISR register

Bits 24, 20, **CGIF_x**: Channel x Global interrupt clear ($x = 1 \dots 7$)

16, 12, 8, 4, This bit is set and cleared by software.

0 0: No effect

1: Clears the GIF, TEIF, HTIF and TCIF flags in the DMA_ISR register

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

Address offset: 0x08 + 20d × Channel number

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	MEM2 MEM	PL[1:0]		MSIZE[1:0]		PSIZE[1:0]		MINC	PINC	CIRC	DIR	TEIE	HTIE	TCIE	EN
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:15 Reserved, always read as 0.

Bit 14 **MEM2MEM:** *Memory to memory mode*

This bit is set and cleared by software.

- 0: Memory to memory mode disabled
- 1: Memory to memory mode enabled

Bits 13:12 **PL[1:0]:** *Channel Priority level*

These bits are set and cleared by software.

- 00: Low
- 01: Medium
- 10: High
- 11: Very high

Bits 11:10 **MSIZE[1:0]:** *Memory size*

These bits are set and cleared by software.

- 00: 8-bits
- 01: 16-bits
- 10: 32-bits
- 11: Reserved

Bits 9:8 **PSIZE[1:0]:** *Peripheral size*

These bits are set and cleared by software.

- 00: 8-bits
- 01: 16-bits
- 10: 32-bits
- 11: Reserved

Bit 7 **MINC:** *Memory increment mode*

This bit is set and cleared by software.

- 0: Memory increment mode disabled
- 1: Memory increment mode enabled

Bit 6 **PINC:** *Peripheral increment mode*

This bit is set and cleared by software.

- 0: Peripheral increment mode disabled
- 1: Peripheral increment mode enabled

Bit 5 **CIRC:** *Circular mode*

This bit is set and cleared by software.

0: Circular mode disabled

1: Circular mode enabled

Bit 4 **DIR:** *Data transfer direction*

This bit is set and cleared by software.

0: Read from peripheral

1: Read from memory

Bit 3 **TEIE:** *Transfer error interrupt enable*

This bit is set and cleared by software.

0: TE interrupt disabled

1: TE interrupt enabled

Bit 2 **HTIE:** *Half Transfer interrupt enable*

This bit is set and cleared by software.

0: HT interrupt disabled

1: HT interrupt enabled

Bit 1 **TCIE:** *Transfer complete interrupt enable*

This bit is set and cleared by software.

0: TC interrupt disabled

1: TC interrupt enabled

Bit 0 **EN:** *Channel enable*

This bit is set and cleared by software.

0: Channel disabled

1: Channel enabled

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

Address offset: 0x0C + 20d × Channel number

Reset value: 0x0000 0000

Bits 31:16 Reserved, always read as 0.

Bits 15:0 **NDT[15:0]:** *Number of data to Transfer*

Number of data to be transferred (0 up to 65535). This register can only be written when the channel is disabled. Once the channel is enabled, this register is read-only, indicating the remaining bytes to be transmitted. This register decrements after each DMA transfer.

Once the transfer is completed, this register can either stay at zero or be reloaded automatically by the value previously programmed if the channel is configured in auto-reload mode.

If this register is zero, no transaction can be served whether the channel is enabled or not.

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

Address offset: $0x10 + 20d \times \text{Channel number}$

Reset value: 0x0000 0000

Bits 31:0 **PA[31:0]: Peripheral Address**

Base address of the peripheral data register from/to which the data will be read/written.

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

Address offset: $0x14 + 20d \times \text{Channel number}$

Reset value: 0x0000 0000

Bits 31:0 **MA[31:0]**: *Memory Address*

Base address of the memory area from/to which the data will be read/written.

7.5 DMA register map

Table 31. DMA - register map and reset values

Table 31. 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
028h	DMA_CMAR2																																
	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		
02Ch																																	
030h	DMA_CCR3																																
	Reset value																																
034h	DMA_CNDTR3																																
	Reset value																																
038h	DMA_CPAR3																																
	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		
03Ch	DMA_CMAR3																																
	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		
040h																																	
044h	DMA_CCR4																																
	Reset value																																
048h	DMA_CNDTR4																																
	Reset value																																
04Ch	DMA_CPAR4																																
	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		
050h	DMA_CMAR4																																
	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		
054h																																	
058h	DMA_CCR5																																
	Reset value																																
05Ch	DMA_CNDTR5																																
	Reset value																																
060h	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		
064h	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		
068h																																	
06Ch	DMA_CCR6																																
	Reset value																																

Table 31. 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
0x070	DMA_CNDTR6																																
	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																																	
0x080	DMA_CCR7																																
	Reset value																																
0x084	DMA_CNDTR7																																
	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																																	

Refer to [Table 1 on page 27](#) for the register boundary addresses.

8 Real-time clock (RTC)

8.1 Introduction

The real-time clock is an independent timer. The RTC provides a set of continuously running counters which can be used, with suitable software, to provide a clock-calendar function. The counter values can be written to set the current time/date of the system.

The RTC core and clock configuration (RCC_BDCR register) are in the Backup domain, which means that RTC setting and time are kept after reset or wakeup from Standby mode.

After reset, access to Backup registers and RTC is disabled and the Backup domain is protected against possible parasitic write access. The DBP bit must be set in the Power Control Register (PWR_CR) to enable access to the Backup registers and RTC.

8.2 Main features

- Programmable prescaler: division factor up to 2^{20}
- 32-bit programmable counter for long-term measurement
- Two separate clocks: PCLK1 for the APB1 interface and RTC clock (must be at least four times slower than the PCLK1 clock)
- Two separate reset types:
 - The APB1 interface is reset by system reset
 - The RTC Core (Prescaler, Alarm, Counter and Divider) is reset only by a Backup domain reset (see [Section 4.1.3: Backup domain Reset on page 47](#)).
- Three dedicated maskable interrupt lines:
 - Alarm interrupt, for generating a software programmable alarm interrupt.
 - Seconds interrupt, for generating a periodic interrupt signal with a programmable period length (up to 1 second).
 - Overflow interrupt, to detect when the internal programmable counter rolls over to zero.

8.3 Functional description

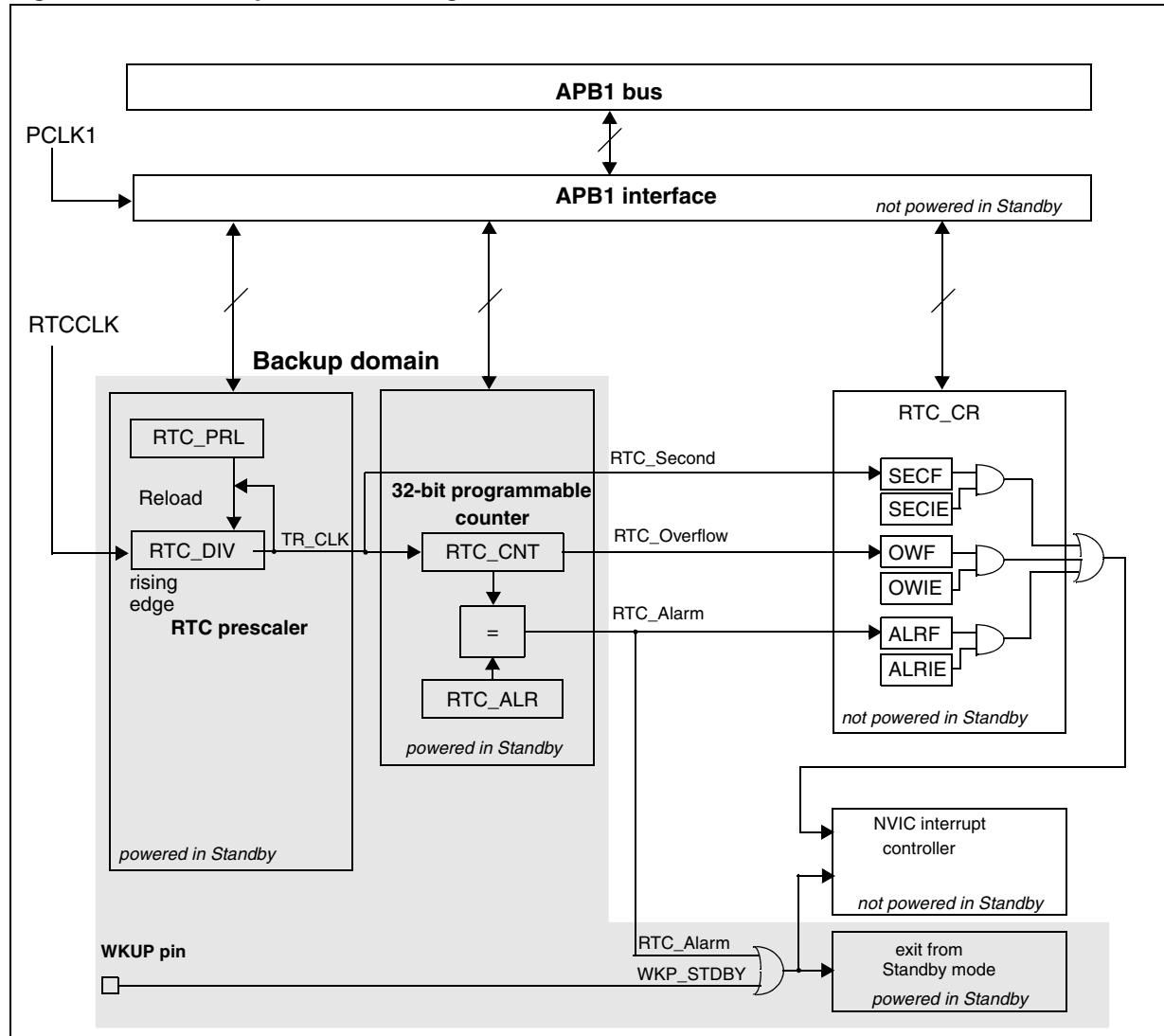
8.3.1 Overview

The RTC consists of two main units (see [Figure 19 on page 122](#)). The first one (APB1 Interface) is used to interface with the APB1 bus. This unit also contains a set of 16-bit registers accessible from the APB1 bus in read or write mode (for more information refer to [Section 8.4: RTC register description on page 125](#)). The APB1 interface is clocked by the APB1 bus clock in order to interface with the APB1 bus.

The other unit (RTC Core) consists of a chain of programmable counters made of two main blocks. The first block is the RTC prescaler block, which generates the RTC time base TR_CLK that can be programmed to have a period of up to 1 second. It includes a 20-bit programmable divider (RTC Prescaler). Every TR_CLK period, the RTC generates an interrupt (Second Interrupt) if it is enabled in the RTC_CR register. The second block is a 32-bit programmable counter that can be initialized to the current system time. The system

time is incremented at the TR_CLK rate and compared with a programmable date (stored in the RTC_ALR register) in order to generate an alarm interrupt, if enabled in the RTC_CR control register.

Figure 19. RTC simplified block diagram



8.3.2 Resetting RTC registers

All system registers are asynchronously reset by a System Reset or Power Reset, except for RTC_PRL, RTC_ALR, RTC_CNT, and RTC_DIV.

The RTC_PRL, RTC_ALR, RTC_CNT, and RTC_DIV registers are reset only by a Backup Domain reset. Refer to [Section 4.1.3 on page 47](#).

8.3.3 Reading RTC registers

The RTC core is completely independent from the RTC APB1 interface.

Software accesses the RTC prescaler, counter and alarm values through the APB1 interface but the associated readable registers are internally updated at each rising edge of the RTC clock resynchronized by the RTC APB1 clock. This is also true for the RTC flags.

This means that the first read to the RTC APB1 registers may be corrupted (generally read as 0) if the APB1 interface has previously been disabled and the read occurs immediately after the APB1 interface is enabled but before the first internal update of the registers. This can occur if:

- A system reset or power reset has occurred
- The MCU has just woken up from Standby mode (see [Section 3.3: Low-power modes](#))
- The MCU has just woken up from Stop mode (see [Section 3.3: Low-power modes](#))

In all the above cases, the RTC core has been kept running while the APB1 interface was disabled (reset, not clocked or unpowered).

Consequently when reading the RTC registers, after having disabled the RTC APB1 interface, the software must first wait for the RSF bit (Register Synchronized Flag) in the RTC_CRL register to be set by hardware.

Note that the RTC APB1 interface is not affected by WFI and WFE low-power modes.

8.3.4 Configuring RTC registers

To write in the RTC_PRL, RTC_CNT, RTC_ALR registers, the peripheral must enter Configuration Mode. This is done by setting the CNF bit in the RTC_CRL register.

In addition, writing to any RTC register is only enabled if the previous write operation is finished. To enable the software to detect this situation, the RTOFF status bit is provided in the RTC_CR register to indicate that an update of the registers is in progress. A new value can be written to the RTC registers only when the RTOFF status bit value is '1'.

Configuration procedure:

1. Poll RTOFF, wait until its value goes to '1'
2. Set the CNF bit to enter configuration mode
3. Write to one or more RTC registers
4. Clear the CNF bit to exit configuration mode
5. Poll RTOFF, wait until its value goes to '1' to check the end of the write operation.

The write operation only executes when the CNF bit is cleared; it takes at least three RTCCLK cycles to complete.

8.3.5 RTC flag assertion

The RTC Second flag (SECF) is asserted on each RTC Core clock cycle before the update of the RTC Counter.

The RTC Overflow flag (OWF) is asserted on the last RTC Core clock cycle before the counter reaches 0x0000.

The RTC_Alarm and RTC Alarm flag (ALRF) (see [Figure 20](#)) are asserted on the last RTC Core clock cycle before the counter reaches the RTC Alarm value stored in the Alarm register increased by one (RTC_ALR + 1). The write operation in the RTC Alarm and RTC Second flag must be synchronized by using one of the following sequences:

- Use the RTC Alarm interrupt and inside the RTC interrupt routine, the RTC Alarm and/or RTC Counter registers are updated.
- Wait for SECF bit to be set in the RTC Control register. Update the RTC Alarm and/or the RTC Counter register.

Figure 20. RTC second and alarm waveform example with PR=0003, ALARM=00004

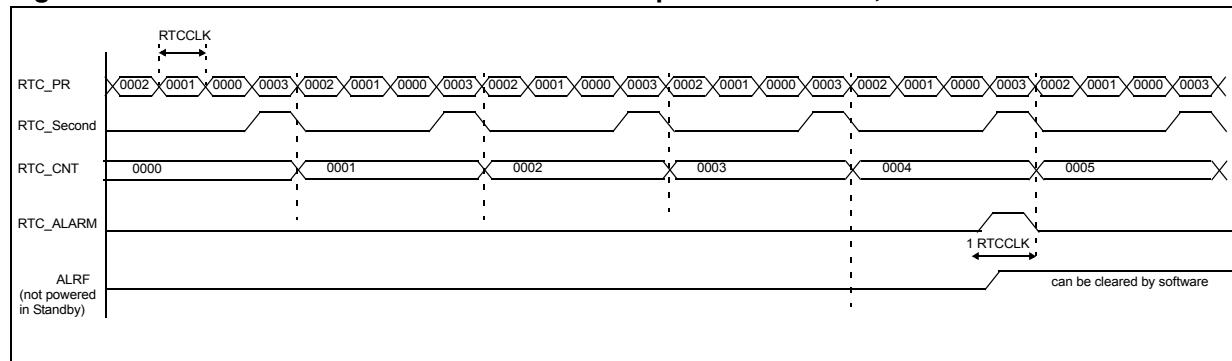
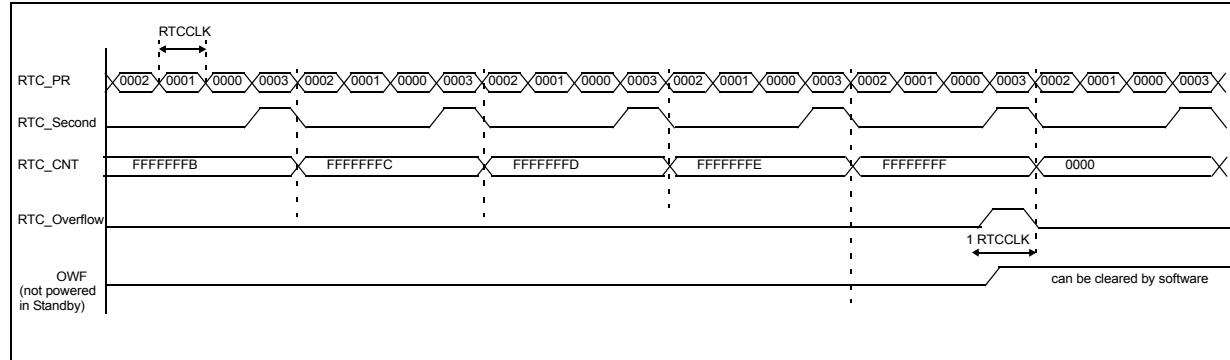


Figure 21. RTC Overflow waveform example with PR=0003

8.4 RTC register description

Refer to [Section 1.1 on page 23](#) for a list of abbreviations used in register descriptions.

8.4.1 RTC control register High (RTC_CRH)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												OWIE	ALRIE	SECIE	
rw										rw	rw	rw			

Bits 15:3 Reserved, forced by hardware to 0.

Bit 2 **OWIE:** *Overflow Interrupt Enable*

- 0: Overflow interrupt is masked.
- 1: Overflow interrupt is enabled.

Bit 1 **ALRIE:** *Alarm Interrupt Enable*

- 0: Alarm interrupt is masked.
- 1: Alarm interrupt is enabled.

Bit 0 **SECIE:** *Second Interrupt Enable*

- 0: Second interrupt is masked.
- 1: Second interrupt is enabled.

These bits are used to mask interrupt requests. Note that at reset all interrupts are disabled, so it is possible to write to the RTC registers to ensure that no interrupt requests are pending after initialization. It is not possible to write to the RTC_CRH register when the peripheral is completing a previous write operation (flagged by RTOFF=0, see [Section 8.3.4 on page 124](#)).

The RTC functions are controlled by this control register. Some bits must be written using a specific configuration procedure (see [Configuration procedure](#)).

8.4.2 RTC control register low (RTC_CRL)

Address offset: 0x04

Reset value: 0x0020

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								RTOFF	CNF	RSF	OWF	ALRF	SECF		
								r	rw	rc_w0	rc_w0	rc_w0	rc_w0		

Bits 15:6 Reserved, forced by hardware to 0.

Bit 5 **RTOFF**: RTC operation OFF

With this bit the RTC reports the status of the last write operation performed on its registers, indicating if it has been completed or not. If its value is '0' then it is not possible to write to any of the RTC registers. This bit is read only.

0: Last write operation on RTC registers is still ongoing.

1: Last write operation on RTC registers terminated.

Bit 4 **CNF**: Configuration Flag

This bit must be set by software to enter in configuration mode so as to allow new values to be written in the RTC_CNT, RTC_ALR or RTC_PRL registers. The write operation is only executed when the CNF bit is reset by software after has been set.

0: Exit configuration mode (start update of RTC registers).

1: Enter configuration mode.

Bit 3 **RSF**: Registers Synchronized Flag

This bit is set by hardware at each time the RTC_CNT and RTC_DIV registers are updated and cleared by software. Before any read operation after an APB1 reset or an APB1 clock stop, this bit must be cleared by software, and the user application must wait until it is set to be sure that the RTC_CNT, RTC_ALR or RTC_PRL registers are synchronized.

0: Registers not yet synchronized.

1: Registers synchronized.

Bit 2 **OWF**: Overflow Flag

This bit is set by hardware when the 32-bit programmable counter overflows. An interrupt is generated if OWIE=1 in the RTC_CRH register. It can be cleared only by software. Writing '1' has no effect.

0: Overflow not detected

1: 32-bit programmable counter overflow occurred.

Bit 1 **ALRF**: Alarm Flag

This bit is set by hardware when the 32-bit programmable counter reaches the threshold set in the RTC_ALR register. An interrupt is generated if ALRIE=1 in the RTC_CRH register. It can be cleared only by software. Writing '1' has no effect.

0: Alarm not detected

1: Alarm detected

Bit 0 **SECF**: Second Flag

This bit is set by hardware when the 32-bit programmable prescaler overflows, thus incrementing the RTC counter. Hence this flag provides a periodic signal with a period corresponding to the resolution programmed for the RTC counter (usually one second). An interrupt is generated if SECIE=1 in the RTC_CRH register. It can be cleared only by software. Writing '1' has no effect.

0: Second flag condition not met.

1: Second flag condition met.

The functions of the RTC are controlled by this control register. It is not possible to write to the RTC_CR register while the peripheral is completing a previous write operation (flagged by RTOFF=0, see [Section 8.3.4 on page 124](#)).

- Note:**
- 1 *Any flag remains pending until the appropriate RTC_CR request bit is reset by software, indicating that the interrupt request has been granted.*
 - 2 *At reset the interrupts are disabled, no interrupt requests are pending and it is possible to write to the RTC registers.*
 - 3 *The OWF, ALRF, SECF and RSF bits are not updated when the APB1 clock is not running.*
 - 4 *The OWF, ALRF, SECF and RSF bits can only be set by hardware and only cleared by software.*
 - 5 *If ALRF = 1 and ALRIE = 1, the RTC global interrupt is enabled. If EXTI Line 17 is also enabled through the EXTI Controller, both the RTC global interrupt and the RTC Alarm interrupt are enabled.*
 - 6 *If ALRF = 1, the RTC Alarm interrupt is enabled if EXTI Line 17 is enabled through the EXTI Controller in interrupt mode. When the EXTI Line 17 is enabled in event mode, a pulse is generated on this line (no RTC Alarm interrupt generation).*

8.4.3 RTC prescaler load register (RTC_PRLH / RTC_PRLL)

The Prescaler Load registers keep the period counting value of the RTC prescaler. They are write-protected by the RTOFF bit in the RTC_CR register, and a write operation is allowed if the RTOFF value is '1'.

RTC prescaler load register high (RTC_PRLH)

Address offset: 0x08

Write only (see [Section 8.3.4 on page 124](#))

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								PRL[19:16]							
W								W							

Bits 15:4 Reserved, forced by hardware to 0.

Bits 3:0 **PRL[19:16]: RTC Prescaler Reload Value High**

These bits are used to define the counter clock frequency according to the following formula:

$$f_{TR_CLK} = f_{RTCCLK}/(PRL[19:0]+1)$$

Caution: The zero value is not recommended. RTC interrupts and flags cannot be asserted correctly.

RTC prescaler load register low (RTC_PRLL)

Address offset: 0x0C

Write only (see [Section 8.3.4 on page 124](#))

Reset value: 0x8000

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

Bits 15:0 **PRL[15:0]: RTC Prescaler Reload Value Low**

These bits are used to define the counter clock frequency according to the following formula:

$$f_{TR_CLK} = f_{RTCCLK}/(PRL[19:0]+1)$$

Note: If the input clock frequency (f_{RTCCLK}) is 32.768 kHz, write 7FFFh in this register to get a signal period of 1 second.

8.4.4 RTC prescaler divider register (RTC_DIVH / RTC_DIVL)

During each period of TR_CLK, the counter inside the RTC prescaler is reloaded with the value stored in the RTC_PRL register. To get an accurate time measurement it is possible to read the current value of the prescaler counter, stored in the RTC_DIV register, without stopping it. This register is read-only and it is reloaded by hardware after any change in the RTC_PRL or RTC_CNT registers.

RTC prescaler divider register high (RTC_DIVH)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												RTC_DIV[19:16]			
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:4 Reserved

Bits 3:0 **RTC_DIV[19:16]: RTC Clock Divider High**

RTC prescaler divider register low (RTC_DIVL)

Address offset: 0x14

Reset value: 0x8000

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

Bits 15:0 **RTC_DIV[15:0]: RTC Clock Divider Low**

8.4.5 RTC counter register (RTC_CNTH / RTC_CNTL)

The RTC core has one 32-bit programmable counter, accessed through two 16-bit registers; the count rate is based on the TR_CLK time reference, generated by the prescaler.

RTC_CNT registers keep the counting value of this counter. They are write-protected by bit RTOFF in the RTC_CR register, and a write operation is allowed if the RTOFF value is '1'. A write operation on the upper (RTC_CNTH) or lower (RTC_CNTL) registers directly loads the corresponding programmable counter and reloads the RTC Prescaler. When reading, the current value in the counter (system date) is returned.

RTC counter register high (RTC_CNTH)

Address offset: 0x18

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTC_CNT[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 RTC_CNT[31:16]: RTC Counter High

Reading the RTC_CNTH register, the current value of the high part of the RTC Counter register is returned. To write to this register it is necessary to enter configuration mode (see [Section 8.3.4: Configuring RTC registers on page 124](#)).

RTC counter register low (RTC_CNTL)

Address offset: 0x1C

Reset value: 0x0000

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

Bits 15:0 RTC_CNT[15:0]: RTC Counter Low

Reading the RTC_CNTL register, the current value of the lower part of the RTC Counter register is returned. To write to this register it is necessary to enter configuration mode (see [Section 8.3.4: Configuring RTC registers on page 124](#)).

8.4.6 RTC alarm register high (RTC_ALRH / RTC_ALRL)

When the programmable counter reaches the 32-bit value stored in the RTC_ALR register, an alarm is triggered and the RTC_alarmIT interrupt request is generated. This register is write-protected by the RTOFF bit in the RTC_CR register, and a write operation is allowed if the RTOFF value is ‘1’.

RTC alarm register high (RTC_ALRH)

Address offset: 0x20

Write only (see [Section 8.3.4 on page 124](#))

Reset value: 0xFFFF

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTC_ALR[31:16]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 15:0 RTC_ALR[31:16]: RTC Alarm High

The high part of the alarm time is written by software in this register. To write to this register it is necessary to enter configuration mode (see [Section 8.3.4: Configuring RTC registers on page 124](#)).

RTC alarm register low (RTC_ALRL)

Address offset: 0x24

Write only (see [Section 8.3.4 on page 124](#))

Reset value: 0xFFFF

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

Bits 15:0 RTC_ALR[15:0]: RTC Alarm Low

The low part of the alarm time is written by software in this register. To write to this register it is necessary to enter configuration mode (see [Section 8.3.4: Configuring RTC registers on page 124](#)).

8.5 RTC register map

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

Table 32. RTC - register map and reset values

Refer to [Table 1 on page 27](#) for the register boundary addresses.

9 Backup registers (BKP)

9.1 Introduction

The backup registers are ten 16-bit registers for storing 20 bytes of user application data. They are implemented in the backup domain that remains powered on by V_{BAT} when the V_{DD} power is switched off. They are not reset when the device wakes up from Standby mode or by a system reset or power reset.

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

After reset, the access to Backup registers and RTC is disabled and the Backup domain is protected against possible parasitic write access.

The DBP bit must be set in the *Power control register (PWR_CR)* to enable access to the Backup registers and RTC.

9.2 Features

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

9.3 Tamper detection

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

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

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

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

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

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

9.4 RTC calibration

For measurement purposes, the 32.768 kHz RTC clock can be output on the TAMPER pin. This is enabled by setting the CCO bit in the [RTC clock calibration register \(BKP_RTCCR\)](#).

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

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

9.5 BKP register description

Refer to [Section 1.1 on page 23](#) for a list of abbreviations used in register descriptions.

9.5.1 Backup data register x (BKP_DRx) (x = 1 ..10)

Address offset: 0x04 to 0x28

Reset value: 0x0000 0000

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

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

These bits can be written with user data.

Note: The BKP_DRx registers are not reset by a System reset or Power reset or when the device wakes up from Standby mode.

They are reset by a Backup Domain reset or by a TAMPER pin event (if the TAMPER pin function is activated).

9.5.2 RTC clock calibration register (BKP_RTCCR)

Address offset: 0x2C

Reset value: 0x0000 0000

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

Bits 15:10 Reserved, always read as 0.

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

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

0: RTC Alarm pulse output selected

1: RTC Second pulse output selected

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

Bit 8 ASOE Alarm or Second Output Enable

Setting this bit outputs either the RTC Alarm pulse signal or the Second pulse signal on the TAMPER pin depending on the ASOS bit.

The output pulse duration is one LSE period. The TAMPER pin must not be enabled while the ASOE bit is set.

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

Bit 7 CCO Calibration Clock Output

0: No effect

1: Setting this bit outputs the RTC clock with a frequency divided by 64 on the TAMPER pin. The TAMPER pin must not be enabled while the CCO bit is set in order to avoid unwanted Tamper detection.

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

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

This value indicates the number of clock pulses that will be ignored every 2²⁰ clock pulses. This allows the calibration of the RTC, slowing down the clock by steps of 1000000/2²⁰ PPM.

The clock of the RTC can be slowed down from 0 to 121PPM.

9.5.3 Backup control register (BKP_CR)

Address offset: 0x30

Reset value: 0x0000 0000

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

Bits 15:2 Reserved, always read as 0.

Bit 1 TPAL TAMPER pin active level

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

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

Bit 0 TPE TAMPER pin enable

0: The TAMPER pin is free for general purpose I/O

1: Tamper alternate I/O function is activated.

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

9.5.4 Backup control/status register (BKP_CSR)

Address offset: 0x34

Reset value: 0x0000 0000

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

Bits 15:10 Reserved, always read as 0.

Bit 9 **TIF Tamper Interrupt Flag**

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

0: No Tamper interrupt

1: A Tamper interrupt occurred

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

Bit 8 **TEF Tamper Event Flag**

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

0: No Tamper event

1: A Tamper event occurred

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

Bits 7:3 Reserved, always read as 0.

Bit 2 **TPIE TAMPER Pin interrupt enable**

0: Tamper interrupt disabled

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

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

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

Bit 1 **CTI Clear Tamper Interrupt**

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

0: No effect

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

Bit 0 **CTE Clear Tamper event**

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

0: No effect

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

9.6 BKP register map

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

Table 33. BKP - register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h																																	
04h	BKP_DR1 Reset value																																
08h	BKP_DR2 Reset value																																
0Ch	BKP_DR3 Reset value																																
10h	BKP_DR4 Reset value																																
14h	BKP_DR5 Reset value																																
18h	BKP_DR6 Reset value																																
1Ch	BKP_DR7 Reset value																																
20h	BKP_DR8 Reset value																																
24h	BKP_DR9 Reset value																																
28h	BKP_DR10 Reset value																																
2C	BKP_RTCCR Reset value																																
30h	BKP_CR Reset value																																
34h	BKP_CSR Reset value																																

Refer to [Table 1 on page 27](#) for the register boundary addresses.

10 Independent watchdog (IWDG)

The STM32F10xxx has two embedded watchdog peripherals which offer a combination of high safety level, timing accuracy and flexibility of use. Both Watchdog peripherals (Independent and Window) serve to detect and resolve malfunctions due to software failure, and to trigger system reset or an interrupt (Window Watchdog only) when the counter reaches a given timeout value.

The independent watchdog (IWDG) is clocked by its own dedicated low-speed clock (40 kHz) and thus stays active even if the main clock fails. The Window Watchdog (WWDG) clock is prescaled from the APB1 clock and has a configurable time-window that can be programmed to detect abnormally late or early application behavior.

The IWDG is best suited to applications which require the watchdog to run as a totally independent process outside the main application, but have lower timing accuracy constraints. The WWDG is best suited to applications which require the watchdog to react within an accurate timing window.

For further information on the Window Watchdog, refer to [Section 11 on page 144](#).

10.1 Introduction

Figure 22 shows the functional blocks of the independent Watchdog module.

When the independent watchdog is started by writing the value 0xCCCC in the Key Register (IWDG_KR), the counter starts counting down from the reset value of 0xFFFF. When it reaches the end of count value (0x000) a reset signal is generated (IWDG RESET).

Whenever the key value 0xAAAA is written in the IWDG_KR register, the IWDG_RLR value is reloaded in the counter and the watchdog reset is prevented.

10.1.1 Hardware watchdog

If the “Hardware watchdog” feature is enabled through the device option bits, the watchdog is automatically enabled at power on, and will generate a reset unless the Key register is written by the software before the counter reaches end of count.

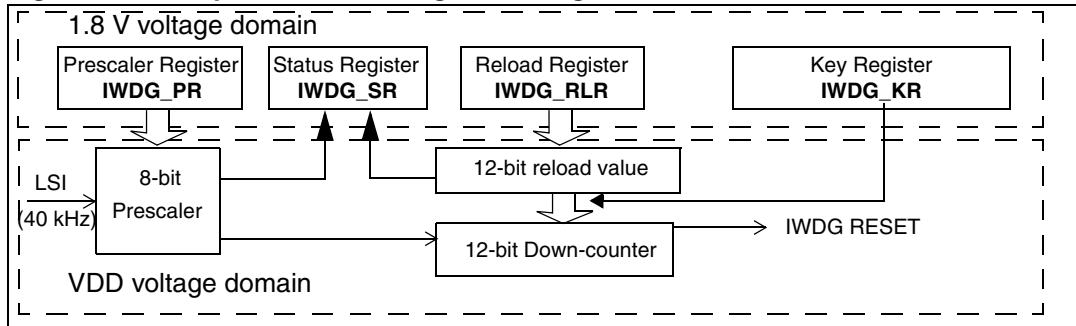
10.1.2 Register access protection

Write access to the IWDG_PR and IWDG_RLR registers is protected. To modify them, you must first write the code 0x5555 in the IWDG_KR register. A write access to this register with a different value will break the sequence and register access will be protected again. This implies that it is the case of the reload operation (writing 0xAAAA).

A status register is available to indicate that an update of the prescaler or the down-counter reload value is on going.

10.1.3 Debug mode

When the microcontroller enters debug mode (Cortex-M3 core halted), the IWDG counter either continues to work normally or stops, depending on DBG_IWDG_STOP configuration bit in DBG module. For more details, refer to [Section 20.15.2: Debug support for timers, watchdog, bxCAN and I2C](#).

Figure 22. Independent watchdog block diagram

Note:

The watchdog function is implemented in the V_{DD} voltage domain that is still functional in Stop and Standby modes.

Table 34. Watchdog timeout period (with 40 kHz input clock)⁽¹⁾

Prescaler divider	PR[2:0] bits	Min timeout (ms) RL[11:0]= 0x000	Max timeout (ms) RL[11:0]= 0xFFFF
/4	0	0.1	409.6
/8	1	0.2	819.2
/16	2	0.4	1638.4
/32	3	0.8	3276.8
/64	4	1.6	6553.6
/128	5	3.2	13107.2
/256	6 (or 7)	6.4	26214.4

- These timings are given for a 40 kHz clock but the microcontroller's internal RC frequency can vary from 30 to 60 kHz. Moreover, given an exact RC oscillator frequency, the exact timings still depend on the phasing of the APB interface clock versus the RC oscillator 40 kHz clock so that there is always a full RC period of uncertainty.

10.2 IWDG register description

Refer to [Section 1.1 on page 23](#) for a list of abbreviations used in register descriptions.

10.2.1 Key register (IWDG_KR)

Address offset: 0x00

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

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

Bits 31:16 Reserved, read as 0.

Bits 15:0 **KEY[15:0]: Key value (write only, read 0000h)**

These bits must be written by software at regular intervals with the key value AAAAh, otherwise the watchdog generates a reset when the counter reaches 0.

Writing the key value 5555h to enables access to the IWDG_PR and IWDG_RLR registers (see [Section 10.1.2](#))

Writing the key value CCCCh starts the watchdog (except if the hardware watchdog option is selected)

10.2.2 Prescaler register (IWDG_PR)

Address offset: 0x04

Reset value: 0x0000 0000

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

Bits 31:3 Reserved, read as 0.

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

These bits are write access protected see [Section 10.1.2](#). They are written by software to select the prescaler divider feeding the counter clock. PVU bit of IWDG_SR must be reset in order to be able to change the prescaler divider.

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

Note: Reading this register returns the prescaler value from the VDD voltage domain. This value may not be up to date/valid if a write operation to this register is ongoing. For this reason the value read from this register is valid only when the PVU bit in the IWDG_SR register is reset.

10.2.3 Reload register (IWDG_RLR)

Address offset: 0x08

Reset value: 0x0000 0FFF (reset by Standby mode)

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

Bits 31:12 Reserved, read as 0.

Bits11:0 **RL[11:0]: Watchdog counter reload value**

These bits are write access protected see [Section 10.1.2](#). They are written by software to define the value to be loaded in the watchdog counter each time the value AAAAh is written in the IWDG_KR register. The watchdog counter counts down from this value. The timeout period is a function of this value and the clock prescaler. Refer to [Table 34](#).

The RVU bit in the IWDG_SR register must be reset in order to be able to change the reload value.

Note: reading this register returns the reload value from the VDD voltage domain. This value may not be up to date/valid if a write operation to this register is ongoing on this register. For this reason the value read from this register is valid only when the RVU bit in the IWDG_SR register is reset.

10.2.4 Status register (IWDG_SR)

Address offset: 0x0C

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

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

Bits 31:2 Reserved

Bit 1 **RVU**: Watchdog counter reload value update

This bit is set by hardware to indicate that an update of the reload value is ongoing. It is reset by hardware when the reload value update operation is completed in the V_{DD} voltage domain (takes up to 5 RC 40 kHz cycles).

Reload value can be updated only when RVU bit is reset.

Bit 0 **PVU**: Watchdog prescaler value update

This bit is set by hardware to indicate that an update of the prescaler value is ongoing. It is reset by hardware when the prescaler update operation is completed in the V_{DD} voltage domain (takes up to 5 RC 40 kHz cycles).

Prescaler value can be updated only when PVU bit is reset.

Note:

If several reload values or prescaler values are used by application, it is mandatory to wait until RVU bit is reset before changing the reload value and to wait until PVU bit is reset before changing the prescaler value. However, after updating the prescaler and/or the reload value it is not necessary to wait until RVU or PVU is reset before continuing code execution (even in case of low-power mode entry, the write operation is taken into account and will complete)

10.3 IWDG register map

Table 35. IWDG register map and reset values

Refer to [Table 1 on page 27](#) for the register boundary addresses.

11 Window watchdog (WWDG)

11.1 Introduction

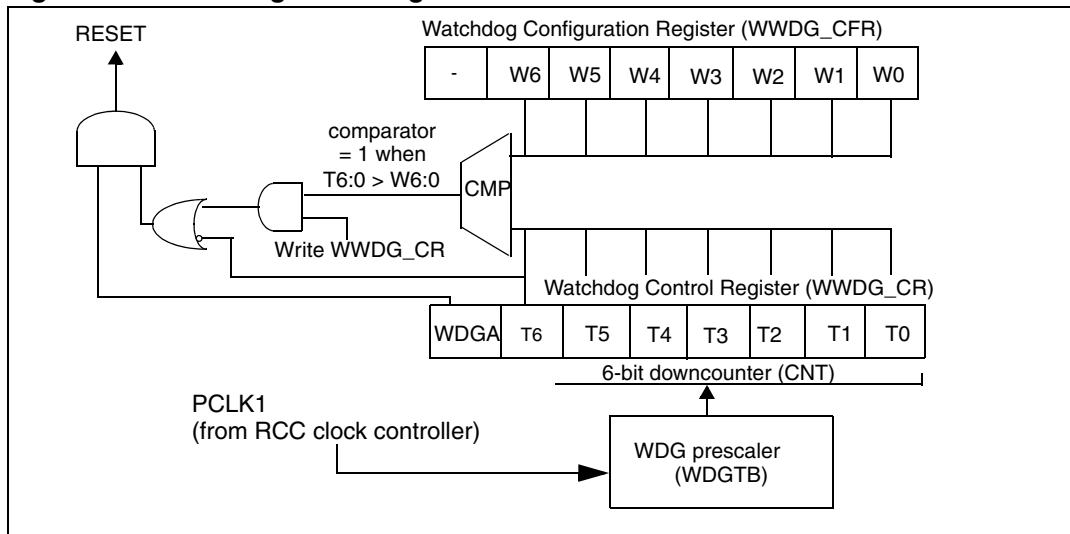
The window watchdog is used to detect the occurrence of a software fault, usually generated by external interference or by unforeseen logical conditions, which causes the application program to abandon its normal sequence. The Watchdog circuit generates an MCU reset on expiry of a programmed time period,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

11.2 Main features

- Programmable free-running downcounter
- Conditional reset
 - Reset (if watchdog activated) when the downcounter value becomes less than 40h
 - Reset (if watchdog activated) if the downcounter is reloaded outside the window (see [Figure 24](#))
- Early wakeup interrupt (EWI): triggered (if enabled and the watchdog activated) when the downcounter is equal to 40h. Can be used to reload the counter and prevent WWDG reset

11.3 Functional description

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

Figure 23. Watchdog block diagram

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

- Enabling the watchdog:
The watchdog is always disabled after a reset. It is enabled by setting the WDGA bit in the WWDG_CR register, then it cannot be disabled again except by a reset.
- Controlling the downcounter:
This downcounter is free-running: It counts down even if the watchdog is disabled. When the watchdog is enabled, the T6 bit must be set to prevent generating an immediate reset.
The T[5:0] bits contain the number of increments which represents the time delay before the watchdog produces a reset. The timing varies between a minimum and a maximum value due to the unknown status of the prescaler when writing to the WWDG_CR register (see [Figure 24](#)).
The Configuration register (WWDG_CFR) contains the high limit of the window: To prevent a reset, the downcounter must be reloaded when its value is lower than the window register value and greater than 0x3F. [Figure 24](#) describes the window watchdog process.
Another way to reload the counter is to use the early wakeup interrupt (EWI). This interrupt is enabled by setting the EWI bit in the WWDG_CFR register. When the downcounter reaches the value 40h, this interrupt is generated and the corresponding interrupt service routine (ISR) can be used to reload the counter to prevent WWDG reset.
This interrupt is cleared by writing '0' to the EWIF bit in the WWDG_SR register.

Note:

The T6 bit can be used to generate a software reset (the WDGA bit is set and the T6 bit is cleared).

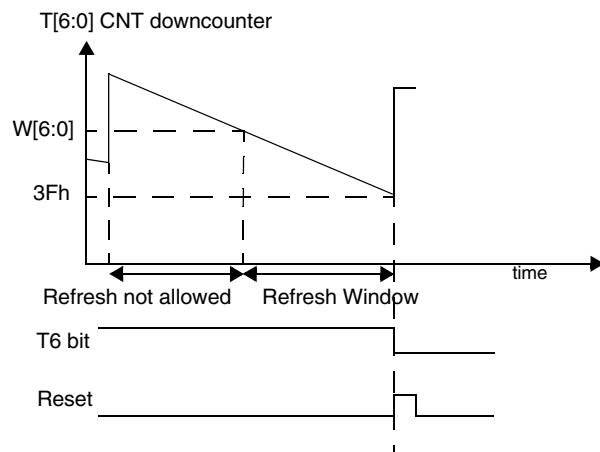
11.4**How to program the watchdog timeout**

[Figure 24](#) shows the linear relationship between the 6-bit value to be loaded in the Watchdog Counter (CNT) and the resulting timeout duration in milliseconds. This can be

used for a quick calculation without taking the timing variations into account. If more precision is needed, use the formulae in [Figure 24](#).

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

Figure 24. Window watchdog timing diagram



The formula to calculate the timeout value is given by:

$$T_{\text{WWDG}} = T_{\text{PCLK1}} \times 4096 \times 2^{\text{WDGTB}} \times (T[5:0] + 1) \quad ;(\text{ms})$$

where:

T_{WWDG} : WWDG timeout

T_{PCLK1} : APB1 Clock period measured in ms

Min-max timeout value @36MHz (PCLK1)

WDGTB	Min timeout value	Max timeout value
0	113 µs	7.28 ms
1	227 µs	14.56 ms
2	455 µs	29.12 ms
3	910 µs	58.25 ms

11.5 Debug mode

When the microcontroller enters debug mode (Cortex-M3 core halted), the WWDG counter either continues to work normally or stops, depending on DBG_WWDG_STOP configuration bit in DBG module. For more details, refer to [Section 20.15.2: Debug support for timers, watchdog, bxCAN and I2C](#).

11.6 Register description

Refer to [Section 1.1 on page 23](#) for a list of abbreviations used in register descriptions.

11.6.1 Control Register (WWDG_CR)

Address offset: 0x00

Reset value: 0x7F

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								WDGA	T6	T5	T4	T3	T2	T1	T0
Res.								rs	rw						

Bits 31:8 Reserved

Bit 7 WDGA: Activation bit

This bit is set by software and only cleared by hardware after a reset. When WDGA = 1, the watchdog can generate a reset.

0: Watchdog disabled

1: Watchdog enabled

Bits 6:0 T[6:0]: 7-bit counter (MSB to LSB).

These bits contain the value of the watchdog counter. It is decremented every $(4096 \times 2^{\text{WDGTB}})$ PCLK1 cycles. A reset is produced when it rolls over from 40h to 3Fh (T6 becomes cleared).

11.6.2 Configuration register (WWDG_CFR)

Address offset: 0x04

Reset value: 0x7F

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								EWI	WDG TB1	WDG TB0	W6	W5	W4	W3	W2
Res.								rs	rw	rw	rw	rw	rw	rw	rw

Bit 31:10 Reserved

Bit 9 EWI: Early Wakeup Interrupt

When set, an interrupt occurs whenever the counter reaches the value 40h. This interrupt is only cleared by hardware after a reset.

Bits 8:7 **WDGTB[1:0]: Timer Base**

The time base of the prescaler can be modified as follows:

- 00: CK Counter Clock (PCLK1 div 4096) div 1
- 01: CK Counter Clock (PCLK1 div 4096) div 2
- 10: CK Counter Clock (PCLK1 div 4096) div 4
- 11: CK Counter Clock (PCLK1 div 4096) div 8

Bits 6:0 **W[6:0] 7-bit window value**

These bits contain the window value to be compared to the downcounter.

11.6.3 Status register (WWDG_SR)

Address offset: 0x08

Reset value: 0x00

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															
Res.															EWIF
rc_w0															

Bit 31:1 Reserved

Bit 0 EWIF: Early Wakeup Interrupt Flag

This bit is set by hardware when the counter has reached the value 40h. It must be cleared by software by writing '0'. A write of '1' has no effect. This bit is also set if the interrupt is not enabled.

11.7 WWDG register map**Table 36. WWDG register map and reset values**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	WWDG_CR	Reserved																WDGA	T[6:0]														
		Reset value																	0	1	1	1	1	1	1								
0x04	WWDG_CFR	Reserved																EWI	W[6:0]														
		Reset value																	0	0	0	1	1	1	1	1	1	1	1	1			
0x08	WWDG_SR	Reserved																EWIF															
	Reset value																		0														

Refer to [Table 1 on page 27](#) for the register boundary addresses.

12 Advanced control timer (TIM1)

12.1 Introduction

The advanced control timer (TIM1) 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, 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 12.4.20](#).

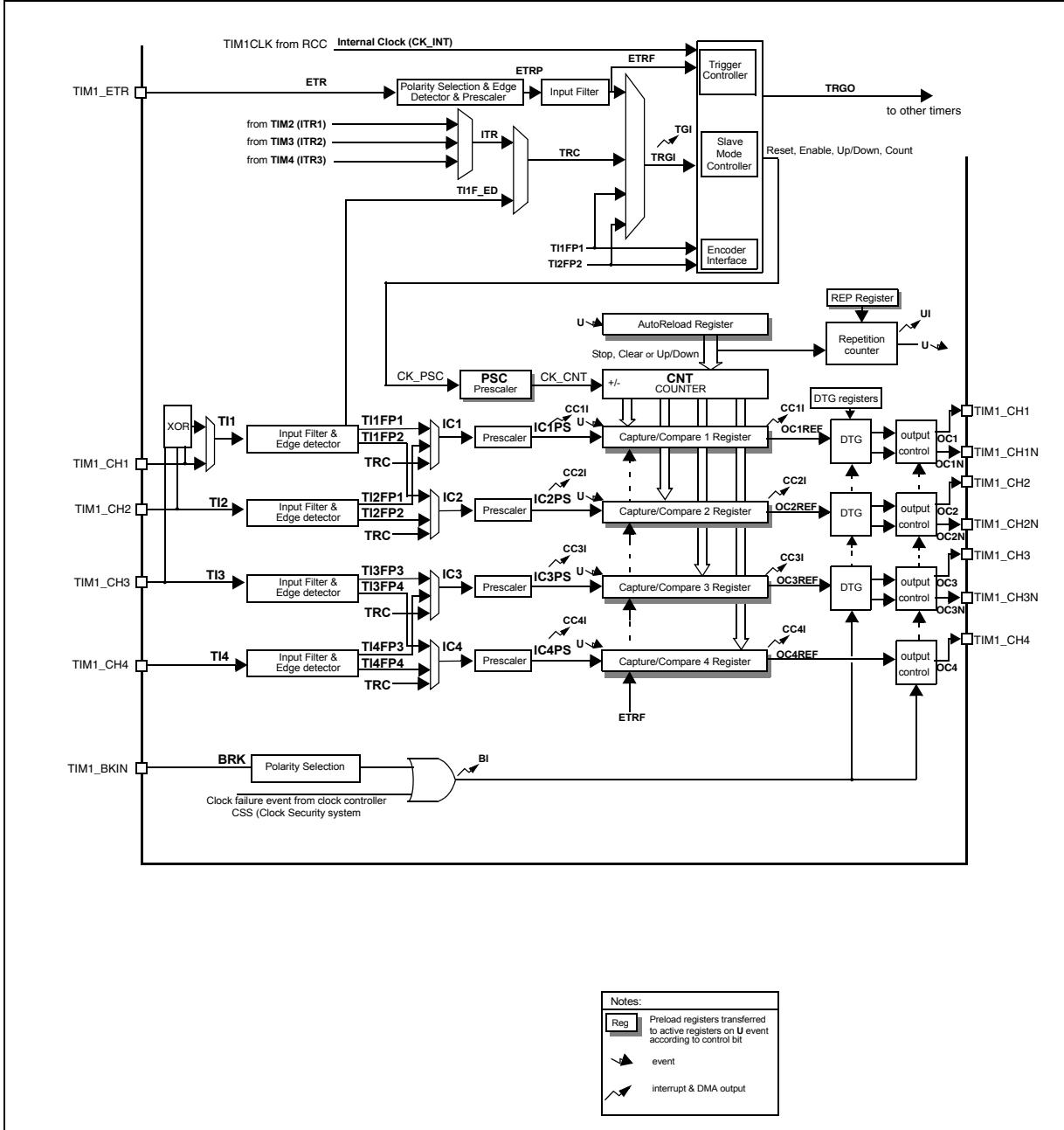
12.2 Main features

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 Mode)
 - One Pulse Mode output
 - Complementary Outputs with programmable dead-time
- Synchronization circuit to control the timer with external signals and to interconnect several timers together.
- Repetition counter to update the timer registers only after a given number of cycles of the counter.
- Break input to put the timer’s output signals in reset state or in a known state.
- Interrupt/DMA generation on the following events:
 - Update: counter overflow/underflow, counter initialization (by software or internal/external trigger)
 - Trigger event (counter start, stop, initialization or count by internal/external trigger)
 - Input capture
 - Output compare
 - Break input

12.3 Block diagram

Figure 25. Advanced control timer (TIM1) block diagram



12.4 Functional description

12.4.1 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 (TIM1_CNT)
- Prescaler Register (TIM1_PSC)
- Auto-Reload Register (TIM1_ARR)
- Repetition Counter Register (TIM1_RCR)

The auto-reload register is preloaded. Writing or reading the auto-reload register access the preload register. The content of the preload register is transferred in the shadow register permanently or at each update event UEV, depending on the auto-reload preload enable bit (ARPE) in TIM1_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 TIM1_CR1 register. It can also be generated by software. The generation of the update event is described in detail for each configuration.

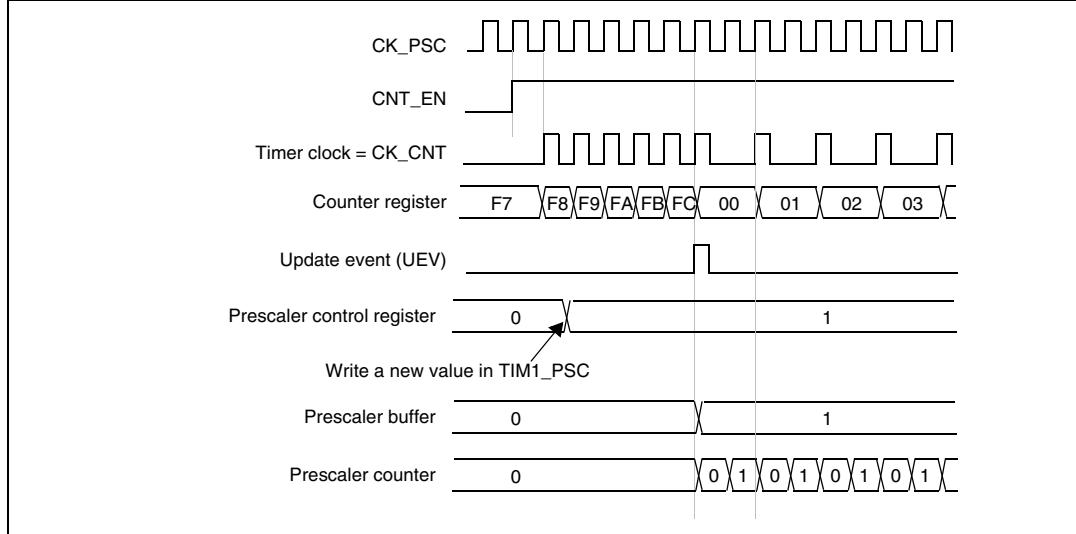
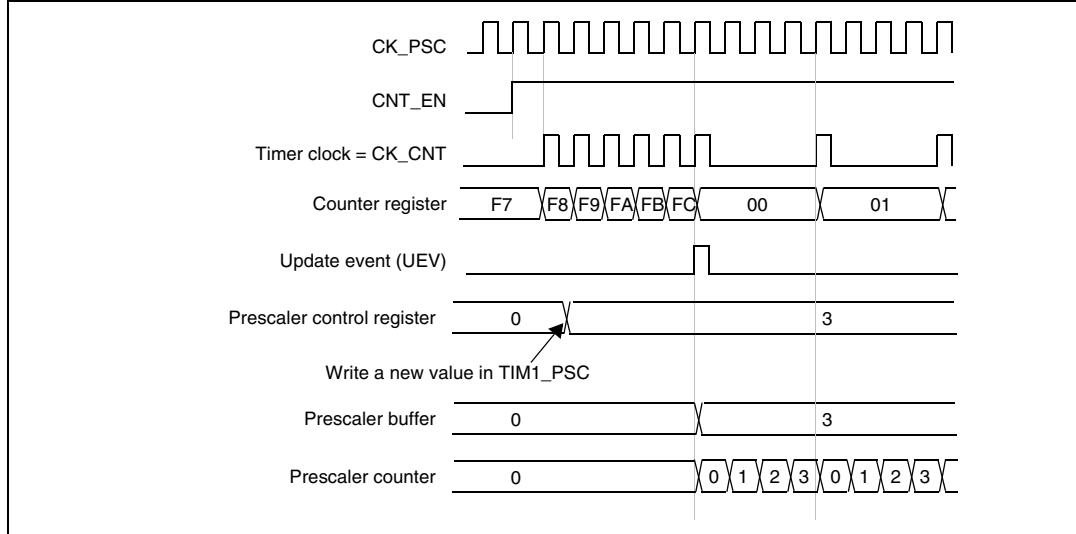
The counter is clocked by the prescaler output CK_CNT, which is enabled only when the counter enable bit (CEN) in TIM1_CR1 register is set (refer also to the slave mode controller description to get more details on counter enabling).

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

Prescaler description

The prescaler can divide the counter clock frequency by any factor between 1 and 65536. It is based on a 16-bit counter controlled through a 16-bit register (in the TIM1_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 27 and *Figure 28* give some examples of the counter behavior when the prescaler ratio is changed on the fly:

Figure 26. Counter timing diagram with prescaler division change from 1 to 2**Figure 27. Counter timing diagram with prescaler division change from 1 to 4**

12.4.2 Counter modes

Upcounting mode

In upcounting mode, the counter counts from 0 to the auto-reload value (content of the TIM1_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 (TIM1_RCR). Else the update event is generated at each counter overflow.

Setting the UG bit in the TIM1_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 TIM1_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 doesn't change). In addition, if the URS bit (update request selection) in TIM1_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 TIM1_SR register) is set (depending on the URS bit):

- The repetition counter is reloaded with the content of TIM1_RCR register,
- The auto-reload shadow register is updated with the preload value (TIM1_ARR),
- The buffer of the prescaler is reloaded with the preload value (content of the TIM1_PSC register).

The following figures show some examples of the counter behavior for different clock frequencies when TIM1_ARR=0x36.

Figure 28. Counter timing diagram, internal clock divided by 1

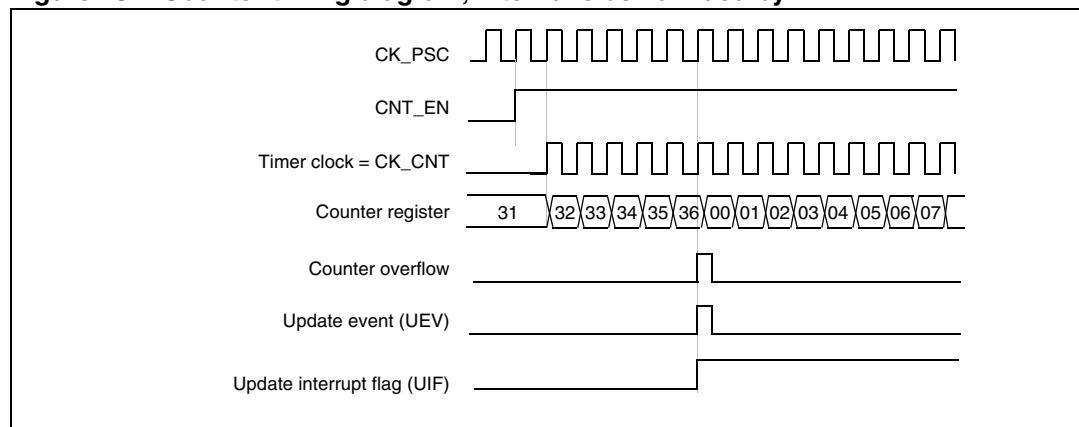


Figure 29. Counter timing diagram, internal clock divided by 2

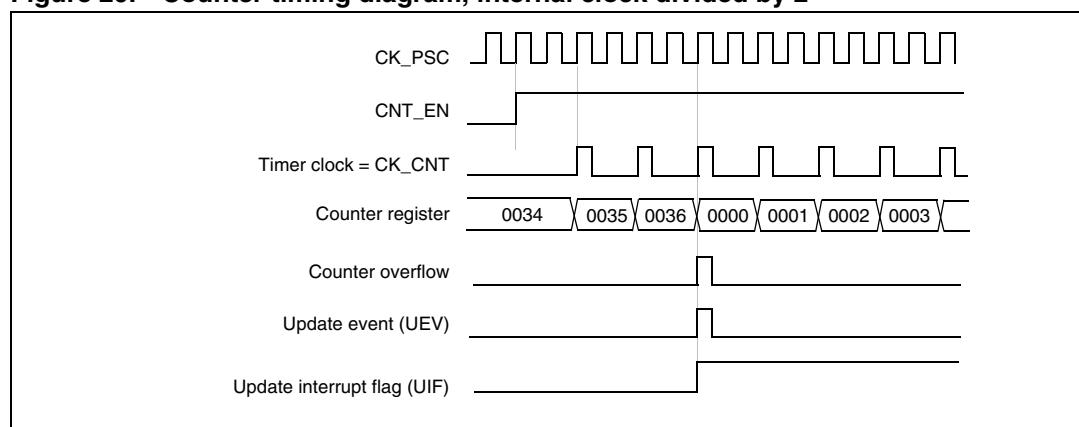


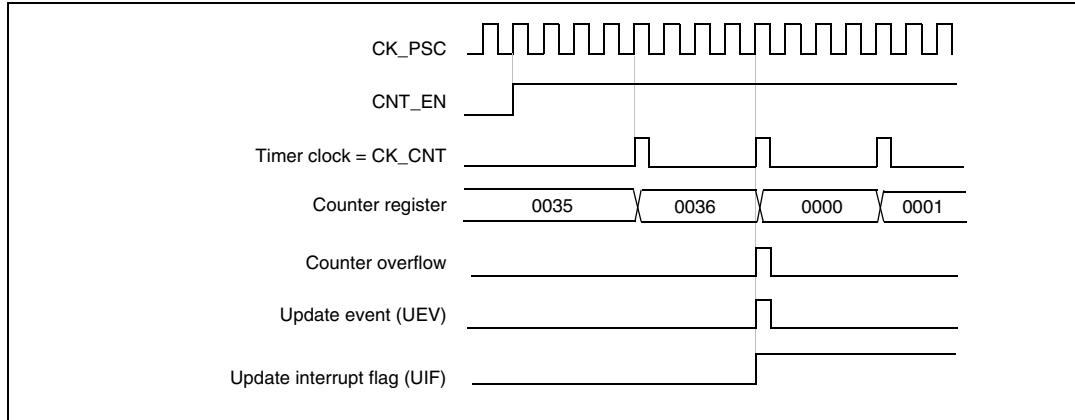
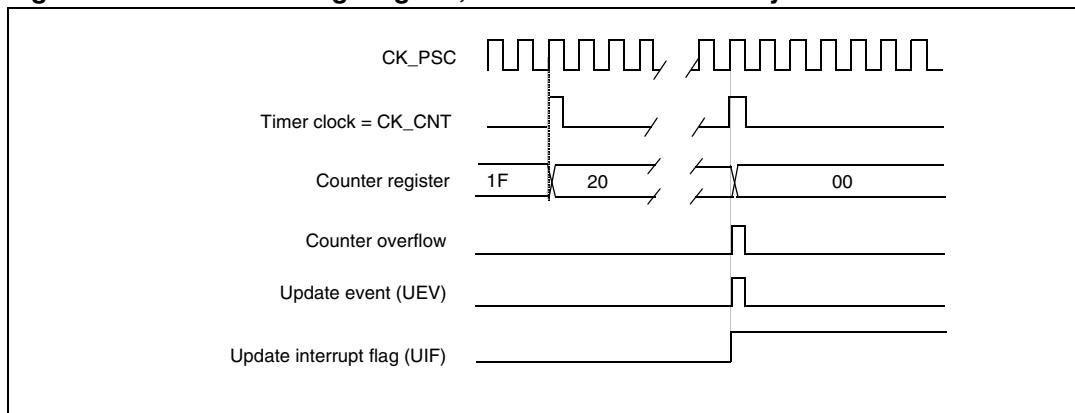
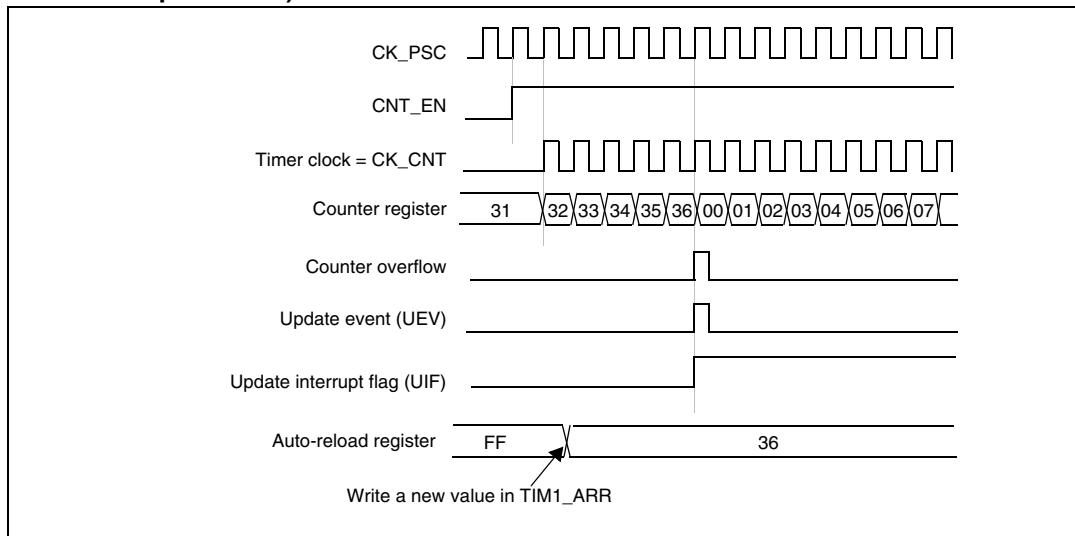
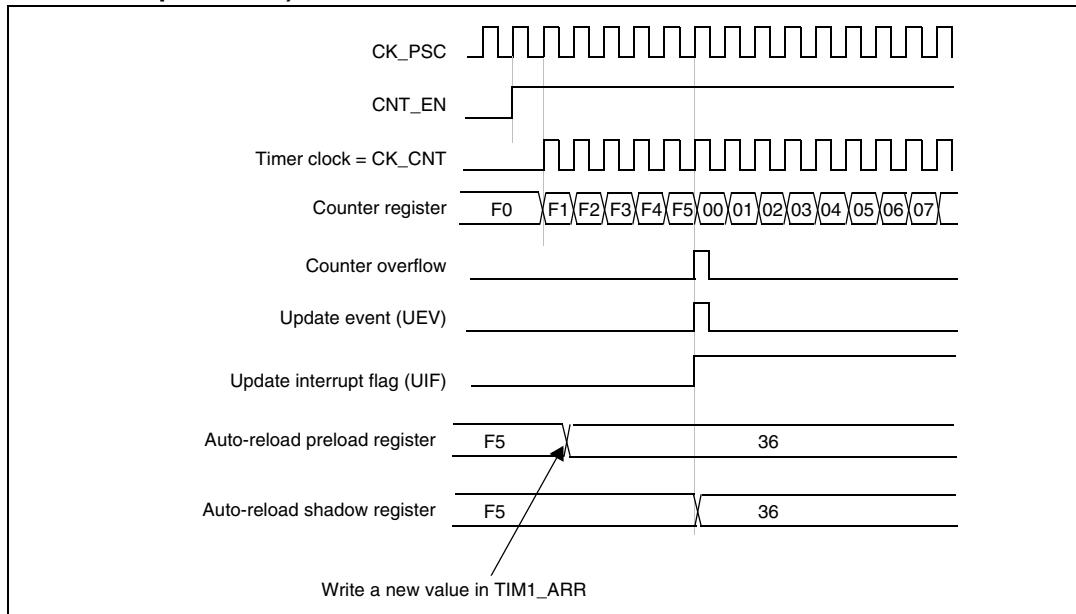
Figure 30. Counter timing diagram, internal clock divided by 4**Figure 31. Counter timing diagram, internal clock divided by N****Figure 32. Counter timing diagram, update event when ARPE=0 (TIM1_ARR not preloaded)**

Figure 33. Counter timing diagram, update event when ARPE=1 (TIM1_ARR preloaded)



Downcounting mode

In downcounting mode, the counter counts from the auto-reload value (content of the TIM1_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 (TIM1_RCR). Else the update event is generated at each counter underflow.

Setting the UG bit in the TIM1_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 TIM1_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 TIM1_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 TIM1_SR register) is set (depending on the URS bit):

- The repetition counter is reloaded with the content of TIM1_RCR register
- The auto-reload active register is updated with the preload value (content of the TIM1_ARR register). Note that the auto-reload is updated before the counter is reloaded, so that the next period is the expected one

The following figures show some examples of the counter behavior for different clock frequencies when TIM1_ARR=0x36.

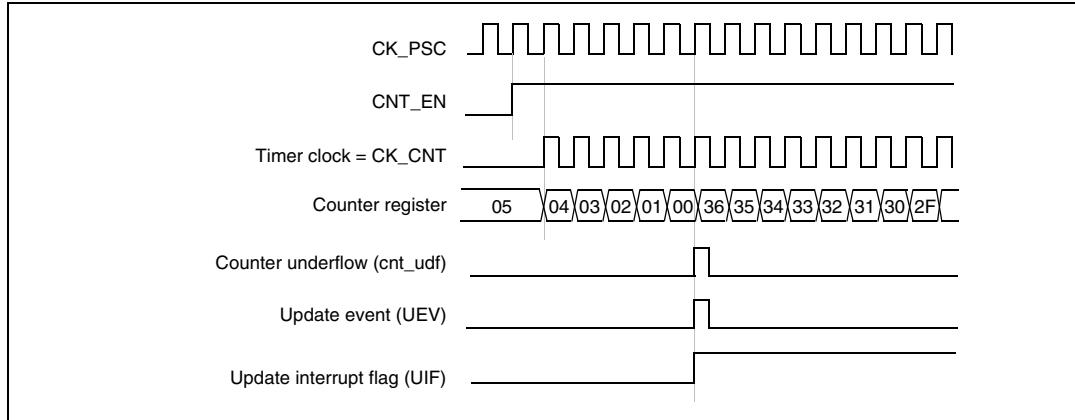
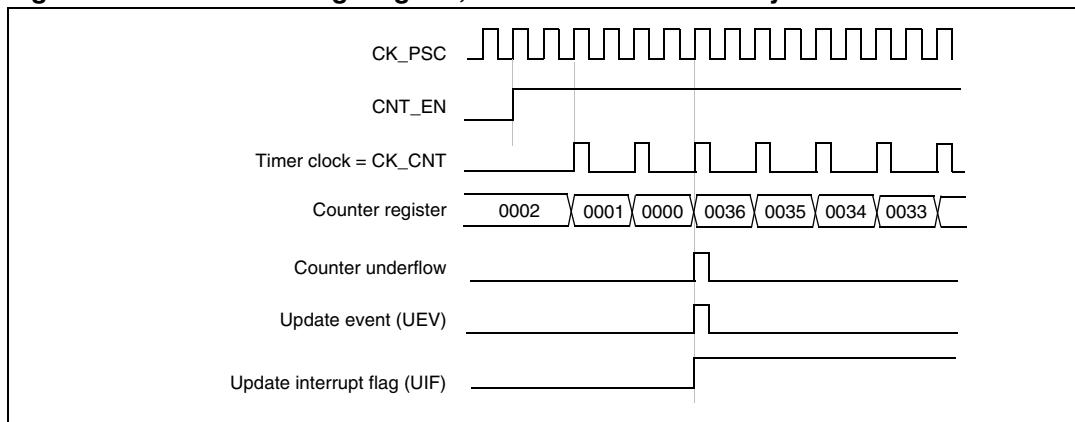
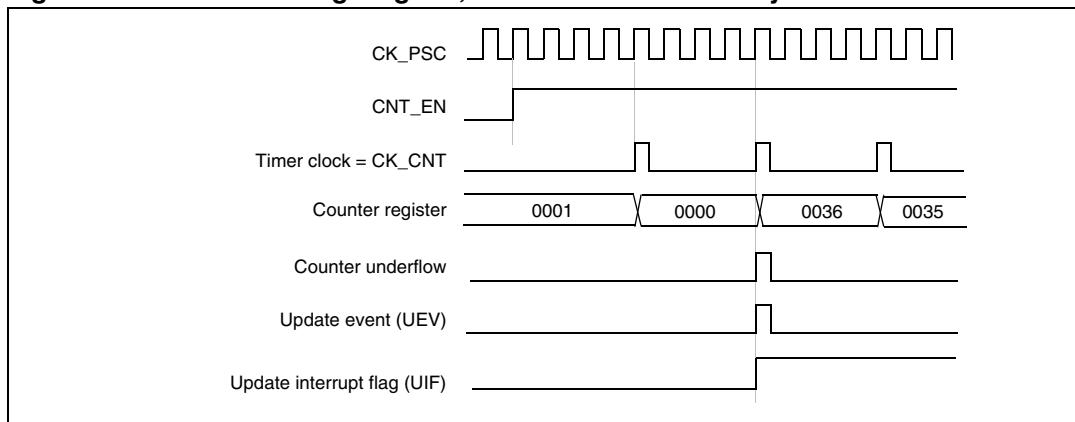
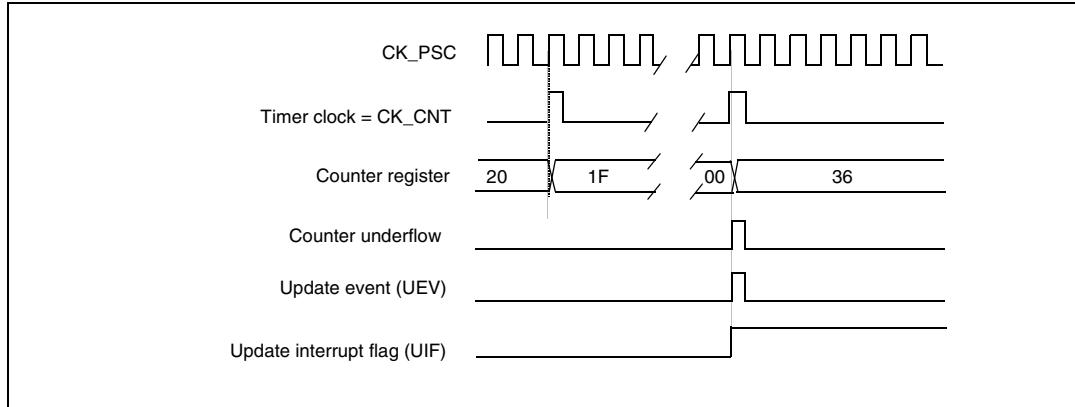
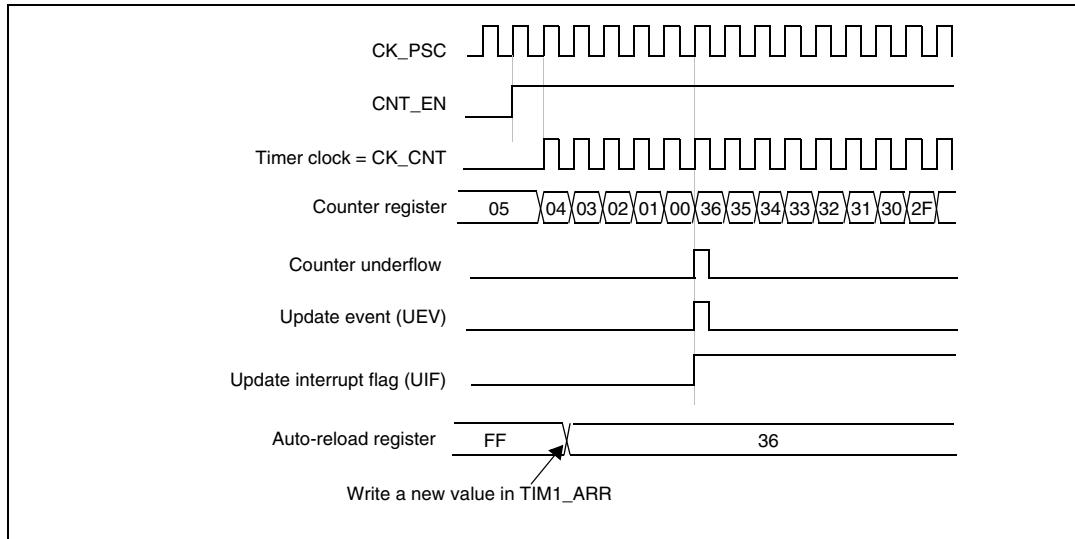
Figure 34. Counter timing diagram, internal clock divided by 1**Figure 35. Counter timing diagram, internal clock divided by 2****Figure 36. Counter timing diagram, internal clock divided by 4**

Figure 37. Counter timing diagram, internal clock divided by N**Figure 38. 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 TIM1_ARR register) – 1, generates a counter overflow event, then counts from the auto-reload value down to 1 and generates a counter underflow event. Then it restarts counting from 0.

In this mode, the DIR direction bit in the TIM1_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 TIM1_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 TIM1_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 TIM1_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 TIM1_SR register) is set (depending on the URS bit):

- The repetition counter is reloaded with the content of TIM1_RCR register
- The auto-reload active register is updated with the preload value (content of the TIM1_ARR register). Note that if the update source is a counter overflow, the auto-reload is updated before the counter is reloaded, so that the next period is the expected one (the counter is loaded with the new value).

The following figures show some examples of the counter behavior for different clock frequencies.

Figure 39. Counter timing diagram, internal clock divided by 1, TIM1_ARR=0x6

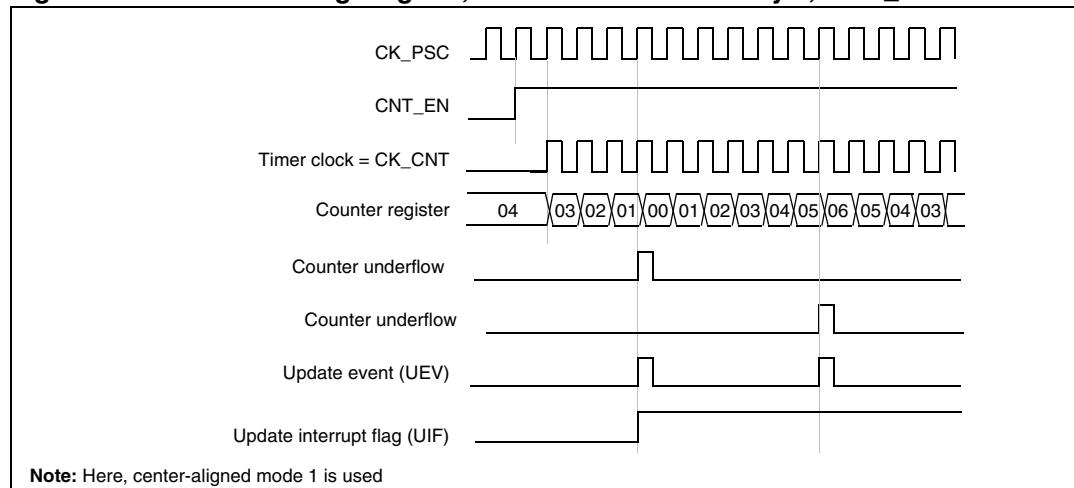


Figure 40. Counter timing diagram, internal clock divided by 2

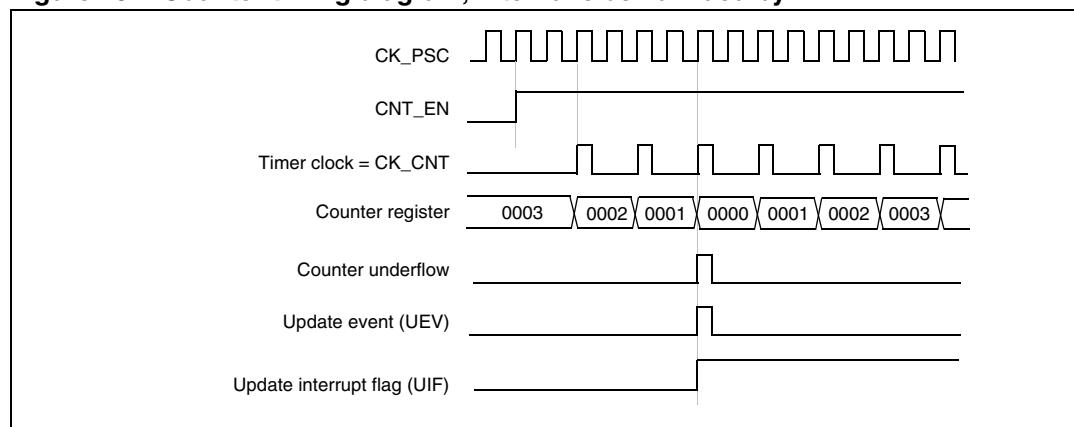


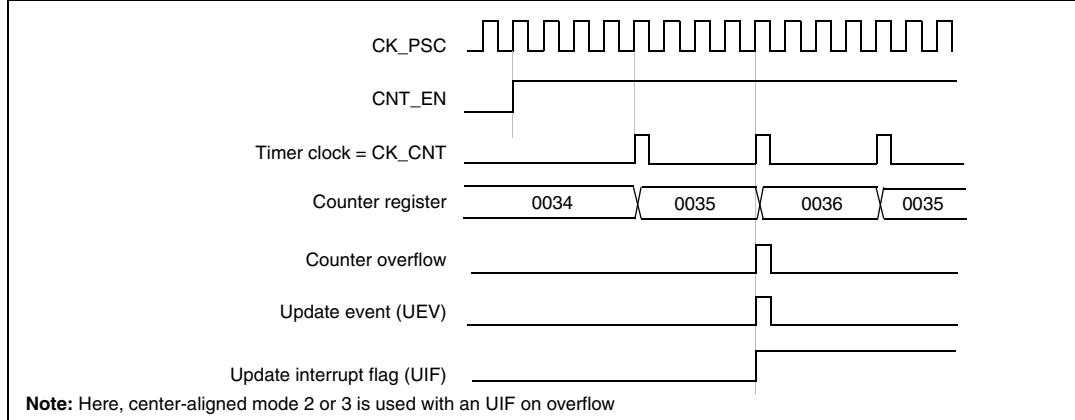
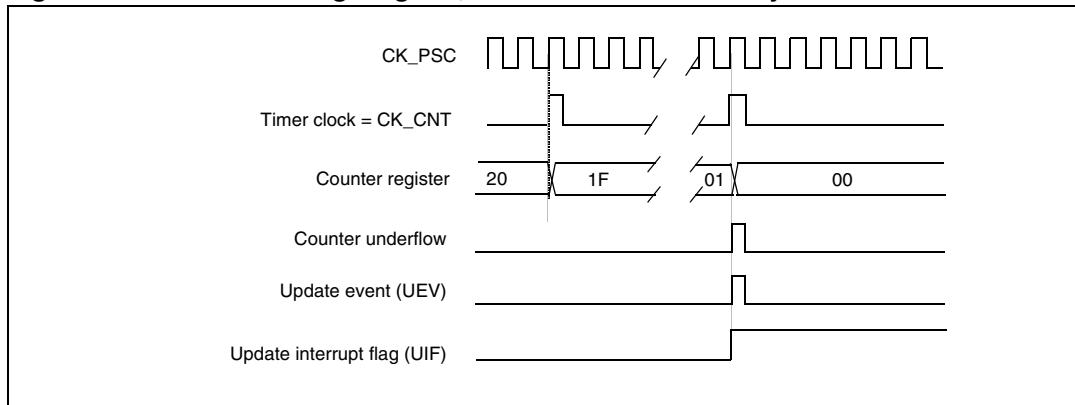
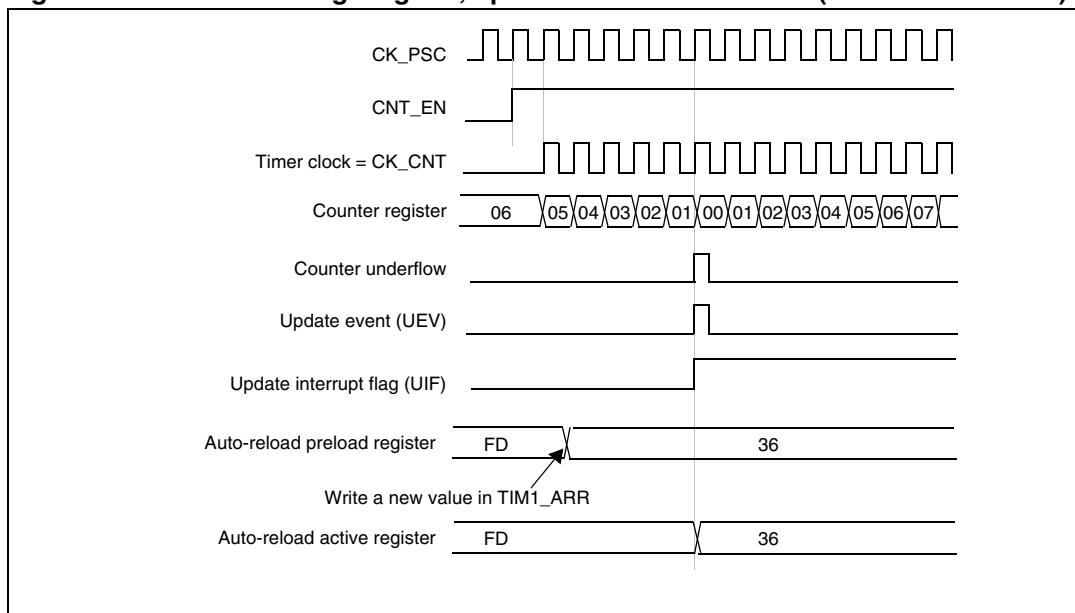
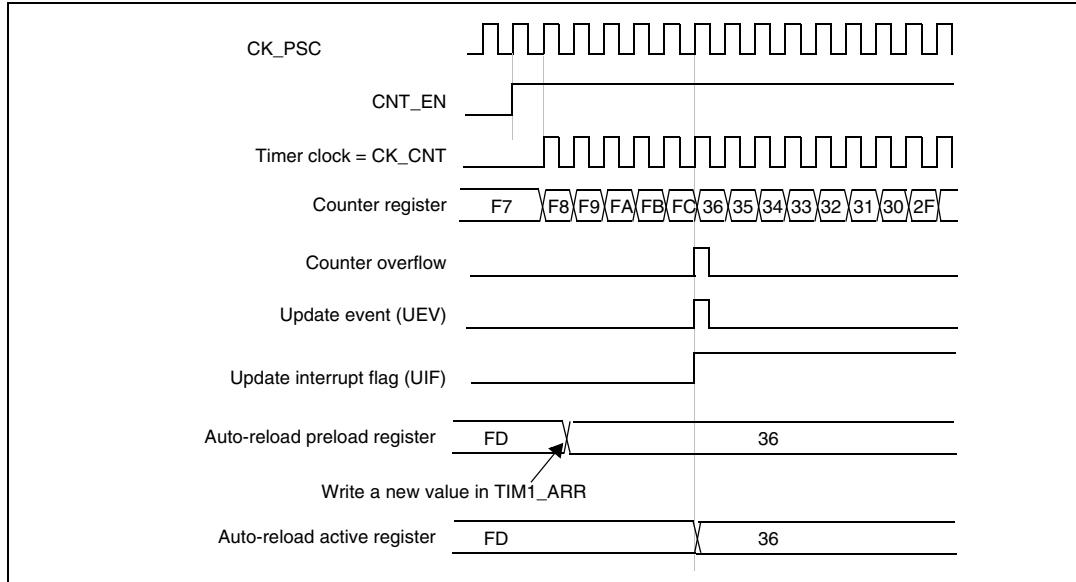
Figure 41. Counter timing diagram, internal clock divided by 4, TIM1_ARR=0x36**Figure 42. Counter timing diagram, internal clock divided by N****Figure 43. Counter timing diagram, update event with ARPE=1 (counter underflow)**

Figure 44. Counter timing diagram, Update event with ARPE=1 (counter overflow)

12.4.3 Repetition downcounter

Section 12.4.1: 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 downcounter has reached zero. This can be useful when generating PWM signals.

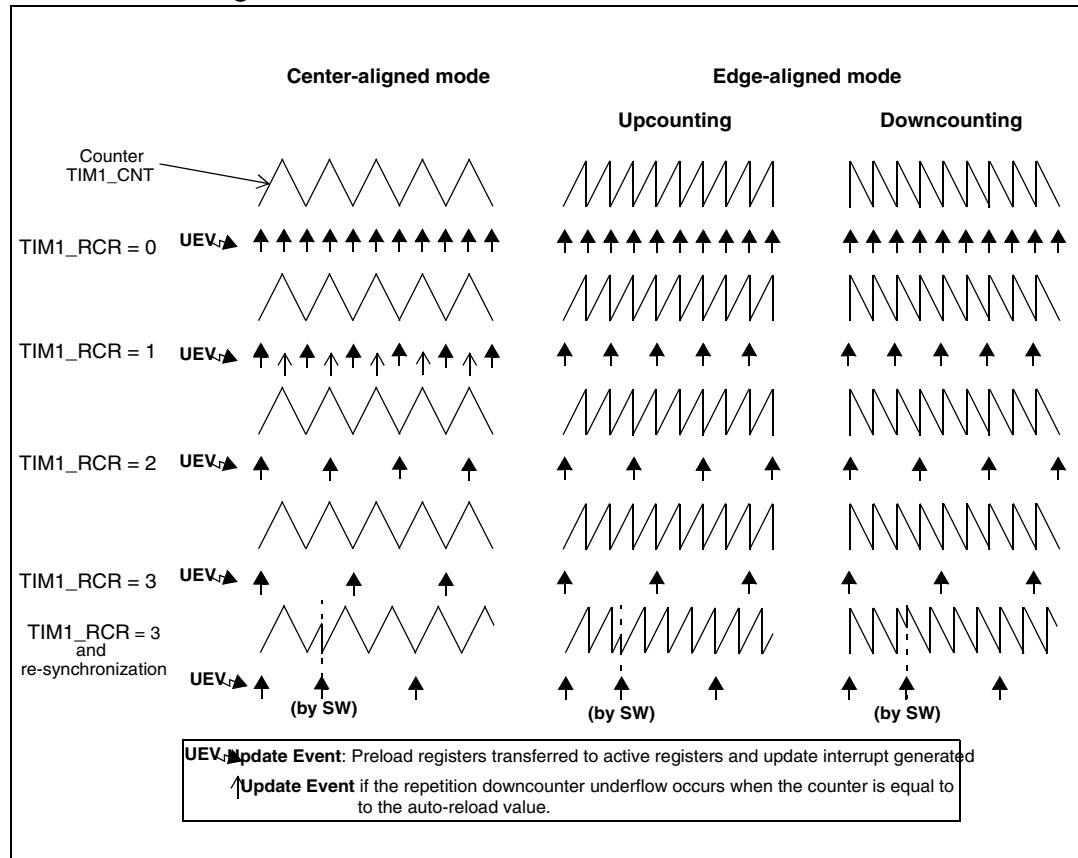
This means that data are transferred from the preload registers to the shadow registers (TIM1_ARR auto-reload register, TIM1_PSC prescaler register, but also TIM1_CCRx capture/compare registers in compare mode) every N counter overflows or underflows, where N is the value in the TIM1_RCR repetition counter register.

The repetition downcounter 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 downcounter is an auto-reload type; the repetition rate is maintained as defined by the TIM1_RCR register value (refer to [Figure 45](#)). When the update event is generated by software (by setting the UG bit in TIM1_EGR register) or by hardware through the slave mode controller, it occurs immediately whatever the value of the repetition downcounter is and the repetition downcounter is reloaded with the content of the TIM1_RCR register.

Figure 45. Update rate examples depending on mode and TIM1_RCR register settings



12.4.4 Clock selection

The counter clock can be provided by the following clock sources:

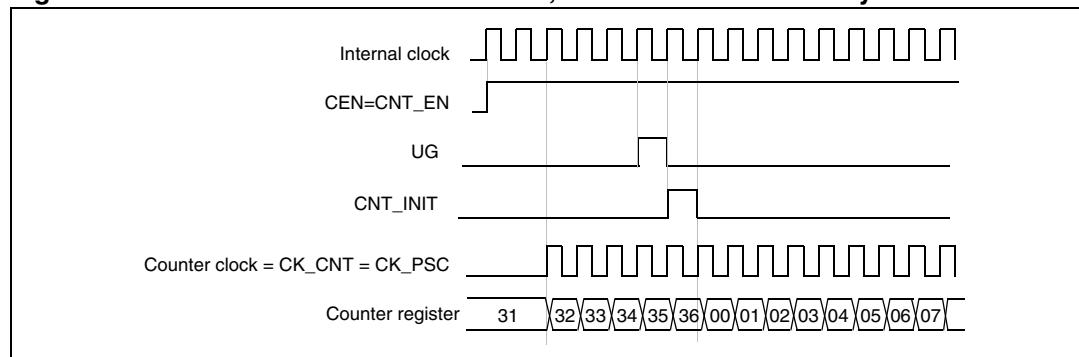
- Internal clock (CK_INT)
- External clock mode1: external input pin
- External clock mode2: external trigger input ETR
- Internal trigger inputs (ITRx): using one timer as prescaler for another timer, for example, you can configure Timer 1 to act as a prescaler for Timer 2. Refer to [Section : Using one timer as prescaler for another timer on page 250](#) for more details.

Internal clock source (CK_INT)

If the slave mode controller is disabled (SMS=000), then the CEN, DIR (in the TIM1_CR1 register) and UG bits (in the TIM1_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 46](#) shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

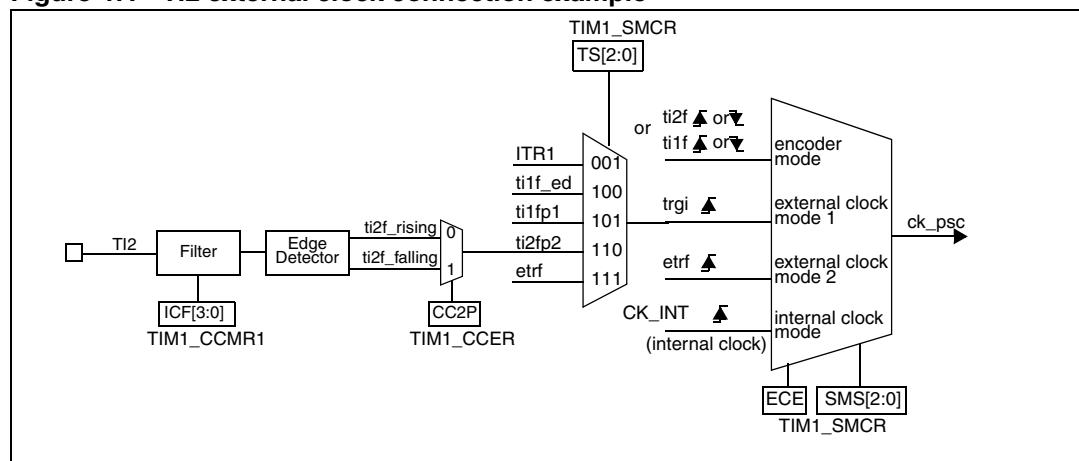
Figure 46. Control circuit in normal mode, internal clock divided by 1



External clock source mode 1

This mode is selected when SMS=111 in the TIM1_SMCR register. The counter can count at each rising or falling edge on a selected input.

Figure 47. 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:

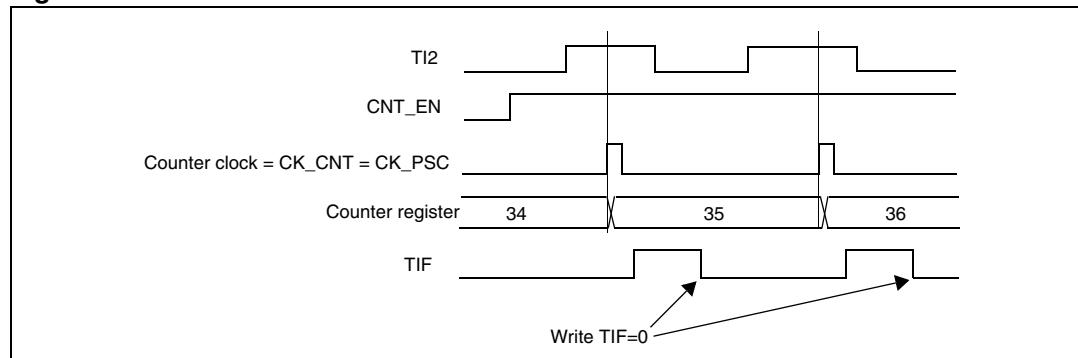
1. Configure channel 2 to detect rising edges on the TI2 input by writing CC2S = '01' in the TIM1_CCMR1 register.
2. Configure the input filter duration by writing the IC2F[3:0] bits in the TIM1_CCMR1 register (if no filter is needed, keep IC2F=0000).
3. Select rising edge polarity by writing CC2P=0 in the TIM1_CCER register.
4. Configure the timer in external clock mode 1 by writing SMS=111 in the TIM1_SMCR register.
5. Select TI2 as the trigger input source by writing TS=110 in the TIM1_SMCR register.
6. Enable the counter by writing CEN=1 in the TIM1_CR1 register.

Note: The capture prescaler is not used for triggering, so you don't need to configure it.

When a rising edge occurs on TI2, the counter counts once and the TIF flag is set.

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

Figure 48. Control circuit in external clock mode 1



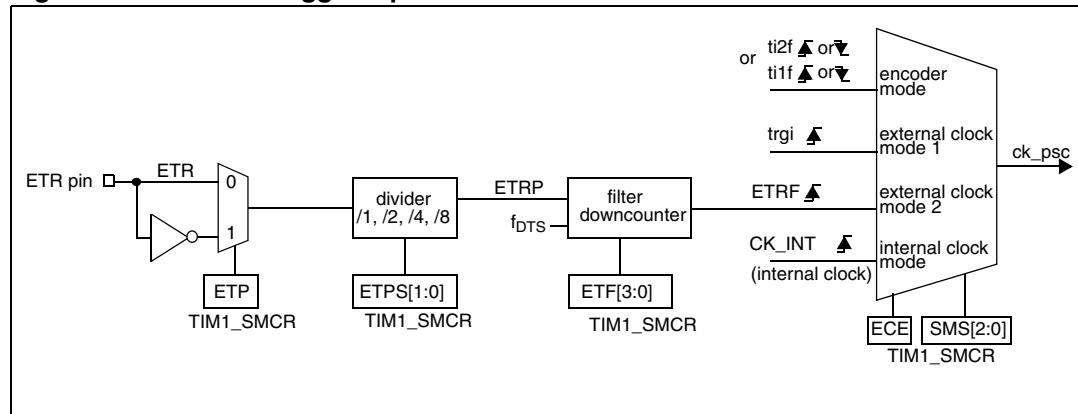
External clock source mode 2

This mode is selected by writing ECE=1 in the TIM1_SMCR register.

The counter can count at each rising or falling edge on the external trigger input ETR.

The [Figure 49](#) gives an overview of the external trigger input block.

Figure 49. External trigger input block



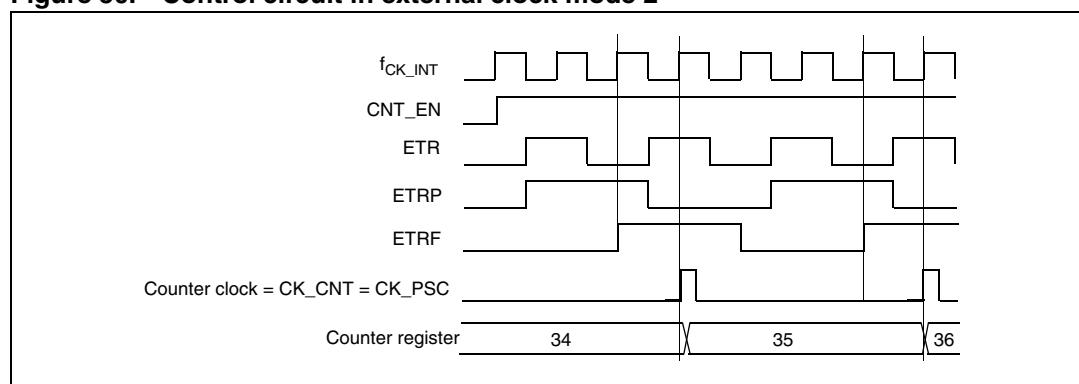
For example, to configure the upcounter to count each 2 rising edges on ETR, use the following procedure:

1. As no filter is needed in this example, write ETF[3:0]=0000 in the TIM1_SMCR register.
2. Set the prescaler by writing ETPS[1:0]=01 in the TIM1_SMCR register
3. Select rising edge detection on the ETR pin by writing ETP=0 in the TIM1_SMCR register
4. Enable external clock mode 2 by writing ECE=1 in the TIM1_SMCR register.
5. Enable the counter by writing CEN=1 in the TIM1_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 50. Control circuit in external clock mode 2

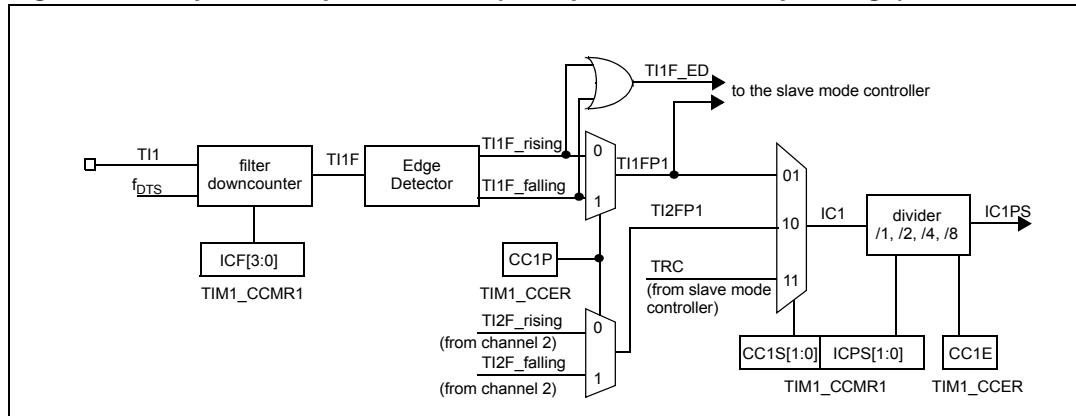


12.4.5 Capture/compare channels

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 51 to Figure 54 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 51. 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.

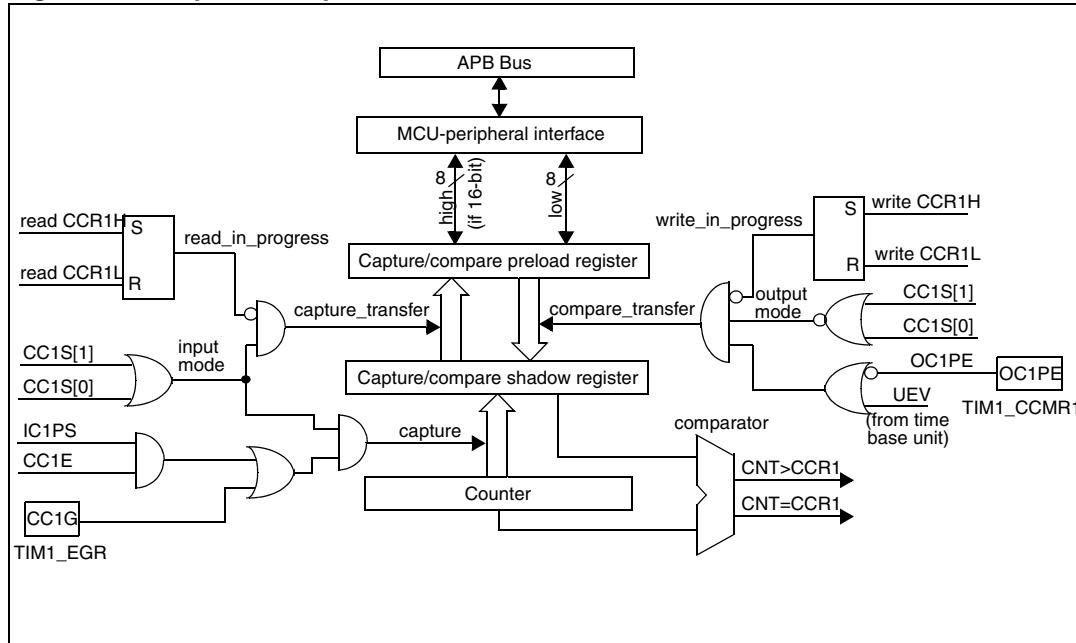
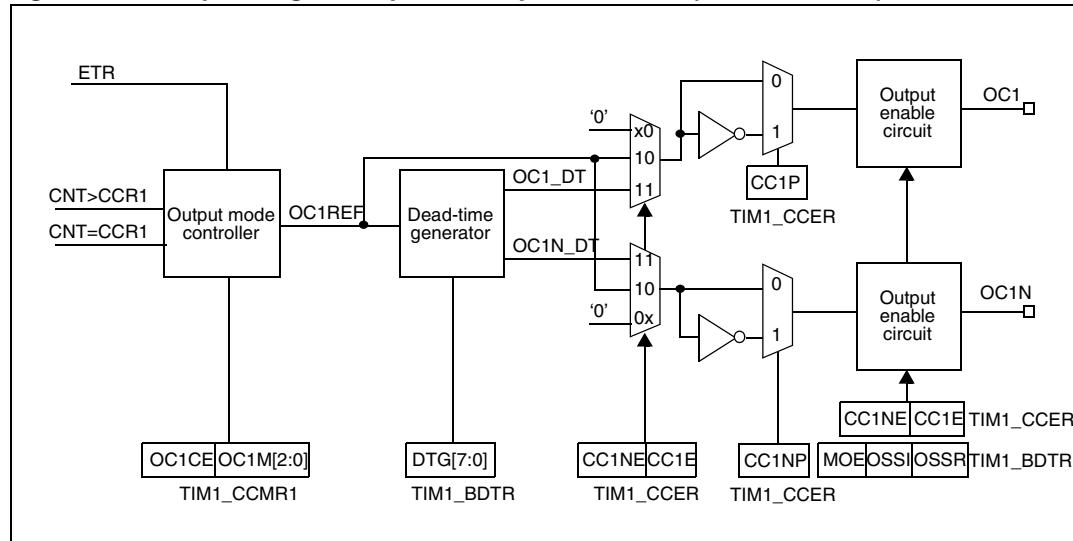
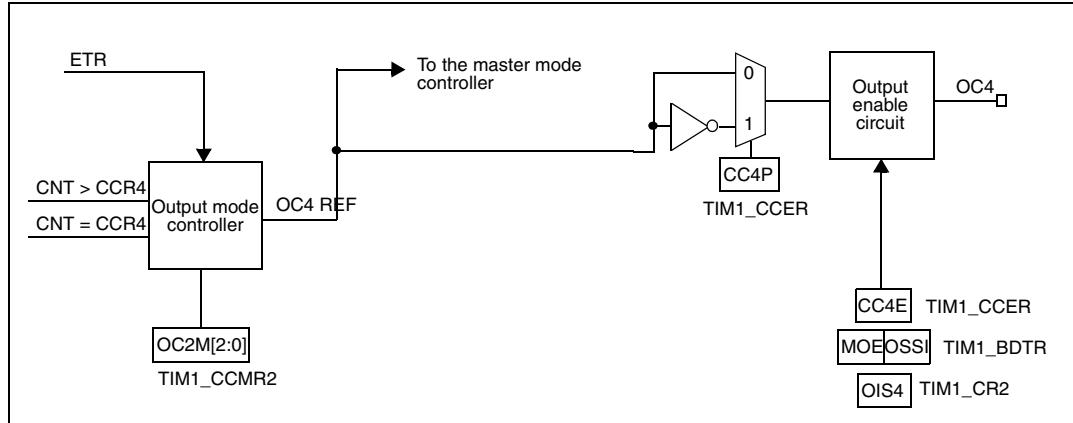
Figure 52. Capture/compare channel 1 main circuit

Figure 53. Output stage of capture/compare channel (channel 1 to 3)**Figure 54. Output stage of capture/compare channel (channel 4)**

The capture/compare block is made of one preload register and one shadow register. Write and read always access the preload register.

In capture mode, captures are actually done in the shadow register, which is copied into the preload register.

In compare mode, the content of the preload register is copied into the shadow register which is compared to the counter.

12.4.6 Input capture mode

In Input capture mode, the Capture/Compare Registers (TIM1_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 (TIM1_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 (TIM1_SR register) is set. CCxIF can be cleared by software by writing it to '0' or by reading the captured data stored in the TIM1_CCRx register. CCxOF is cleared when you write it to '0'.

The following example shows how to capture the counter value in TIM1_CCR1 when TI1 input rises. To do this, use the following procedure:

- Select the active input: TIM1_CCR1 must be linked to the TI1 input, so write the CC1S bits to 01 in the TIM1_CCMR1 register. As soon as CC1S becomes different from 00, the channel is configured in input and the TIM1_CCR1 register becomes read-only.
- Program the input filter duration you need with respect to the signal you connect to the timer (when the input is one of the TIx (ICxF bits in the TIM1_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 TIM1_CCMR1 register.
- Select the edge of the active transition on the TI1 channel by writing CC1P bit to 0 in the TIM1_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 TIM1_CCMR1 register).
- Enable capture from the counter into the capture register by setting the CC1E bit in the TIM1_CCER register.
- If needed, enable the related interrupt request by setting the CC1IE bit in the TIM1_DIER register, and/or the DMA request by setting the CC1DE bit in the TIM1_DIER register.

When an input capture occurs:

- The TIM1_CCR1 register gets the value of the counter on the active transition.
- CC1IF flag is set (interrupt flag). CC1OF is also set if at least two consecutive captures occurred whereas the flag was not cleared.
- An interrupt is generated depending on the CC1IE bit.
- A DMA request is generated depending on the CC1DE bit.

In order to handle the overcapture, it is recommended to read the data before the overcapture flag. This is to avoid missing an overcapture which could happen after reading the flag and before reading the data.

Note: IC interrupt and/or DMA requests can be generated by software by setting the corresponding CCxG bit in the TIM1_EGR register.

12.4.7 PWM input mode

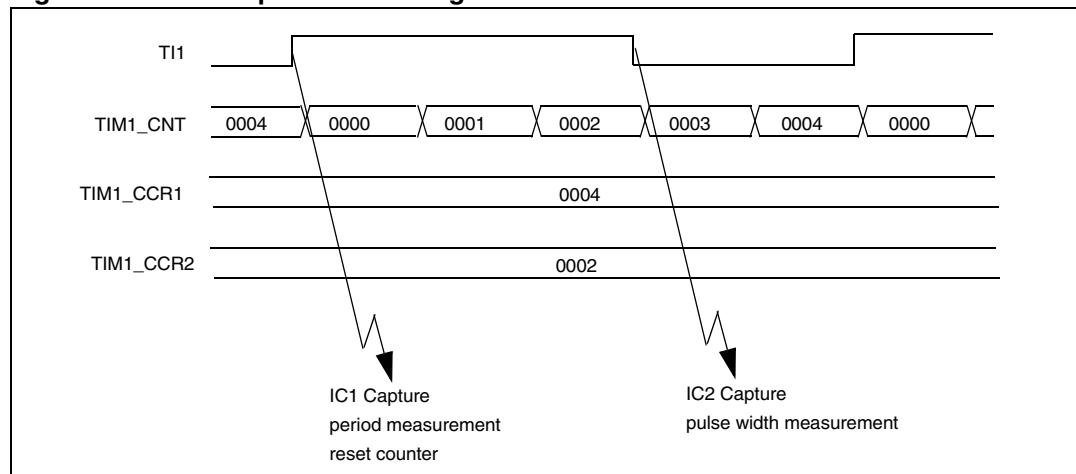
This mode is a particular case of input capture mode. The procedure is the same except:

- Two ICx signals are mapped on the same TIx input.
- These 2 ICx signals are active on edges with opposite polarity.
- One of the two TIxFP signals is selected as trigger input and the slave mode controller is configured in reset mode.

For example, you can measure the period (in TIM1_CCR1 register) and the duty cycle (in TIM1_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 TIM1_CCR1: write the CC1S bits to 01 in the TIM1_CCMR1 register (TI1 selected).
- Select the active polarity for TI1FP1 (used both for capture in TIM1_CCR1 and counter clear): write the CC1P bit to '0' (active on rising edge).
- Select the active input for TIM1_CCR2: write the CC2S bits to 10 in the TIM1_CCMR1 register (TI1 selected).
- Select the active polarity for TI1FP2 (used for capture in TIM1_CCR2): write the CC2P bit to '1' (active on falling edge).
- Select the valid trigger input: write the TS bits to 101 in the TIM1_SMCR register (TI1FP1 selected).
- Configure the slave mode controller in reset mode: write the SMS bits to 100 in the TIM1_SMCR register.
- Enable the captures: write the CC1E and CC2E bits to '1' in the TIM1_CCER register.

Figure 55. PWM input mode timing



12.4.8 Forced output mode

In output mode (CCxS bits = 00 in the TIM1_CCMRx register), each output compare signal (OCxREF and then OCx/OCxN) can be forced to active or inactive level directly by software, independently of any comparison between the output compare register and the counter.

To force an output compare signal (OCXREF/OCx) to its active level, you just need to write 101 in the OCxM bits in the corresponding TIM1_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 TIM1_CCMRx register.

Anyway, the comparison between the TIM1_CCRx shadow register and the counter is still performed and allows the flag to be set. Interrupt and DMA requests can be sent accordingly. This is described in the output compare mode section below.

12.4.9 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 TIM1_CCMRx register) and the output polarity (CCxP bit in the TIM1_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 TIM1_SR register).
- Generates an interrupt if the corresponding interrupt mask is set (CCXIE bit in the TIM1_DIER register).
- Sends a DMA request if the corresponding enable bit is set (CCxDE bit in the TIM1_DIER register, CCDS bit in the TIM1_CR2 register for the DMA request selection).

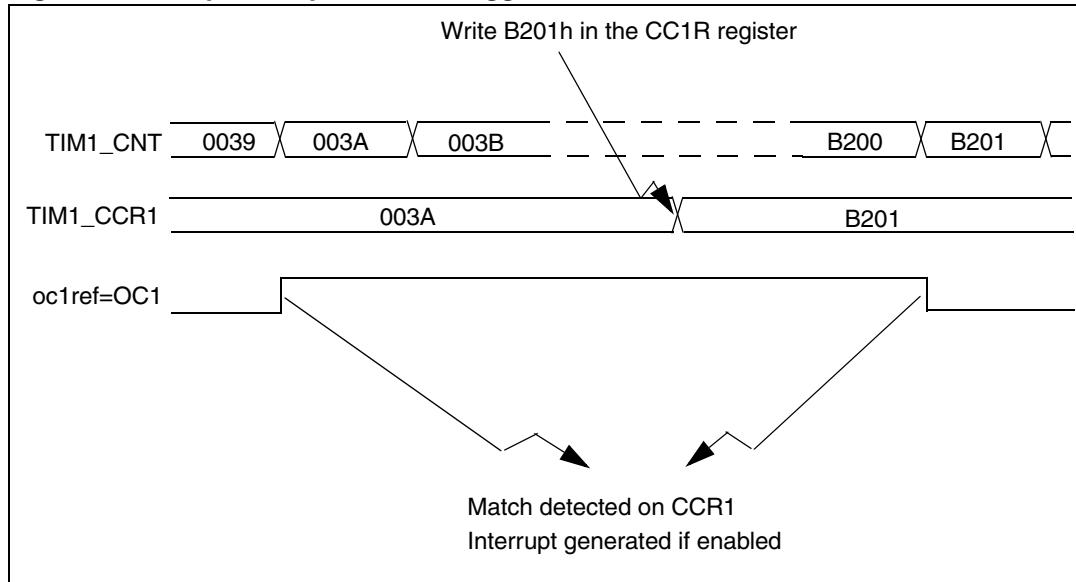
The TIM1_CCRx registers can be programmed with or without preload registers using the OCxPE bit in the TIM1_CCMRx register.

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 TIM1_ARR and TIM1_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 TIM1_CR1 register.

The TIM1_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 TIM1_CCRx shadow register is updated only at the next update event UEV). An example is given in [Figure 56](#).

Figure 56. Output compare mode, toggle on OC1.

12.4.10 PWM mode

Pulse Width Modulation mode allows you to generate a signal with a frequency determined by the value of the TIM1_ARR register and a duty cycle determined by the value of the TIM1_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 TIM1_CCMRx register. You must enable the corresponding preload register by setting the OCxPE bit in the TIM1_CCMRx register, and eventually the auto-reload preload register (in upcounting or center-aligned modes) by setting the ARPE bit in the TIM1_CR1 register.

As the preload registers are transferred to the shadow registers only when an update event occurs, before starting the counter, you have to initialize all the registers by setting the UG bit in the TIM1_EGR register.

OCx polarity is software programmable using the CCxP bit in the TIM1_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 (TIM1_CCER and TIM1_BDTR registers). Refer to the TIM1_CCER register description for more details.

In PWM mode (1 or 2), TIM1_CNT and TIM1_CCRx are always compared to determine whether $\text{TIM1_CCRx} \leq \text{TIM1_CNT}$ or $\text{TIM1_CNT} \leq \text{TIM1_CCRx}$ (depending on the direction of the counter).

The timer is able to generate PWM in edge-aligned mode or center-aligned mode depending on the CMS bits in the TIM1_CR1 register.

PWM edge-aligned mode

- Upcounting configuration

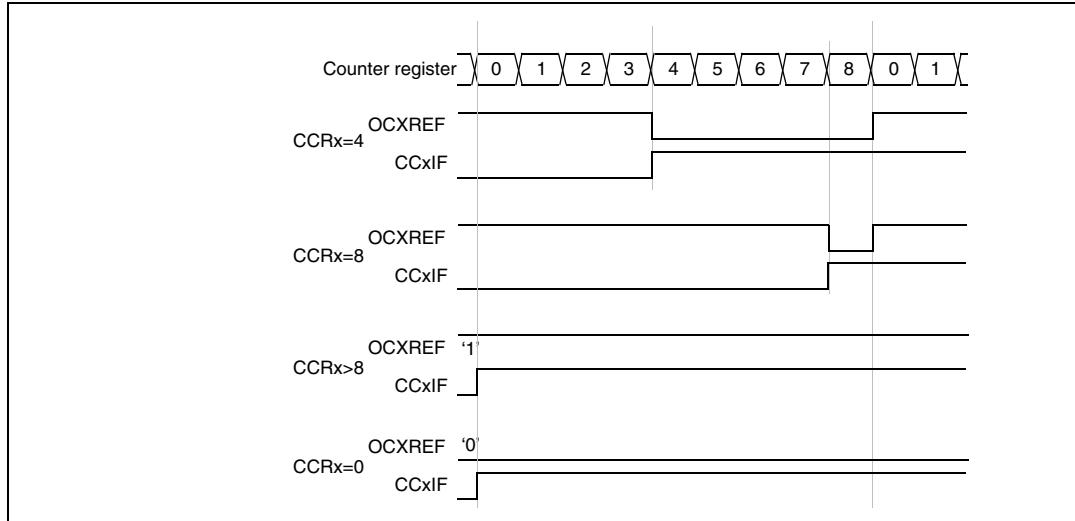
Upcounting is active when the DIR bit in the TIM1_CR1 register is low. Refer to the [Upcounting mode on page 152](#).

In the following example, we consider PWM mode 1. The reference PWM signal OCxREF is high as long as $\text{TIM1_CNT} < \text{TIM1_CCRx}$ else it becomes low. If the

compare value in TIM1_CCRx is greater than the auto-reload value (in TIM1_ARR) then OCxREF is held at '1'. If the compare value is 0 then OCxRef is held at '0'.

Figure 57 shows some edge-aligned PWM waveforms in an example where TIM1_ARR=8.

Figure 57. Edge-aligned PWM waveforms (ARR=8)



- Downcounting configuration

Downcounting is active when DIR bit in TIM1_CR1 register is high. Refer to the [Downcounting mode on page 155](#)

In PWM mode 1, the reference signal OCxRef is low as long as TIM1_CNT > TIM1_CCRx else it becomes high. If the compare value in TIM1_CCRx is greater than the auto-reload value in TIM1_ARR, then OCxREF is held at '1'. 0% PWM is not possible in this mode.

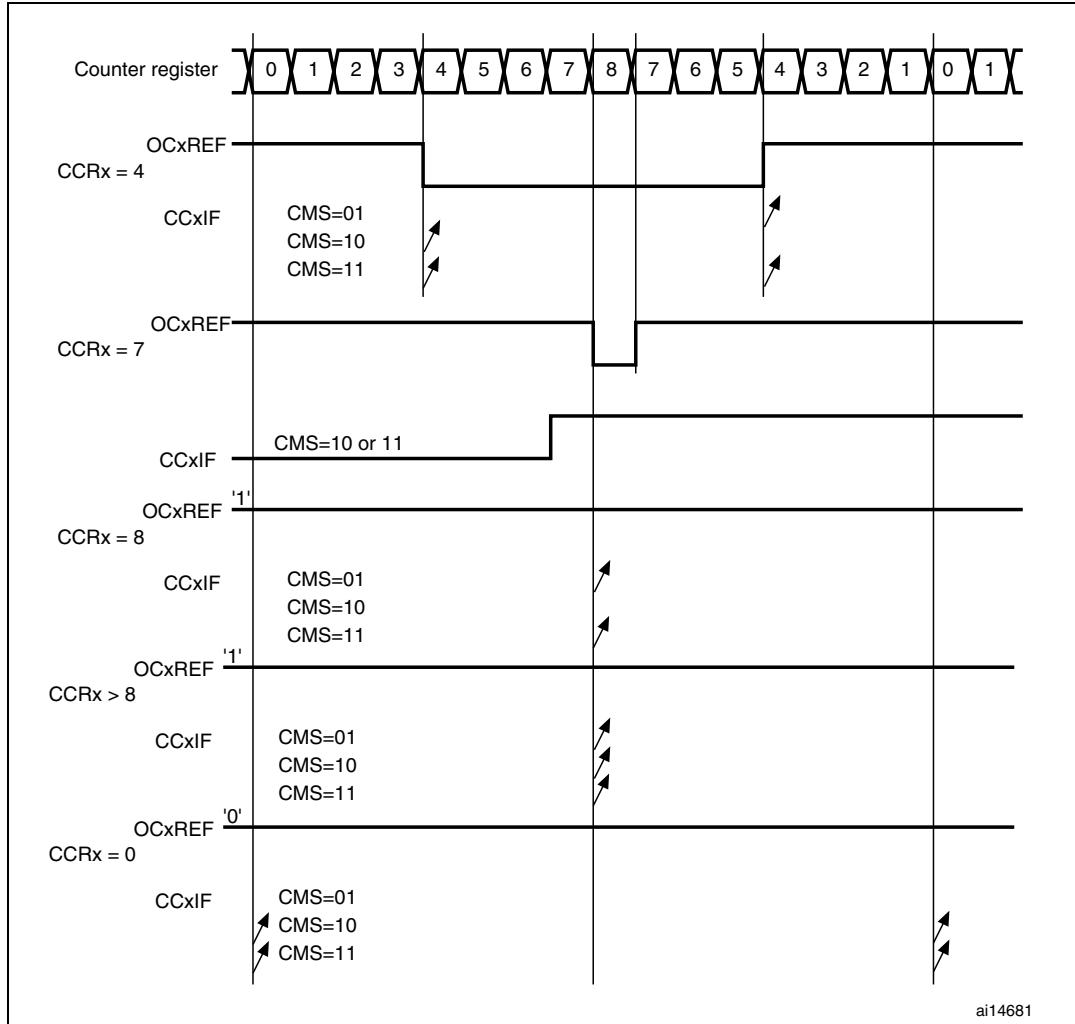
PWM center-aligned mode

Center-aligned mode is active when the CMS bits in TIM1_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 TIM1_CR1 register is updated by hardware and must not be changed by software. Refer to the [Center-aligned mode \(up/down counting\) on page 157](#).

Figure 58 shows some center-aligned PWM waveforms in an example where:

- TIM1_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 TIM1_CR1 register.

Figure 58. 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 TIM1_CR1 register. Moreover, the DIR and CMS bits must not be changed at the same time by the software.
- Writing to the counter while running in center-aligned mode is not recommended as it can lead to unexpected results. In particular:
 - The direction is not updated if you write a value in the counter that is greater than the auto-reload value (TIM1_CNT>TIM1_ARR). For example, if the counter was counting up, it continues to count up.
 - The direction is updated if you write 0 or write the TIM1_ARR value in the counter but no Update Event UEV is generated.
- The safest way to use center-aligned mode is to generate an update by software (setting the UG bit in the TIM1_EGR register) just before starting the counter and not to write the counter while it is running.

12.4.11 Complementary outputs and dead-time insertion

The Advanced Control Timer 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 you have to adjust it depending on the devices you have connected to the outputs and their characteristics (intrinsic delays of level-shifters, delays due to power switches...).

You can select the polarity of the outputs (main output OC_x or complementary OC_{xN}) independently for each output. This is done by writing to the CC_{xP} and CC_{xNP} bits in the TIM1_CCER register.

The complementary signals OC_x and OC_{xN} are activated by a combination of several control bits: the CC_{xE} and CC_{xNE} bits in the TIM1_CCER register and the MOE, OIS_x, OIS_{xN}, OSSI and OSSR bits in the TIM1_BDTR and TIM1_CR2 registers. Refer to [Table 38: Output control bits for complementary OC_x and OC_{xN} channels with break feature on page 206](#) 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_{xE} and CC_{xNE} 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_{xREF}, it generates 2 outputs OC_x and OC_{xN}. If OC_x and OC_{xN} are active high:

- The OC_x 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_{xN} 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_x or OC_{xN}) 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_{xREF}. (we suppose CC_{xP}=0, CC_{xNP}=0, MOE=1, CC_{xE}=1 and CC_{xNE}=1 in these examples)

Figure 59. Complementary output with dead-time insertion.

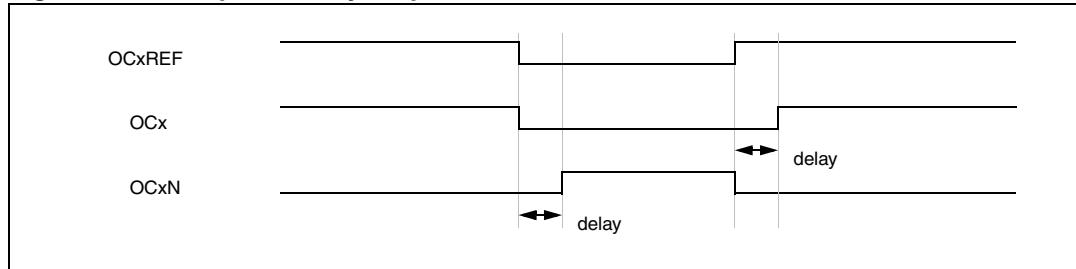


Figure 60. Dead-time waveforms with delay greater than the negative pulse.

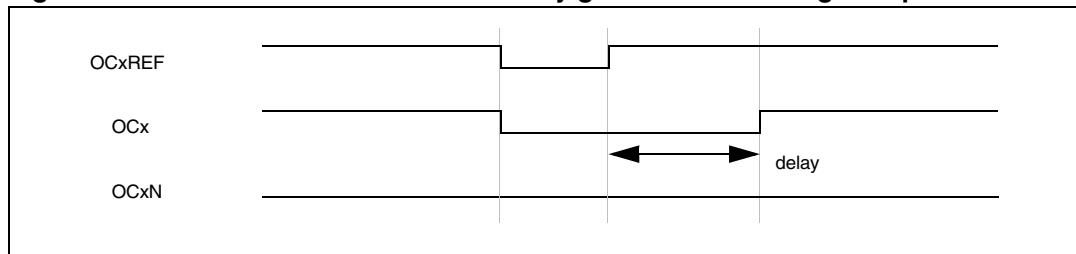
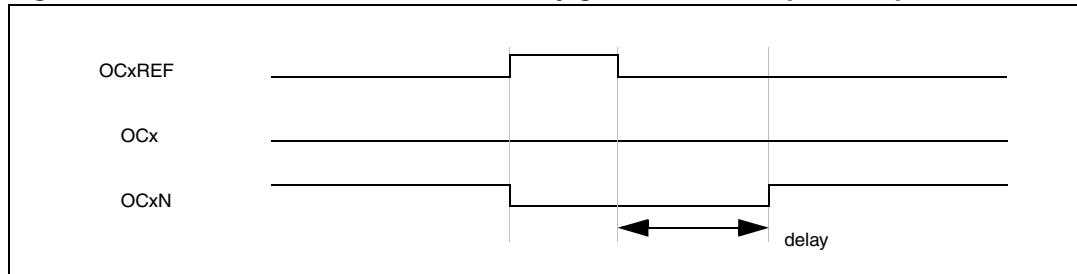


Figure 61. Dead-time waveforms with delay greater than the positive pulse.

The dead-time delay is the same for each of the channels and is programmable with the DTG bits in the TIM1_BDTR register. Refer to [Section 12.5.18: Break and dead-time register \(TIM1_BDTR\) on page 210](#) 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 TIM1_CCER register.

This allows you to send a specific waveform (such as PWM or static active level) on one output while the complementary remains at its inactive level. Other alternative possibilities are to have both outputs at inactive level or both outputs active and complementary with dead-time.

Note:

When only OCxN is enabled (CCxE=0, CCxNE=1), it is not complemented and becomes active as soon as OCxREF is high. For example, if CCxNP=0 then OCxN=OCxRef. On the other hand, when both OCx and OCxN are enabled (CCxE=CCxNE=1) OCx becomes active when OCxREF is high whereas OCxN is complemented and becomes active when OCxREF is low.

12.4.12 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 TIM1_BDTR register, OISx and OISxN bits in the TIM1_CR2 register). In any case, the OCx and OCxN outputs cannot be set both to active level at a given time. Refer to [Table 38: Output control bits for complementary OCx and OCxN channels with break feature on page 206](#) for more details.

The break source can be either the break input pin or a clock failure event, generated by the Clock Security System (CSS), from the Reset Clock Controller. For further information on the Clock Security System, refer to [Section 4.2.7: Clock security system \(CSS\) on page 52](#).

When exiting from reset, the break circuit is disabled and the MOE bit is low. You can enable the break function by setting the BKE bit in the TIM1_BDTR register. The break input polarity can be selected by configuring the BKP bit in the same register. BKE and BKP can be modified at the same time.

Because MOE falling edge can be asynchronous, a resynchronization circuit has been inserted between the actual signal (acting on the outputs) and the synchronous control bit (accessed in the TIM1_BDTR register). It results in some delays between the asynchronous and the synchronous signals. In particular, if you write MOE to 1 whereas it was low, you must insert a delay (dummy instruction) before reading it correctly. This is because you write the asynchronous signal and read the synchronous signal.

When a break occurs (selected level on the break input):

- The MOE bit is cleared asynchronously, putting the outputs in inactive state, idle state or in reset state (selected by the 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 TIM1_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 TIM1_SR register) is set. An interrupt can be generated if the BIE bit in the TIM1_DIER register is set. A DMA request can be sent if the BDE bit in the TIM1_DIER register is set.
- If the AOE bit in the TIM1_BDTR register is set, the MOE bit is automatically set again at the next update event UEV. This can be used to perform a regulation, for instance. Else, MOE remains low until you write it to '1' again. In this case, it can be used for security and you can connect the break input to an alarm from power drivers, thermal sensors or any security components.

Note:

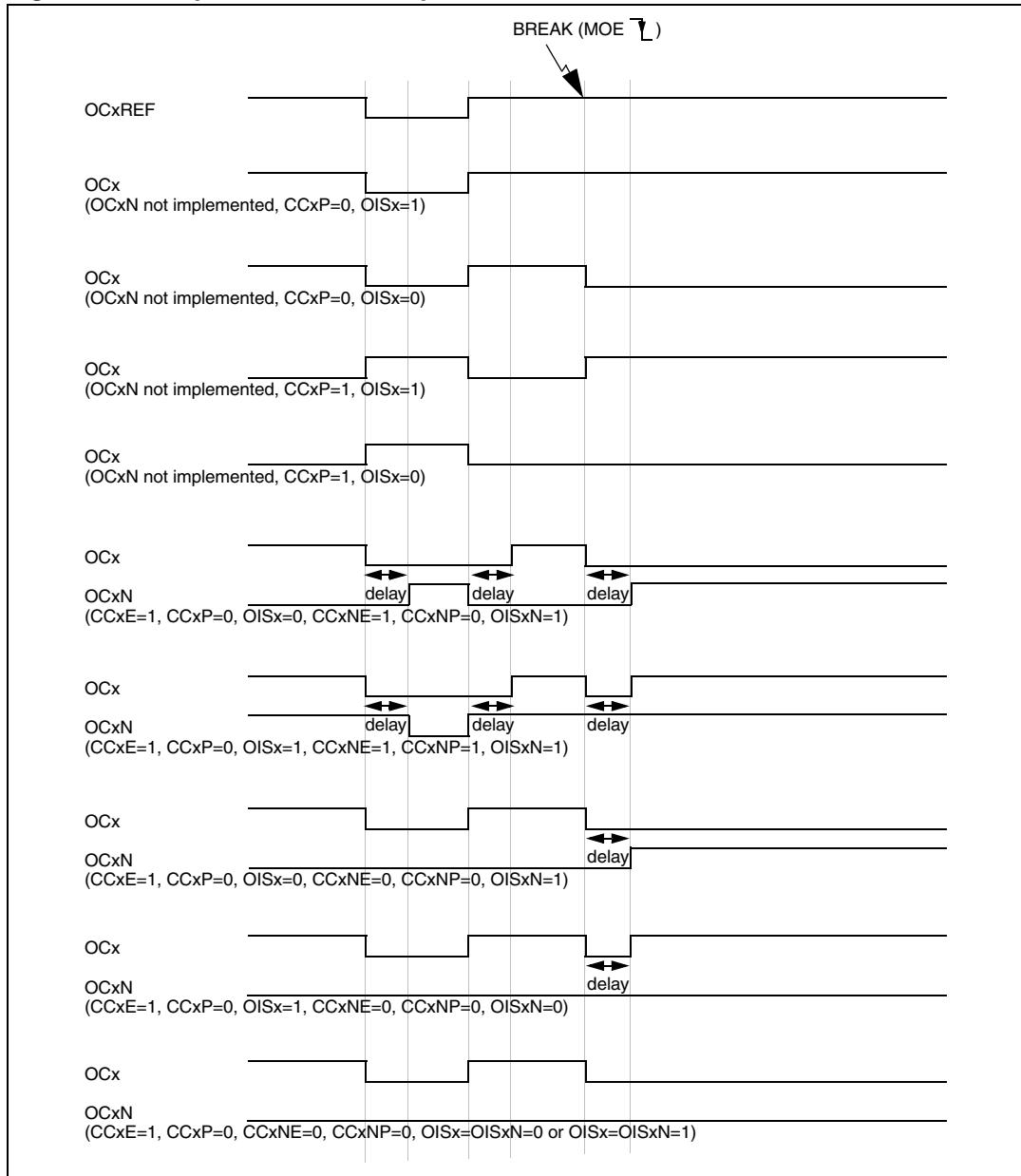
The break inputs is acting on level. Thus, the MOE cannot be set while the break input is active (neither automatically nor by software). In the meantime, the status flag BIF cannot be cleared.

The break can be generated by the BRK input which has a programmable polarity and an enable bit BKE in the TIM1_BDTR Register.

In addition to the break input and the output management, a write protection has been implemented inside the break circuit to safeguard the application. It allows you to freeze the configuration of several parameters (dead-time duration, OCx/OCxN polarities and state when disabled, OCxM configurations, break enable and polarity). You can choose from 3 levels of protection selected by the LOCK bits in the TIM1_BDTR register. Refer to [Section 12.5.18: Break and dead-time register \(TIM1_BDTR\) on page 210](#). The LOCK bits can be written only once after an MCU reset.

The [Figure 62](#) shows an example of behavior of the outputs in response to a break.

Figure 62. Output behavior in response to a break.



12.4.13 Clearing the OCxREF signal on an external event

The OC_xREF signal for a given channel can be driven Low by applying a High level to the ETRF input (OC_xCE enable bit of the corresponding TIM1_CCMRx register set to '1'). The OC_xREF signal remains Low until the next update event, UEV, occurs.

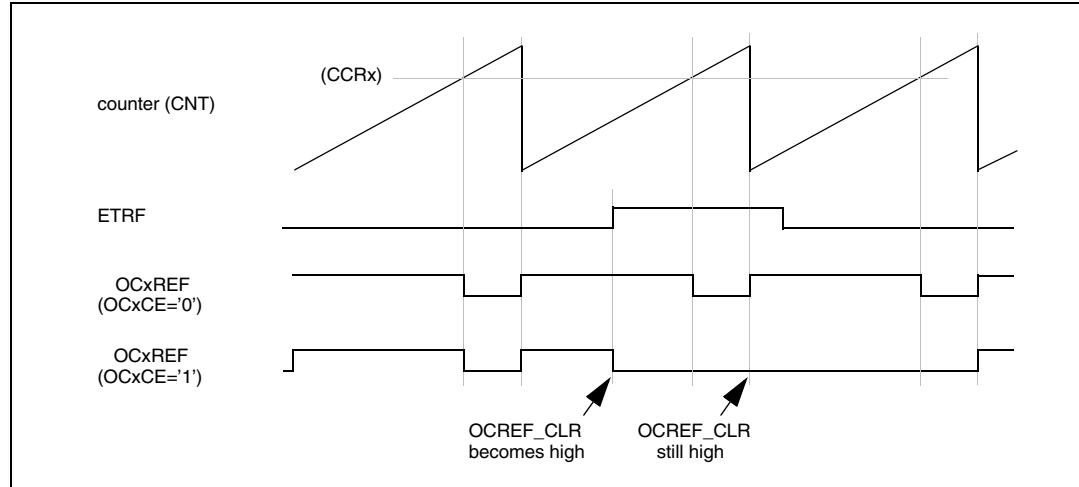
This function can only be used in output compare and PWM modes, and does not work in forced mode.

For example, the OC_xREF 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 TIM1_SMCR register set to '00'.
2. The external clock mode 2 must be disabled: bit ECE of the TIM1_SMCR register set to '0'.
3. The External Trigger Polarity (ETP) and the External Trigger Filter (ETF) can be configured according to the user needs.

Figure 63 shows the behavior of the OC_xREF signal when the ETRF Input becomes High, for both values of the enable bit OC_xCE. In this example, the timer TIM1 is programmed in PWM mode.

Figure 63. Clearing TIM1 OC_xREF



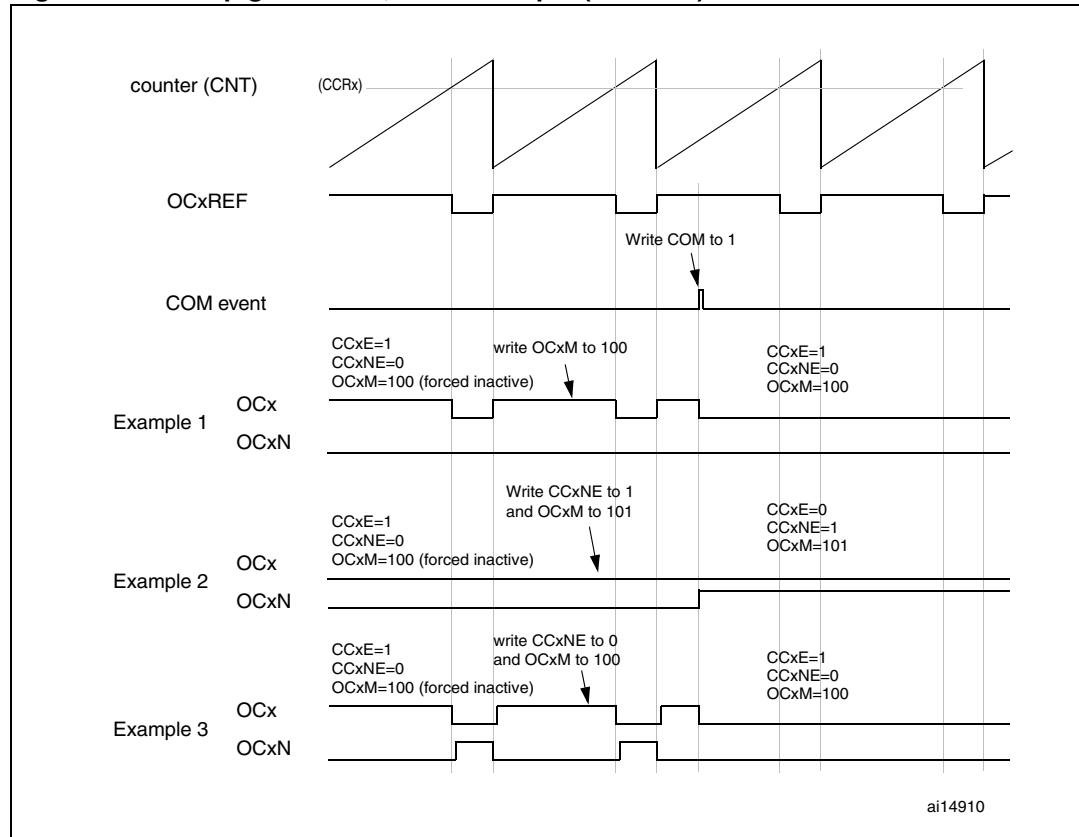
12.4.14 6-step PWM generation

When complementary outputs are used on a channel, preload bits are available on the OCxM, CCxE and CCxNE bits. The preload bits are transferred to the shadow bits at the COM commutation event. Thus you can program in advance the configuration for the next step and change the configuration of all the channels at the same time. COM can be generated by software by setting the COM bit in the TIM1_EGR register or by hardware (on TRGI rising edge).

A flag is set when the COM event occurs (COMIF bit in the TIM1_SR register), which can generate an interrupt (if the COMIE bit is set in the TIM1_DIER register) or a DMA request (if the COMDE bit is set in the TIM1_DIER register).

The [Figure 64](#) describes the behavior of the OCx and OCxN outputs when a COM event occurs, in 3 different examples of programmed configurations.

Figure 64. 6-step generation, COM example (OSSR=1)



12.4.15 One-pulse mode

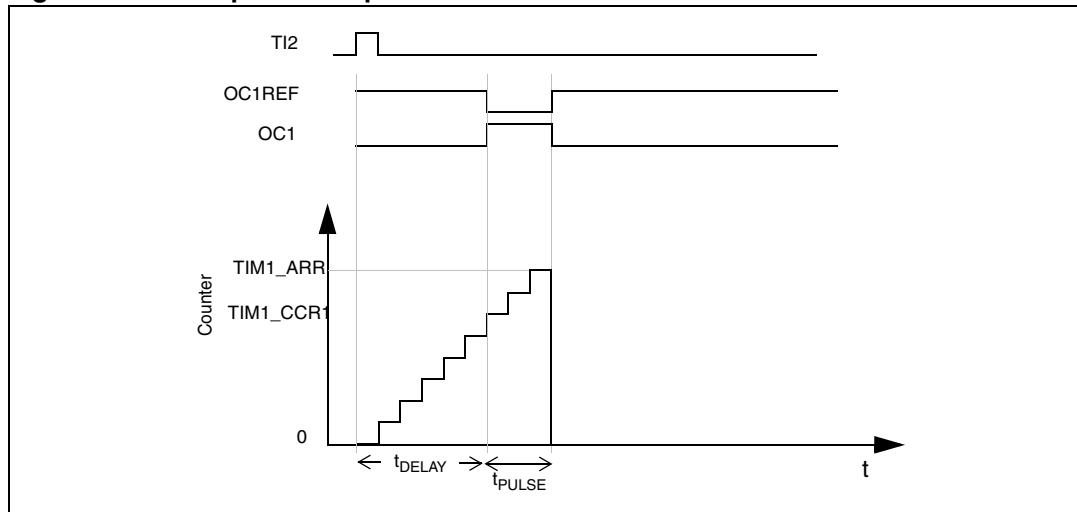
One-pulse mode (OPM) is a particular case of the previous modes. It allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length after a programmable delay.

Starting the counter can be controlled through the slave mode controller. Generating the waveform can be done in output compare mode or PWM mode. You select One-pulse mode by setting the OPM bit in the TIM1_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 65. Example of one pulse mode.



For example you may want to generate a positive pulse on OC1 with a length of t_{PULSE} and after a delay of t_{DELAY} as soon as a positive edge is detected on the TI2 input pin.

Let's use TI2FP2 as trigger 1:

- Map TI2FP2 to TI2 by writing CC2S='01' in the TIM1_CCMR1 register.
- TI2FP2 must detect a rising edge, write CC2P='0' in the TIM1_CCER register.
- Configure TI2FP2 as trigger for the slave mode controller (TRGI) by writing TS='110' in the TIM1_SMCR register.
- TI2FP2 is used to start the counter by writing SMS to '110' in the TIM1_SMCR register (trigger mode).

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

- The t_{DELAY} is defined by the value written in the TIM1_CCR1 register.
- The t_{PULSE} is defined by the difference between the auto-reload value and the compare value ($TIM1_ARR - TIM1_CCR1$).
- Let's say you want to build a waveform with a transition from '0' to '1' when a compare match occurs and a transition from '1' to '0' when the counter reaches the auto-reload

value. To do this you enable PWM mode 2 by writing OC1M=111 in the TIM1_CCMR1 register. You can optionally enable the preload registers by writing OC1PE='1' in the TIM1_CCMR1 register and ARPE in the TIM1_CR1 register. In this case you have to write the compare value in the TIM1_CCR1 register, the auto-reload value in the TIM1_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 TIM1_CR1 register should be low.

You only want 1 pulse, so you write '1' in the OPM bit in the TIM1_CR1 register to stop the counter at the next update event (when the counter rolls over from the auto-reload value back to 0).

Particular case: OCx fast enable:

In One-pulse mode, the edge detection on TIx input set the CEN bit which enables the counter. Then the comparison between the counter and the compare value makes the output toggle. But several clock cycles are needed for these operations and it limits the minimum delay t_{DELAY} min we can get.

If you want to output a waveform with the minimum delay, you can set the OCxFE bit in the TIM1_CCMRx register. Then OCxRef (and OCx) are forced in response to the stimulus, without taking in account the comparison. Its new level is the same as if a compare match had occurred. OCxFE acts only if the channel is configured in PWM1 or PWM2 mode.

12.4.16 Encoder interface mode

To select Encoder Interface mode write SMS='001' in the TIM1_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 TIM1_CCER register. When needed, you can program the input filter as well.

The two inputs TI1 and TI2 are used to interface to an incremental encoder. Refer to [Table 37](#). 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 TIM1_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 TIM1_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 TIM1_ARR register (0 to ARR or ARR down to 0 depending on the direction). So you must configure TIM1_ARR before starting. in the same way, the capture, compare, prescaler, repetition counter, trigger output features continue to work as normal. Encoder mode and External clock mode 2 are not compatible and must not be selected together.

In this mode, the counter is modified automatically following the speed and the direction of the incremental encoder and its content, therefore, always represents the encoder's position. The count direction correspond to the rotation direction of the connected sensor. The table summarizes the possible combinations, assuming TI1 and TI2 don't switch at the same time.

Table 37. Counting direction versus encoder signals

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

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

The [Figure 66](#) 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' (TIM1_CCMR1 register, TI1FP1 mapped on TI1).
- CC2S='01' (TIM1_CCMR2 register, TI1FP2 mapped on TI2).
- CC1P='0' (TIM1_CCER register, TI1FP1 non-inverted, TI1FP1=TI1).
- CC2P='0' (TIM1_CCER register, TI1FP2 non-inverted, TI1FP2= TI2).
- SMS='011' (TIM1_SMCR register, both inputs are active on both rising and falling edges).
- CEN='1' (TIM1_CR1 register, Counter enabled).

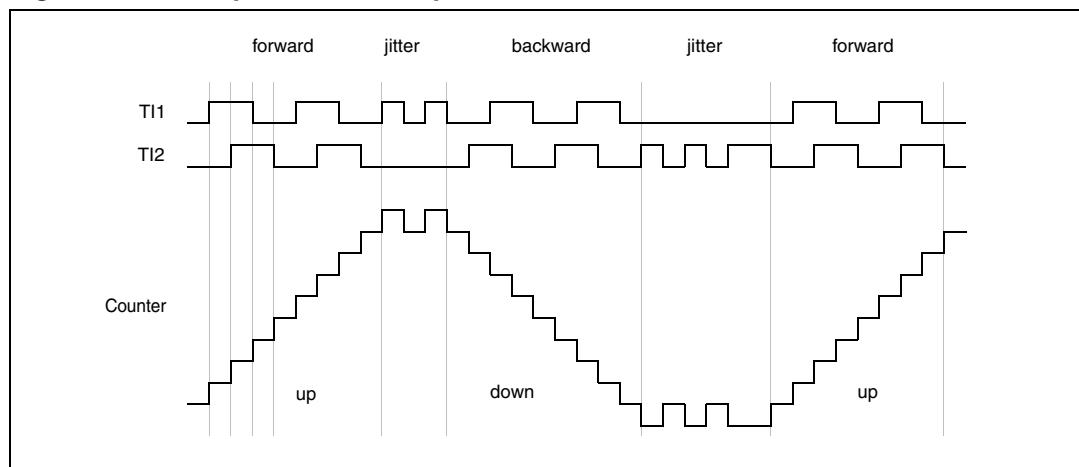
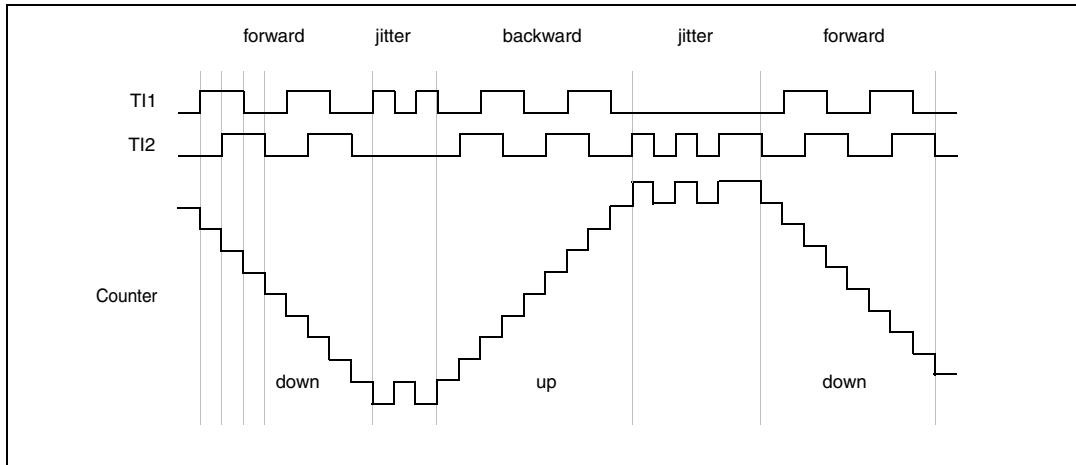
Figure 66. Example of counter operation in encoder interface mode.

Figure 67 gives an example of counter behavior when TI1FP1 polarity is inverted (same configuration as above except CC1P='1').

Figure 67. Example of encoder interface mode with TI1FP1 polarity inverted.



The timer, when configured in Encoder Interface mode provides information on the sensor's current position. You can obtain dynamic information (speed, acceleration, deceleration) by measuring the period between two encoder events using a second timer configured in capture mode. The output of the encoder which indicates the mechanical zero can be used for this purpose. Depending on the time between two events, the counter can also be read at regular times. You can do this by latching the counter value into a third input capture register if available (then the capture signal must be periodic and can be generated by another timer). when available, it is also possible to read its value through a DMA request generated by a Real-Time clock.

12.4.17 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 TIM1_CH1, TIM1_CH2 and TIM1_CH3.

The XOR output can be used with all the timer input functions such as trigger or input capture. An example of this feature used to interface Hall sensors is given in [Section 12.4.18](#) below.

12.4.18 Interfacing with Hall sensors

This is done using the advanced control timer TIM1 to generate PWM signals to drive the motor and another timer TIMx referred to as “interfacing timer” in *Figure 68*. 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 51: Capture/compare channel \(example: channel 1 input stage\) on page 165](#)). 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.

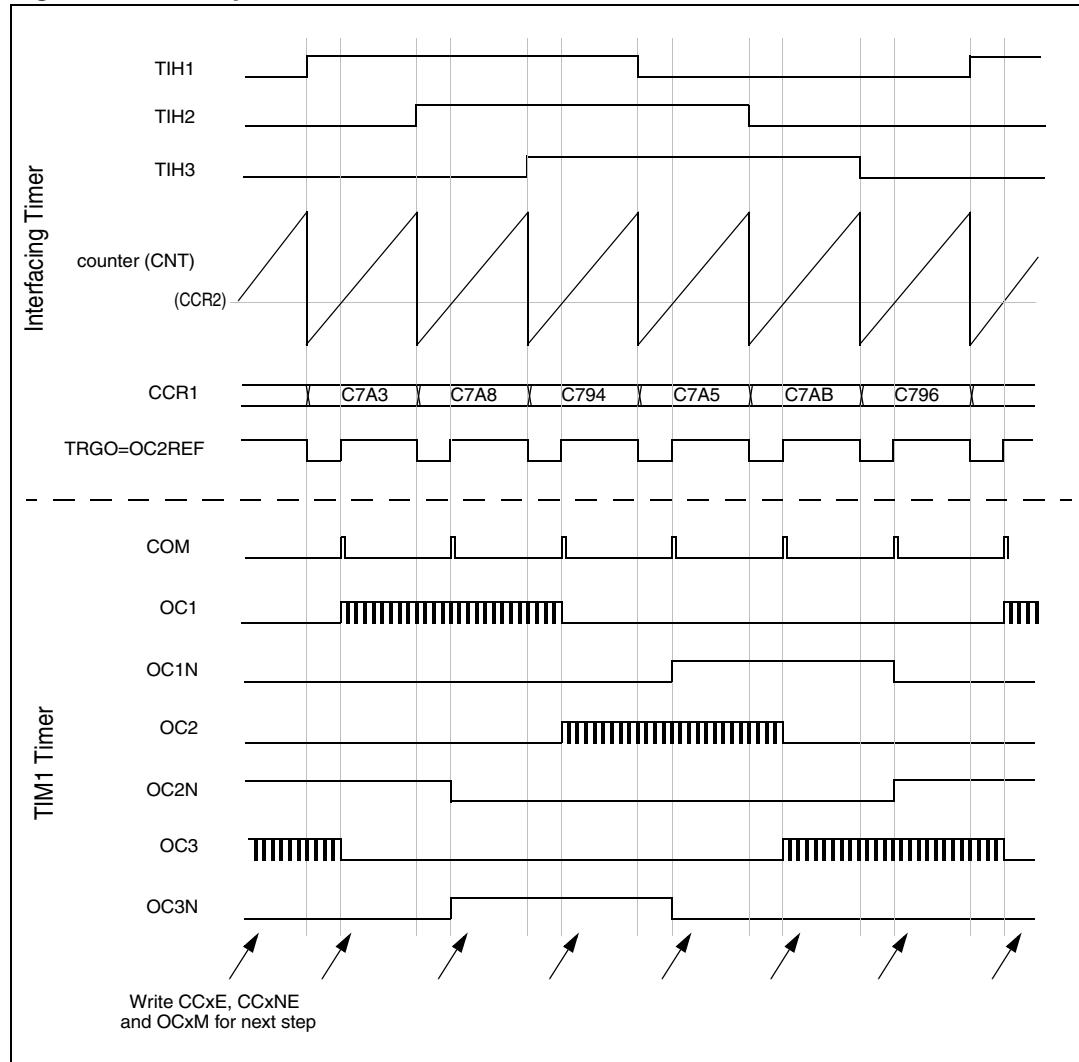
Example: you want to change the PWM configuration of your Advanced Control Timer TIM1 timer after a programmed delay each time a change occurs on the Hall inputs connected to one of the TIMx timers.

- Configure 3 timer inputs ORed to the TI1 input channel by writing the TI1S bit in the TIMx_CR2 register to ‘1’,
- Program the time base: write the TIMx_ARR to the max value (the counter must be cleared by the TI1 change. Set the prescaler to get a maximum counter period longer than the time between 2 changes on the sensors,
- Program the channel 1 in capture mode (TRC selected): write the CC1S bits in the TIMx_CCMR1 register to ‘01’. You can also program the digital filter if needed,
- Program the channel 2 in PWM 2 mode with the desired delay: write the OC2M bits to ‘111’ and the CC2S bits to ‘00’ in the TIM1_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 timer, 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 TIM1_CR2 register) and the COM event is controlled by the trigger input (CCUS=1 in the TIM1_CR2 register). The PWM control bits (CCxE, OCxM) are written after a COM event for the next step (this can be done in an interrupt subroutine generated by the rising edge of OC2REF).

The [Figure 68](#) describes this example.

Figure 68. Example of hall sensor interface



12.4.19 TIM1 and external trigger synchronization

The TIM1 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 TIM1_CR1 register is low, an update event UEV is generated. Then all the preloaded registers (TIM1_ARR, TIM1_CCRx) are updated.

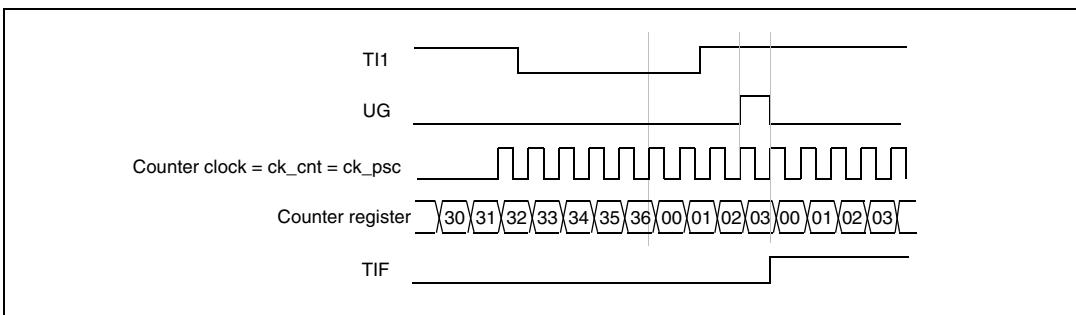
In the following example, the upcounter is cleared in response to a rising edge on TI1 input:

- Configure the channel 1 to detect rising edges on TI1. Configure the input filter duration (in this example, we don't need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so you don't need to configure it. The CC1S bits select the input capture source only, CC1S = 01 in the TIM1_CCMR1 register. Write CC1P=0 in TIM1_CCER register to validate the polarity (and detect rising edges only).
- Configure the timer in reset mode by writing SMS=100 in TIM1_SMCR register. Select TI1 as the input source by writing TS=101 in TIM1_SMCR register.
- Start the counter by writing CEN=1 in the TIM1_CR1 register.

The counter starts counting on the internal clock, then behaves normally until TI1 rising edge. When TI1 rises, the counter is cleared and restarts from 0. In the meantime, the trigger flag is set (TIF bit in the TIM1_SR register) and an interrupt request, or a DMA request can be sent if enabled (depending on the TIE and TDE bits in TIM1_DIER register).

The following figure shows this behavior when the auto-reload register TIM1_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 69. Control circuit in reset mode



Slave mode: Gated mode

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

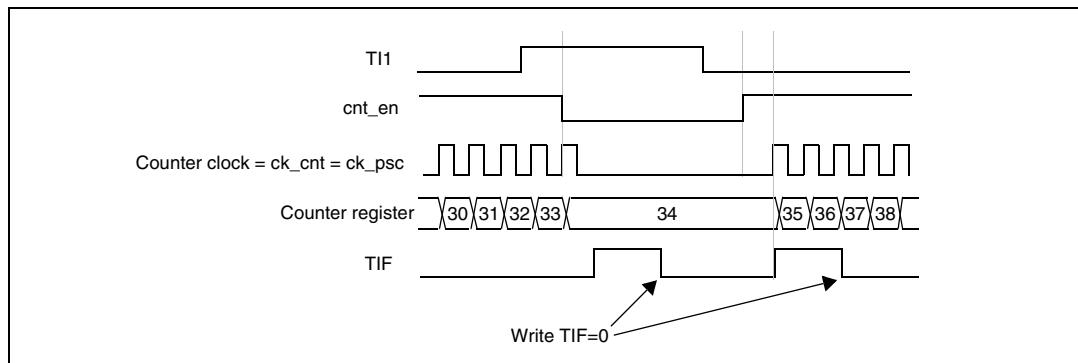
In the following example, the upcounter counts only when TI1 input is low:

- Configure the channel 1 to detect low levels on TI1. Configure the input filter duration (in this example, we don't need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so you don't need to configure it. The CC1S bits select the input capture source only, CC1S=01 in TIM1_CCMR1 register. Write CC1P=1 in TIM1_CCER register to validate the polarity (and detect low level only).
- Configure the timer in gated mode by writing SMS=101 in TIM1_SMCR register. Select TI1 as the input source by writing TS=101 in TIM1_SMCR register.
- Enable the counter by writing CEN=1 in the TIM1_CR1 register (in gated mode, the counter doesn't start if CEN=0, whatever is the trigger input level).

The counter starts counting on the internal clock as long as TI1 is low and stops as soon as TI1 becomes high. The TIF flag in the TIM1_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 70. Control circuit in gated mode



Slave mode: Trigger mode

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

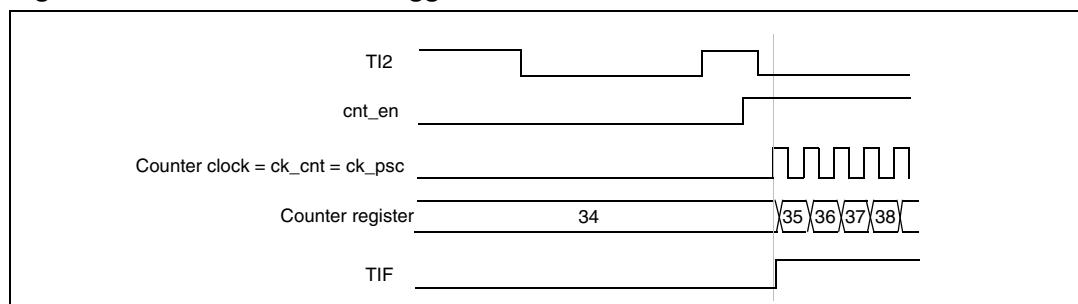
In the following example, the upcounter starts in response to a rising edge on TI2 input:

- Configure the channel 2 to detect rising edges on TI2. Configure the input filter duration (in this example, we don't need any filter, so we keep IC2F=0000). The capture prescaler is not used for triggering, so you don't need to configure it. The CC2S bits are configured to select the input capture source only, CC2S=01 in TIM1_CCMR1 register. Write CC2P=1 in TIM1_CCER register to validate the polarity (and detect low level only).
- Configure the timer in trigger mode by writing SMS=110 in TIM1_SMCR register. Select TI2 as the input source by writing TS=110 in TIM1_SMCR register.

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

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

Figure 71. 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 TIM1_SMCR register.

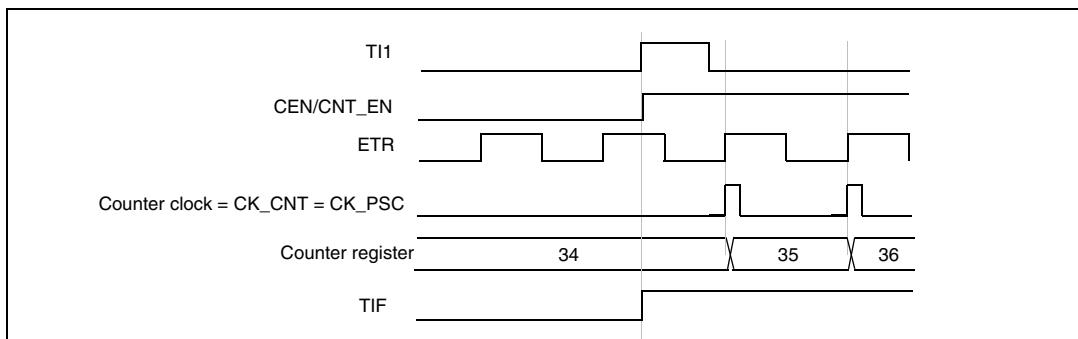
In the following example, the upcounter is incremented at each rising edge of the ETR signal as soon as a rising edge of TI1 occurs:

1. Configure the external trigger input circuit by programming the TIM1_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 TIM1_CCMR1 register to select only the input capture source
 - CC1P=0 in TIM1_CCER register to validate the polarity (and detect rising edge only).
3. Configure the timer in trigger mode by writing SMS=110 in TIM1_SMCR register. Select TI1 as the input source by writing TS=101 in TIM1_SMCR register.

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

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

Figure 72. Control circuit in external clock mode 2 + trigger mode



12.4.20 Timer synchronization

The TIM timers are linked together internally for timer synchronization or chaining. Refer to [Section 13.4.15: Timer synchronization on page 246](#) for details.

12.4.21 Debug mode

When the microcontroller enters debug mode (Cortex-M3 core halted), the TIM1 counter either continues to work normally or stops, depending on DBG_TIM1_STOP configuration bit in DBG module. For more details, refer to [Section 20.15.2: Debug support for timers, watchdog, bxCAN and I2C](#).

12.5 TIM1 register description

Refer to [Section 1.1 on page 23](#) for a list of abbreviations used in register descriptions.

12.5.1 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
						Reserved	CKD[1:0]	ARPE	CMS[1:0]	DIR	OPM	URS	UDIS	CEN	
						Res.	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:10 Reserved, always read as 0

Bits 9:8 **CKD[1:0]: Clock division.**

This bit-field indicates the division ratio between the timer clock (CK_INT) frequency and the dead-time and sampling clock (t_{DTS}) used by the dead-time generators and the digital filters (ETR, TIx),

00: $t_{DTS}=t_{CK_INT}$

01: $t_{DTS}=2*t_{CK_INT}$

10: $t_{DTS}=4*t_{CK_INT}$

11: Reserved, do not program this value.

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

0: TIM1_ARR register is not buffered.

1: TIM1_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 TIM1_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 TIM1_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 TIM1_CCMRx register) are set both when the counter is counting up or down.

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

Bit 4 **DIR: Direction.**

0: Counter used as upcounter.

1: Counter used as downcounter.

Note: This bit is read only when the timer is configured in Center-aligned mode or Encoder mode.

Bit 3 **OPM: One pulse mode.**

0: Counter is not stopped at update event

1: Counter stops counting at the next update event (clearing the bit CEN).

Bit 2 URS: *Update request source.*

This bit is set and cleared by software to select the UEV event sources.

0: Any of the following events generate an update interrupt or DMA request if enabled. These events can be:

- Counter overflow/underflow
 - Setting the UG bit
 - Update generation through the slave mode controller
- 1: Only counter overflow/underflow generates an update interrupt or DMA request if enabled.

Bit 1 UDIS: *Update disable.*

This bit is set and cleared by software to enable/disable UEV event generation.

0: UEV enabled. The Update (UEV) event is generated by one of the following events:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

Buffered registers are then loaded with their preload values.

1: UEV disabled. The Update event is not generated, shadow registers keep their value (ARR, PSC, CCRx). However the counter and the prescaler are reinitialized if the UG bit is set or if a hardware reset is received from the slave mode controller.

Bit 0 CEN: *Counter enable.*

0: Counter disabled

1: Counter enabled

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

12.5.2 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	
Res.	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit 15 Reserved, always read as 0

Bit 14 **OIS4:** *Output Idle state 4 (OC4 output).*

refer to OIS1 bit

Bit 13 **OIS3N:** *Output Idle state 3 (OC3N output).*

refer to OIS1N bit

Bit 12 **OIS3:** *Output Idle state 3 (OC3 output).*

refer to OIS1 bit

Bit 11 **OIS2N:** *Output Idle state 2 (OC2N output).*

refer to OIS1N bit

Bit 10 **OIS2:** *Output Idle state 2 (OC2 output).*

refer to OIS1 bit

Bit 9 **OIS1N:** *Output Idle state 1 (OC1N output).*

0: OC1N=0 after a dead-time when MOE=0

1: OC1N=1 after a dead-time when MOE=0

Note: This bit can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIM1_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 TIM1_BKR register).

Bit 7 **TI1S:** *TI1 Selection.*

0: The TIM1_CH1 pin is connected to TI1 input.

1: The TIM1_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

Bits 6:4 MMS[1:0]: Master Mode Selection.

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

000: **Reset** - the UG bit from the TIM1_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 TIM1_SMCR register).

010: **Update** - The update event is selected as trigger output (TRGO). For instance a master timer can then be used as a prescaler for a slave timer.

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

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

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

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

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

Bit 3 CCDS: Capture/Compare DMA Selection.

0: CCx DMA request sent when CCx event occurs

1: CCx DMA requests sent when update event occurs

Bit 2 CCUS: Capture/Compare Control Update Selection.

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

1: When capture/compare control bits are preloaded (CCPC=1), they are updated by setting the COM bit or when an rising edge occurs on TRGI.

Note: This bit acts only on channels that have a complementary output.

Bit 1 Reserved, always read as 0

Bit 0 CCPC: Capture/Compare Preloaded Control.

0: CCxE, CCxNE and OCxM bits are not preloaded

1: CCxE, CCxNE and OCxM bits are preloaded, after having been written, they are updated only when COM bit is set.

Note: This bit acts only on channels that have a complementary output.

12.5.3 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]			Res.	SMS[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	Res.	rw	rw	rw

Bit 15 **ETP: External trigger polarity.**

This bit selects whether ETR or \overline{ETR} is used for trigger operations

0: ETR is non-inverted, active at high level or rising edge.

1: ETR is inverted, active at low level or falling edge.

Bit 14 **ECE: External clock enable.**

This bit enables External clock mode 2.

0: External clock mode 2 disabled.

1: External clock mode 2 enabled. The counter is clocked by any active edge on the ETRF signal.

Note 1: Setting the ECE bit has the same effect as selecting external clock mode 1 with TRGI connected to ETRF (SMS=111 and TS=111).

Note 2: It is possible to simultaneously use external clock mode 2 with the following slave modes: reset mode, gated mode and trigger mode. Nevertheless, TRGI must not be connected to ETRF in this case (TS bits must not be 111).

Note 3: If external clock mode 1 and external clock mode 2 are enabled at the same time, the external clock input is ETRF.

Bits 13:12 **ETPS[1:0]: External trigger prescaler.**

External trigger signal ETRP frequency must be at most 1/4 of TIM1CLK frequency. A prescaler can be enabled to reduce ETRP frequency. It is useful when inputting fast external clocks.

00: Prescaler OFF.

01: ETRP frequency divided by 2.

10: ETRP frequency divided by 4.

11: ETRP frequency divided by 8.

Bits 11:8 ETF[3:0]: External trigger filter.

This bit-field then defines the frequency used to sample ETRP signal and the length of the digital filter applied to ETRP. The digital filter is made of an event counter in which N events are needed to validate a transition on the output:

- 0000: No filter, sampling is done at f_{DTS} .
- 0001: $f_{SAMPLING} = f_{CK_INT}$, N=2.
- 0010: $f_{SAMPLING} = f_{CK_INT}$, N=4.
- 0011: $f_{SAMPLING} = f_{CK_INT}$, N=8.
- 0100: $f_{SAMPLING} = f_{DTS}/2$, N=6.
- 0101: $f_{SAMPLING} = f_{DTS}/2$, N=8.
- 0110: $f_{SAMPLING} = f_{DTS}/4$, N=6.
- 0111: $f_{SAMPLING} = f_{DTS}/4$, N=8.
- 1000: $f_{SAMPLING} = f_{DTS}/8$, N=6.
- 1001: $f_{SAMPLING} = f_{DTS}/8$, N=8.
- 1010: $f_{SAMPLING} = f_{DTS}/16$, N=5.
- 1011: $f_{SAMPLING} = f_{DTS}/16$, N=6.
- 1100: $f_{SAMPLING} = f_{DTS}/16$, N=8.
- 1101: $f_{SAMPLING} = f_{DTS}/32$, N=5.
- 1110: $f_{SAMPLING} = f_{DTS}/32$, N=6.
- 1111: $f_{SAMPLING} = f_{DTS}/32$, N=8.

Bit 7 MSM: Master/slave mode.

- 0: No action
- 1: The effect of an event on the trigger input (TRGI) is delayed to allow a perfect synchronization between the current timer and its slaves (through TRGO). It is useful if we want to synchronize several timers on a single external event.

Bits 6:4 TS[2:0]: Trigger selection.

This bit-field selects the trigger input to be used to synchronize the counter.

- 000: Reserved
- 001: TIM2 (ITR1)
- 010: TIM3 (ITR2)
- 011: TIM4 (ITR3)
- 100: TI1 Edge Detector (TI1F_ED)
- 101: Filtered Timer Input 1 (TI1FP1)
- 110: Filtered Timer Input 2 (TI2FP2)
- 111: External Trigger input (ETRF)

Note: These bits must be changed only when they are not used (e.g. when SMS=000) to avoid wrong edge detections at the transition.

Bit 3 Reserved, always read as 0.

Bits 2:0 SMS Slave mode selection.

When external signals are selected the active edge of the trigger signal (TRGI) is linked to the polarity selected on the external input (see Input Control register and Control Register description).

000: Slave mode disabled - if CEN = '1' then the prescaler is clocked directly by the internal clock.

001: Encoder mode 1 - Counter counts up/down on TI2FP2 edge depending on TI1FP1 level.

010: Encoder mode 2 - Counter counts up/down on TI1FP1 edge depending on TI2FP2 level.

011: Encoder mode 3 - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input.

100: Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter and generates an update of the registers.

101: Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high. The counter stops (but is not reset) as soon as the trigger becomes low. Both start and stop of the counter are controlled.

110: Trigger Mode - The counter starts at a rising edge of the trigger TRGI (but it is not reset). Only the start of the counter is controlled.

111: External Clock Mode 1 - Rising edges of the selected trigger (TRGI) clock the counter.

Note: The gated mode must not be used if TI1F_ED is selected as the trigger input (TS='100').

Indeed, TI1F_ED outputs 1 pulse for each transition on TI1F, whereas the gated mode checks the level of the trigger signal.

12.5.4 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	COMD E	CC4D E	CC3D E	CC2D E	CC1D E	UDE	BIE	TIE	COMI E	CC4IE	CC3IE	CC2IE	CC1IE	UIE
Res.	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 Reserved, always read as 0.

Bit 14 **TDE**: *Trigger DMA request enable*.

- 0: Trigger DMA request disabled.
- 1: Trigger DMA request enabled.

Bit 13 **COMDE**: *COM DMA request enable*.

- 0: COM DMA request disabled.
- 1: COM DMA request enabled.

Bit 12 **CC4DE**: *Capture/Compare 4 DMA request enable*.

- 0: CC4 DMA request disabled.
- 1: CC4 DMA request enabled.

Bit 11 **CC3DE**: *Capture/Compare 3 DMA request enable*.

- 0: CC3 DMA request disabled.
- 1: CC3 DMA request enabled.

Bit 10 **CC2DE**: *Capture/Compare 2 DMA request enable*.

- 0: CC2 DMA request disabled.
- 1: CC2 DMA request enabled.

Bit 9 **CC1DE**: *Capture/Compare 1 DMA request enable*.

- 0: CC1 DMA request disabled.
- 1: CC1 DMA request enabled.

Bit 8 **UDE**: *Update DMA request enable*.

- 0: Update DMA request disabled.
- 1: Update DMA request enabled.

Bit 7 **BIE**: *Break interrupt enable*.

- 0: Break interrupt disabled.
- 1: Break interrupt enabled.

Bit 6 **TIE**: *Trigger interrupt enable*.

- 0: Trigger interrupt disabled.
- 1: Trigger interrupt enabled.

Bit 5 **COMIE**: *COM interrupt enable*.

- 0: COM interrupt disabled.
- 1: COM interrupt enabled.

Bit 4 **CC4IE**: *Capture/Compare 4 interrupt enable*.

- 0: CC4 interrupt disabled.
- 1: CC4 interrupt enabled.

Bit 3 **CC3IE**: Capture/Compare 3 interrupt enable.

- 0: CC3 interrupt disabled.
- 1: CC3 interrupt enabled.

Bit 2 **CC2IE**: Capture/Compare 2 interrupt enable.

- 0: CC2 interrupt disabled.
- 1: CC2 interrupt enabled.

Bit 1 **CC1IE**: Capture/Compare 1 interrupt enable.

- 0: CC1 interrupt disabled.
- 1: CC1 interrupt enabled.

Bit 0 **UIE**: Update interrupt enable.

- 0: Update interrupt disabled.
- 1: Update interrupt enabled.

12.5.5 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
Reserved		CC4OF	CC3OF	CC2OF	CC1OF	Res.	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF	
Res.	rc	rc	rc	rc	rc	Res.	rc	rc	rc	rc	rc	rc	rc	rc	rc

Bit 15:13 Reserved, always read as 0.

Bit 12 **CC4OF**: Capture/Compare 4 Overcapture Flag.

refer to CC1OF description

Bit 11 **CC3OF**: Capture/Compare 3 Overcapture Flag.

refer to CC1OF description

Bit 10 **CC2OF**: Capture/Compare 2 Overcapture Flag.

refer to CC1OF description

Bit 9 **CC1OF**: Capture/Compare 1 Overcapture Flag.

This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to '0'.

0: No overcapture has been detected.

1: The counter value has been captured in TIM1_CCR1 register while CC1IF flag was already set

Bit 8 Reserved, always read as 0.

Bit 7 **BIF**: Break interrupt Flag.

This flag is set by hardware as soon as the break input goes active. It can be cleared by software if the break input is not active.

0: No break event occurred.

1: An active level has been detected on the break input.

Bit 6 **TIF**: Trigger interrupt Flag.

This flag is set by hardware on trigger event (active edge detected on TRGI input when the slave mode controller is enabled in all modes but gated mode, both edges in case gated mode is selected). It is cleared by software.

0: No trigger event occurred.

1: Trigger interrupt pending.

Bit 5 COMIF: *COM interrupt Flag.*

This flag is set by hardware on COM event (when Capture/compare Control bits - CCxE, CCxNE, OCxM - have been updated). It is cleared by software.

- 0: No COM event occurred.
- 1: COM interrupt pending.

Bit 4 CC4IF: *Capture/Compare 4 interrupt Flag.*

refer to CC1IF description

Bit 3 CC3IF: *Capture/Compare 3 interrupt Flag.*

refer to CC1IF description

Bit 2 CC2IF: *Capture/Compare 2 interrupt Flag.*

refer to CC1IF description

Bit 1 CC1IF: *Capture/Compare 1 interrupt Flag.***If channel CC1 is configured as output:**

This flag is set by hardware when the counter matches the compare value, with some exception in center-aligned mode (refer to the CMS bits in the TIM1_CR1 register description). It is cleared by software.

0: No match.

1: The content of the counter TIM1_CNT has matched the content of the TIM1_CCR1 register.

If channel CC1 is configured as input:

This bit is set by hardware on a capture. It is cleared by software or by reading the TIM1_CCR1 register.

0: No input capture occurred.

1: The counter value has been captured in TIM1_CCR1 register (An edge has been detected on IC1 which matches the selected polarity).

Bit 0 UIF: *Update interrupt Flag.*

This bit is set by hardware on an update event. It is cleared by software.

0: No update occurred.

1: Update interrupt pending. This bit is set by hardware when the registers are updated:

- At overflow or underflow regarding the repetition downcounter value (update if REP_CNT=0) and if the UDIS=0 in the TIM1_CR1 register.
- When CNT is reinitialized by software using the UG bit in TIM1_EGR register, if URS=0 and UDIS=0 in the TIM1_CR1 register.
- When CNT is reinitialized by a trigger event (refer to [Section 12.5.3: Slave mode control register \(TIM1_SMCR\)](#)), if URS=0 and UDIS=0 in the TIM1_CR1 register.

12.5.6 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
								BG	TG	COMG	CC4G	CC3G	CC2G	CC1G	UG
							Res.	w	w	w	w	w	w	w	w

Bits 15:8 Reserved, always read as 0.

Bit 7 **BG**: *Break Generation*.

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action.

1: A break event is generated. MOE bit is cleared and BIF flag is set. Related interrupt or DMA transfer can occur if enabled.

Bit 6 **TG**: *Trigger Generation*.

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action.

1: The TIF flag is set in TIM1_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 TIM1_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 (TIM1_ARR) if DIR=1 (downcounting).

12.5.7 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 you must take care that the same bit can have a different meaning for the input stage and for the output stage.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
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 (TIM1_SMCR register)

Note: CC2S bits are writable only when the channel is OFF (CC2E = '0' in TIM1_CCER).

Bit 7 **OC1CE**: Output Compare 1Clear Enable

OC1CE: Output Compare 1 Clear Enable

0: OC1Ref is not affected by the ETRF Input

1: OC1Ref is cleared as soon as a High level is detected on ETRF input

Bits 6:4 OC1M: Output Compare 1 Mode.

These bits define the behavior of the output reference signal OC1REF from which OC1 and OC1N are derived. OC1REF is active high whereas OC1 and OC1N active level depends on CC1P and CC1NP bits.

000: Frozen - The comparison between the output compare register TIM1_CCR1 and the counter TIM1_CNT has no effect on the outputs.

001: Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIM1_CNT matches the capture/compare register 1 (TIM1_CCR1).

010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIM1_CNT matches the capture/compare register 1 (TIM1_CCR1).

011: Toggle - OC1REF toggles when TIM1_CNT=TIM1_CCR1.

100: Force inactive level - OC1REF is forced low.

101: Force active level - OC1REF is forced high.

110: PWM mode 1 - In upcounting, channel 1 is active as long as TIM1_CNT<TIM1_CCR1 else inactive. In downcounting, channel 1 is inactive (OC1REF='0') as long as TIM1_CNT>TIM1_CCR1 else active (OC1REF='1').

111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIM1_CNT<TIM1_CCR1 else active. In downcounting, channel 1 is active as long as TIM1_CNT>TIM1_CCR1 else inactive.

Note 1: These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIM1_BDTR register) and CC1S='00' (the channel is configured in output).

Note 2: In PWM mode 1 or 2, the OCREF level changes only when the result of the comparison changes or when the output compare mode switches from “frozen” mode to “PWM” mode.

Bit 3 OC1PE: Output Compare 1 Preload enable.

0: Preload register on TIM1_CCR1 disabled. TIM1_CCR1 can be written at anytime, the new value is taken in account immediately.

1: Preload register on TIM1_CCR1 enabled. Read/Write operations access the preload register. TIM1_CCR1 preload value is loaded in the active register at each update event.

Note 1: These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIM1_BDTR register) and CC1S='00' (the channel is configured in output).

Note 2: The PWM mode can be used without validating the preload register only in one pulse mode (OPM bit set in TIM1_CR1 register). Else the behavior is not guaranteed.

Bit 2 OC1FE: Output Compare 1 Fast enable.

This bit is used to accelerate the effect of an event on the trigger in input on the CC output.

0: CC1 behaves normally depending on counter and CCR1 values even when the trigger is ON. The minimum delay to activate CC1 output when an edge occurs on the trigger input is 5 clock cycles.

1: An active edge on the trigger input acts like a compare match on CC1 output. Then, OC is set to the compare level independently from the result of the comparison. Delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. 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 (TIM1_SMCR register)

Note: CC1S bits are writable only when the channel is OFF (CC1E = '0' in TIM1_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 (TIM1_SMCR register)

Note: CC2S bits are writable only when the channel is OFF (CC2E = '0' in TIM1_CCER).

Bits 7:4 **IC1F[3:0]**: *Input Capture 1 Filter*.

This bit-field defines the frequency used to sample TI1 input and the length of the digital filter applied to TI1. The digital filter is made of an event counter in which N events are needed to validate a transition on the output:

0000: No filter, sampling is done at f_{DTS} .

0001: $f_{SAMPLING} = f_{CK_INT}$, N=2.

0010: $f_{SAMPLING} = f_{CK_INT}$, N=4.

0011: $f_{SAMPLING} = f_{CK_INT}$, N=8.

0100: $f_{SAMPLING} = f_{DTS}/2$, N=6.

0101: $f_{SAMPLING} = f_{DTS}/2$, N=8.

0110: $f_{SAMPLING} = f_{DTS}/4$, N=6.

0111: $f_{SAMPLING} = f_{DTS}/4$, N=8.

1000: $f_{SAMPLING} = f_{DTS}/8$, N=6.

1001: $f_{SAMPLING} = f_{DTS}/8$, N=8.

1010: $f_{SAMPLING} = f_{DTS}/16$, N=5.

1011: $f_{SAMPLING} = f_{DTS}/16$, N=6.

1100: $f_{SAMPLING} = f_{DTS}/16$, N=8.

1101: $f_{SAMPLING} = f_{DTS}/32$, N=5.

1110: $f_{SAMPLING} = f_{DTS}/32$, N=6.

1111: $f_{SAMPLING} = f_{DTS}/32$, N=8.

Bits 3:2 **IC1PSC**: *Input Capture 1 Prescaler*.

This bit-field defines the ratio of the prescaler acting on CC1 input (IC1).

The prescaler is reset as soon as CC1E='0' (TIM1_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 (TIM1_SMCR register)

Note: CC1S bits are writable only when the channel is OFF (CC1E = '0' in TIM1_CCER).

12.5.8 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	

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 (TIM1_SMCR register)

Note: CC4S bits are writable only when the channel is OFF (CC4E = '0' in TIM1_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 (TIM1_SMCR register)

Note: CC3S bits are writable only when the channel is OFF (CC3E = '0' in TIM1_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 (TIM1_SMCR register)

Note: CC4S bits are writable only when the channel is OFF (CC4E = '0' in TIM1_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 (TIM1_SMCR register)

Note: CC3S bits are writable only when the channel is OFF (CC3E = '0' in TIM1_CCER).

12.5.9 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
Reserved	CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E	
Res.	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:14 Reserved, always read as 0.

Bit 13 **CC4P**: *Capture/Compare 4 output Polarity*.

refer to CC1P description

Bit 12 **CC4E**: *Capture/Compare 4 output enable*.

refer to CC1E description

Bit 11 **CC3NP**: *Capture/Compare 3 Complementary output Polarity*.

refer to CC1NP description

Bit 10 **CC3NE**: *Capture/Compare 3 Complementary output enable*.

refer to CC1NE description

Bit 9 **CC3P**: *Capture/Compare 3 output Polarity*.

refer to CC1P description

Bit 8 **CC3E**: *Capture/Compare 3 output enable*.

refer to CC1E description

Bit 7 **CC2NP**: *Capture/Compare 2 Complementary output Polarity.*
refer to CC1NP description

Bit 6 **CC2NE**: *Capture/Compare 2 Complementary output enable.*
refer to CC1NE description

Bit 5 **CC2P**: *Capture/Compare 2 output Polarity.*
refer to CC1P description

Bit 4 **CC2E**: *Capture/Compare 2 output enable.*
refer to CC1E description

Bit 3 **CC1NP**: *Capture/Compare 1 Complementary output Polarity.*
0: OC1N active high.
1: OC1N active low.

Note: This bit is not writable as soon as LOCK level 2 or 3 has been programmed (LOCK bits in TIM1_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.*

CC1 channel configured as output:

- 0: OC1 active high.
1: OC1 active low.

CC1 channel configured as input:

This bit selects whether IC1 or IC1 is used for trigger or capture operations.
0: non-inverted: capture is done on a rising edge of IC1. When used as external trigger, IC1 is non-inverted.
1: inverted: capture is done on a falling edge of IC1. When used as external trigger, IC1 is inverted.

Note: This bit is not writable as soon as LOCK level 2 or 3 has been programmed (LOCK bits in TIM1_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 (TIM1_CCR1) or not.

- 0: Capture disabled.
1: Capture enabled.

Table 38. Output control bits for complementary OCx and OCxN channels with break feature

Control bits					Output states ⁽¹⁾	
MOE bit	OSSI bit	OSSR bit	CCxE bit	CCxNE bit	OCx output state	OCxN output state
1	X	0	0	0	Output Disabled (not driven by the timer) OCx=0, OCx_EN=0	Output Disabled (not driven by the timer) OCxN=0, OCxN_EN=0
		0	0	1	Output Disabled (not driven by the timer) OCx=0, OCx_EN=0	OCxREF + Polarity OCxN=OCxREF xor CCxNP, OCxN_EN=1
		0	1	0	OCxREF + Polarity OCx=OCxREF xor CCxP, OCx_EN=1	Output Disabled (not driven by the timer) OCxN=0, OCxN_EN=0
		0	1	1	OCREF + Polarity + dead-time OCx_EN=1	Complementary to OCREF (not OCREF) + Polarity + dead-time OCxN_EN=1
		1	0	0	Output Disabled (not driven by the timer) OCx=CCxP, OCx_EN=0	Output Disabled (not driven by the timer) OCxN=CCxNP, OCxN_EN=0
		1	0	1	Off-State (output enabled with inactive state) OCx=CCxP, OCx_EN=1	OCxREF + Polarity OCxN=OCxREF xor CCxNP, OCxN_EN=1
		1	1	0	OCxREF + Polarity OCx=OCxREF xor CCxP, OCx_EN=1	Off-State (output enabled with inactive state) OCxN=CCxNP, OCxN_EN=1
		1	1	1	OCREF + Polarity + dead-time OCx_EN=1	Complementary to OCREF (not OCREF) + Polarity + dead-time OCxN_EN=1
0	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	

1. When both outputs of a channel are not used (CCxE = CCxNE = 0), the OISx, OISxN, CCxP and CCxNP bits must be kept cleared.

Note: The state of the external I/O pins connected to the complementary OCx and OCxN channels depends on the OCx and OCxN channel state and the GPIO and AFIO registers.

12.5.10 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.**

12.5.11 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 TIM1_EGR register or through trigger controller when configured in “reset mode”).

12.5.12 Auto-reload register (TIM1_ARR)

Address offset: 0x2C

Reset value: 0x0000

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

Bits 15:0 **ARR[15:0]: Prescaler Value.**

ARR is the value to be loaded in the actual auto-reload register.

Refer to the [Section 12.4.1: Time base unit on page 151](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

12.5.13 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
Reserved								REP[7:0]							
Res.								rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:8 Reserved, always read as 0.

Bits 7:0 **REP[7:0]: Repetition Counter Value.**

These bits allow the user to set-up the update rate of the compare registers (i.e. periodic transfers from preload to active registers) when preload registers are enable, as well as the update interrupt generation rate, if this interrupt is enable.

Each time the REP_CNT related downcounter reaches zero, an update event is generated and it restarts counting from REP value. As REP_CNT is reloaded with REP value only at the repetition update event U_RC, any write to the TIM1_RCR register is not taken in account until the next repetition update event.

It means in PWM mode (REP+1) corresponds to:

- the number of PWM periods in edge-aligned mode
- the number of half PWM period in center-aligned mode.

12.5.14 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 TIM1_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 TIM1_CNT and signaled on OC1 output.

If channel CC1 is configured as input:

CCR1 is the counter value transferred by the last input capture 1 event (IC1).

12.5.15 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 TIM1_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 TIM1_CNT and signalled on OC2 output.

If channel CC2 is configured as input:

CCR2 is the counter value transferred by the last input capture 2 event (IC2).

12.5.16 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 TIM1_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 TIM1_CNT and signalled on OC3 output.

If channel CC3 is configured as input:

CCR3 is the counter value transferred by the last input capture 3 event (IC3).

12.5.17 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 TIM1_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 TIM1_CNT and signalled on OC4 output.

If channel CC4 is configured as input:

CCR4 is the counter value transferred by the last input capture 4 event (IC4).

12.5.18 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
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 TIM1_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 TIM1_CCER register).

See OC/OCN enable description for more details ([Section 12.5.9: Capture/compare enable register \(TIM1_CCER\) on page 204](#)).

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 TIM1_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 TIM1_BDTR register).

Bit 12 **BKE**: *Break enable*.

- 0: Break inputs (BRK and BRK_ACTH) disabled
- 1: Break inputs (BRK and BRK_ACTH) enabled

Note: This bit can not be modified as long as LOCK level 1 has been programmed (LOCK bits in TIM1_BDTR register).

Bit 11 **OSSR**: *Off-State Selection for Run mode*.

This bit is used when MOE=1 on channels having a complementary output which are configured as outputs. OSSR is not implemented if no complementary output is implemented in the timer.

See OC/OCN enable description for more details ([Section 12.5.9: Capture/compare enable register \(TIM1_CCER\) on page 204](#)).

- 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 TIM1_BDTR register).

Bit 10 **OSSI**: *Off-State Selection for Idle mode*.

This bit is used when MOE=0 on channels configured as outputs.

See OC/OCN enable description for more details ([Section 12.5.9: Capture/compare enable register \(TIM1_CCER\) on page 204](#)).

- 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 TIM1_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 TIM1_BDTR register, OISx and OISxN bits in TIM1_CR2 register and BKE/BKP/AOE bits in TIM1_BDTR register can no longer be written.

10: LOCK Level 2 = LOCK Level 1 + CC Polarity bits (CCxP/CCxNP bits in TIM1_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 TIM1_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 TIM1_BDTR register has been written, their content is frozen until the next reset.

Bits 7:0 DTG[7:0]: Dead-Time Generator set-up.

This bit-field defines the duration of the dead-time inserted between the complementary outputs. DT correspond to this duration.

DTG[7:5]=0xx => DT=DTG[7:0]x t_{dtg} with t_{dtg}=t_{DTS}.

DTG[7:5]=10x => DT=(64+DTG[5:0])x t_{dtg} with T_{dtg}=2x t_{DTS}.

DTG[7:5]=110 => DT=(32+DTG[4:0])x t_{dtg} with T_{dtg}=8x t_{DTS}.

DTG[7:5]=111 => DT=(32+DTG[4:0])x t_{dtg} with T_{dtg}=16x t_{DTS}.

Example if T_{DTS}=125ns (8MHz), dead-time possible values are:

0 to 15875 ns by 125 ns steps,

16 us to 31750 ns by 250 ns steps,

32 us to 63us by 1 us steps,

64 us to 126 us by 2 us steps

Note: This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIM1_BDTR register).

12.5.19 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
Reserved				DBL[4:0]						Reserved	DBA[4:0]				
Res.	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:13 Reserved, always read as 0

Bits 12:8 DBL[4:0]: DMA Burst Length.

This 5-bit vector defines the length of DMA transfers in burst mode (the timer recognizes a burst transfer when a read or a write access is done to the TIM1_DMAR address), i.e. the number of bytes to be transferred.

00000: 1 byte,

00001: 2 bytes,

00010: 3 bytes,

...

10001: 18 bytes.

Bits 7:5 Reserved, always read as 0

Bits 4:0 DBA[4:0]: DMA Base Address.

This 5-bits vector defines the base-address for DMA transfers in burst mode (when read/write access are done through the TIM1_DMAR address). DBA is defined as an offset starting from the address of the TIM1_CR1 register.

Example:

00000: TIM1_CR1,

00001: TIM1_CR2,

00010: TIM1_SMCR,

...

12.5.20 DMA address for burst mode (TIM1_DMAR)

Address offset: 0x4C

Reset value: 0x0000

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:

“(TIM1_CR1 address) + DBA + (DMA index)” in which:

TIM1_CR1 address is the address of the control register 1,

DBA is the DMA base address configured in TIM1_DCR register,

DMA index is the offset automatically controlled by the DMA transfer, depending on the length of the transfer DBL in the TIM1_DCR register.

12.6 TIM1 register map

TIM1 registers are mapped as 16-bit addressable registers as described in the table below:

Table 39. TIM1 - Register map and reset values

Table 39. 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
0x1C	TIM1_CCMR2 <i>Output Compare mode</i>	Reserved												O24CE	OC4M [2:0]	O24FE	OC4S [1:0]	CC4S [1:0]	OC3E	OC3M [2:0]	OC3FEE	OC3S [1:0]											
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
	TIM1_CCMR2 <i>Input Capture mode</i>	Reserved												IC4F[3:0]	IC4P	PSC [1:0]	CC4S [1:0]	IC3F[3:0]	IC3P	PSC [1:0]	CC3S [1:0]												
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x20	TIM1_CCER	Reserved												CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E						
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x24	TIM1_CNT	Reserved												CNT[15:0]																			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x28	TIM1_PSC	Reserved												PSC[15:0]																			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x2C	TIM1_ARR	Reserved												ARR[15:0]																			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x30	TIM1_RCR	Reserved												REP[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			
0x34	TIM1_CCR1	Reserved												CCR1[15:0]																			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x38	TIM1_CCR2	Reserved												CCR2[15:0]																			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x3C	TIM1_CCR3	Reserved												CCR3[15:0]																			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x40	TIM1_CCR4	Reserved												CCR4[15:0]																			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x44	TIM1_BDTR	Reserved												MOE	AOE	BKP	BKE	OSMR	OSSI	LOCK [1:0]	DT[7:0]												
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x48	TIM1_DCR	Reserved												DBL[4:0]					Reserved	DBA[4:0]													
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x4C	TIM1_DMAR	Reserved												DMAB[15:0]																			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			

Refer to [Table 1 on page 27](#) for the register boundary addresses.

13 General purpose timer (TIMx)

13.1 Introduction

The General purpose timers consist of a 16-bit auto-reload counter driven by a programmable prescaler.

They may be used for a variety of purposes, including measuring the pulse lengths of input signals (*input capture*) or generating output waveforms (*output compare and PWM*).

Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the RCC clock controller prescalers.

The timers are completely independent, and do not share any resources. They can be synchronized together as described in [Section 13.4.15](#).

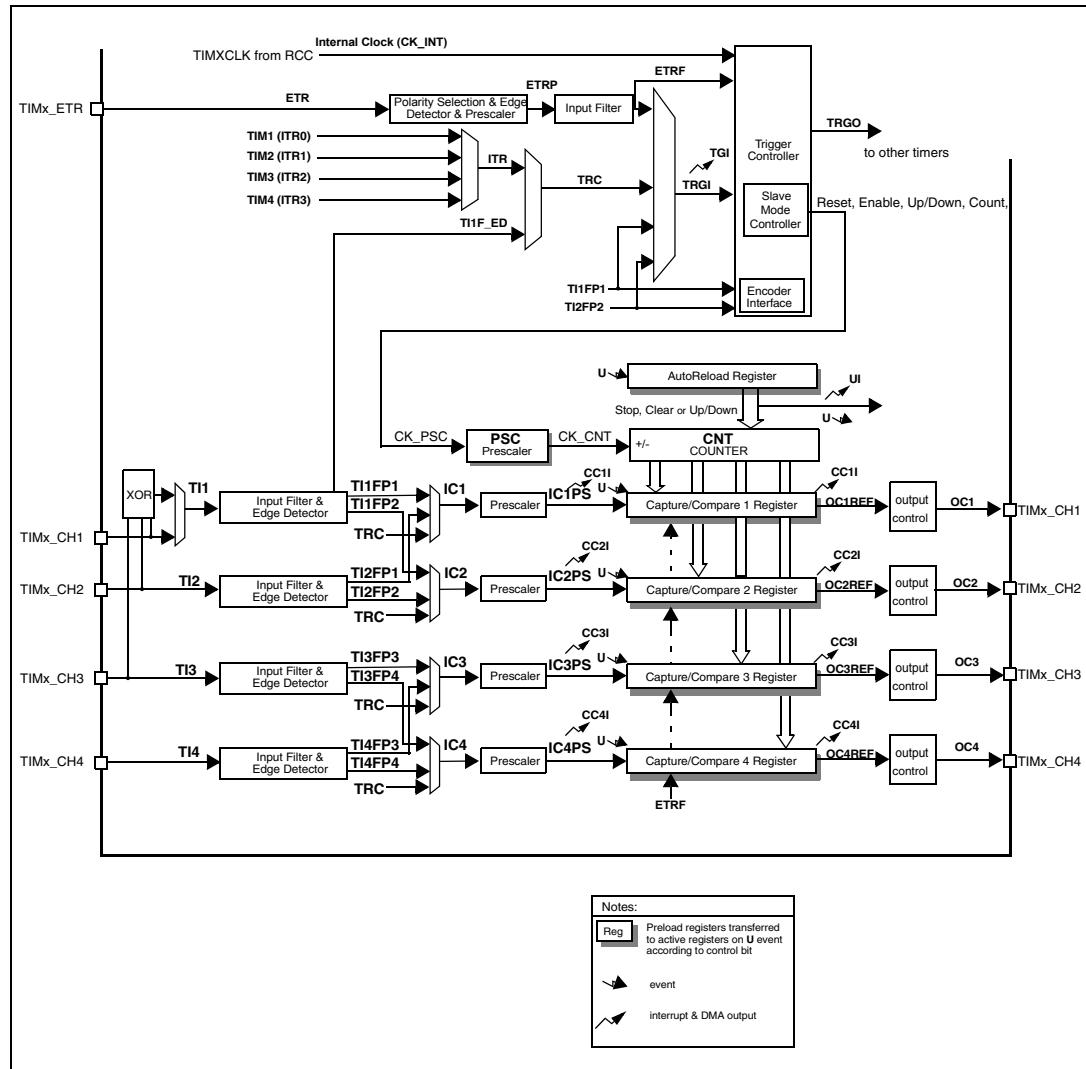
13.2 Main features

General purpose TIMx timer features include:

- 16-bit up, down, up/down auto-reload counter.
- 16-bit programmable prescaler allowing dividing (also “on the fly”) the counter clock frequency either by any factor between 1 and 65535.
- Up to 4 independent channels for:
 - Input Capture
 - Output Compare
 - PWM generation (Edge and Center-aligned Mode)
 - One Pulse Mode output
- Synchronization circuit to control the timer with external signals and to interconnect several timers between them.
- Interrupt/DMA generation on the following events:
 - Update: counter overflow/underflow, counter initialization (by software or internal/external trigger)
 - Trigger event (counter start, stop, initialization or count by internal/external trigger)
 - Input capture
 - Output compare

13.3 Block diagram

Figure 73. General-purpose timer block diagram



13.4 Functional description

13.4.1 Time base unit

The main block of the Programmable Timer is a 16-bit counter with its related auto-reload register. The counter can count up, down or both up and down. The counter clock can be divided by a prescaler.

The counter, the auto-reload register and the prescaler register can be written or read by software. This is true even when the counter is running.

The *Time Base Unit* includes:

- Counter Register (TIMx_CNT)
- Prescaler Register (TIMx_PSC):
- Auto-Reload Register (TIMx_ARR)

The auto-reload register is preloaded. Writing or reading the auto-reload register access the preload register. The content of the preload register is transferred in the shadow register permanently or at each update event UEV, depending on the auto-reload preload enable bit (ARPE) in TIMx_CR1 register. The update event is sent when the counter reaches the overflow (or underflow when downcounting) and if the UDIS bit equals 0 in the TIMx_CR1 register. It can also be generated by software. The generation of the update event is described in detail for each configuration.

The counter is clocked by the prescaler output CK_CNT, which is enabled only when the counter enable bit (CEN) in TIMx_CR1 register is set (refer also to the slave mode controller description to get more details on counter enabling).

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

Prescaler description

The prescaler can divide the counter clock frequency by any factor between 1 and 65536. It is based on a 16-bit counter controlled through a 16-bit register (in the TIMx_PSC register). It can be changed on the fly as this control register is buffered. The new prescaler ratio is taken in account at the next update event.

Figure 74 and *Figure 75* give some examples of the counter behavior when the prescaler ratio is changed on the fly:

Figure 74. Counter timing diagram with prescaler division change from 1 to 2

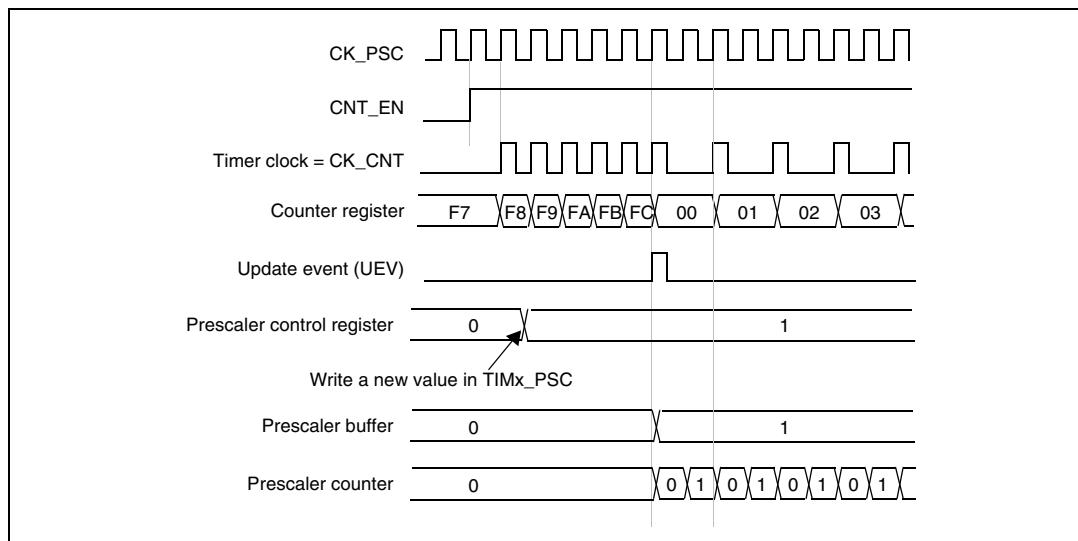
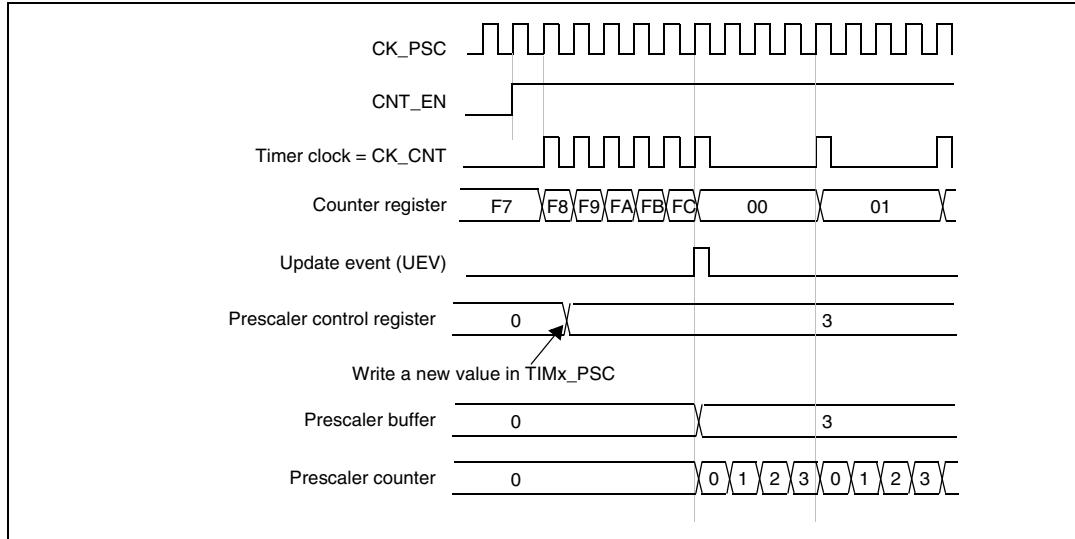


Figure 75. Counter timing diagram with prescaler division change from 1 to 4

13.4.2 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 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 (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.

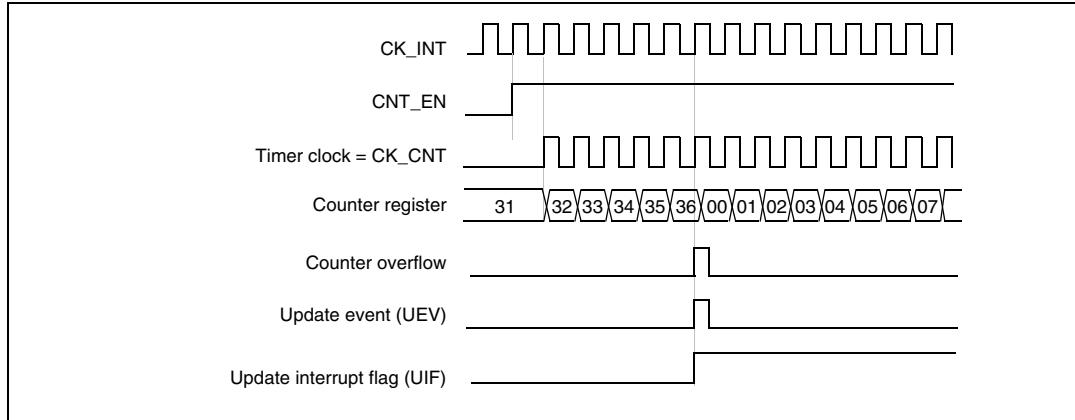
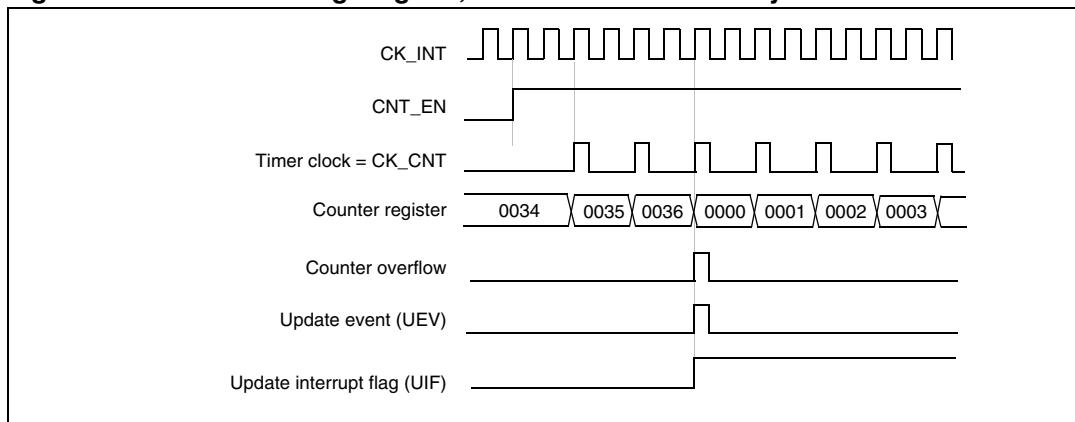
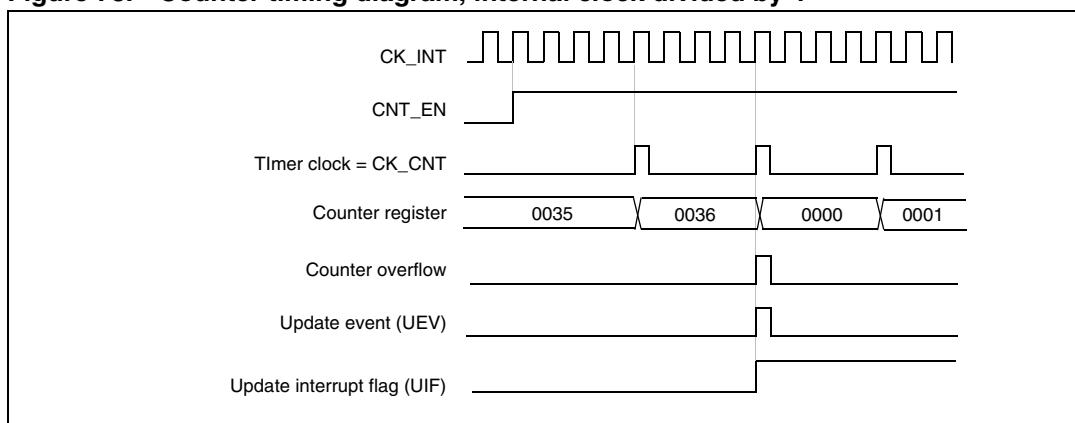
Figure 76. Counter timing diagram, internal clock divided by 1**Figure 77. Counter timing diagram, internal clock divided by 2****Figure 78. Counter timing diagram, internal clock divided by 4**

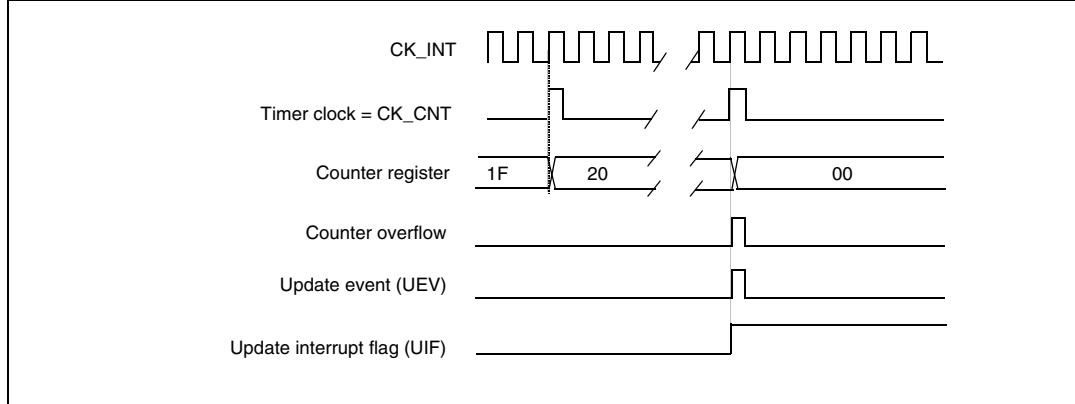
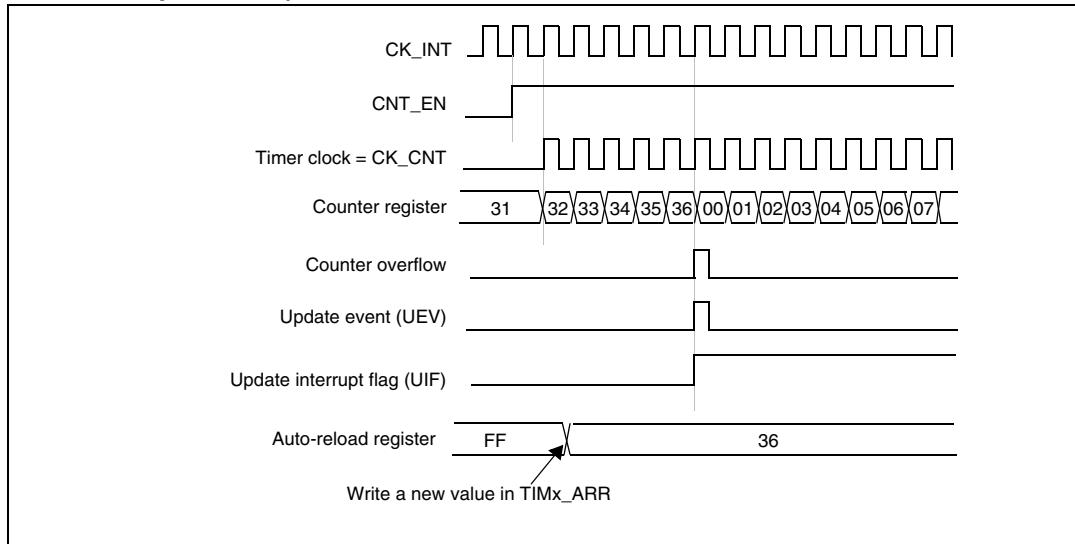
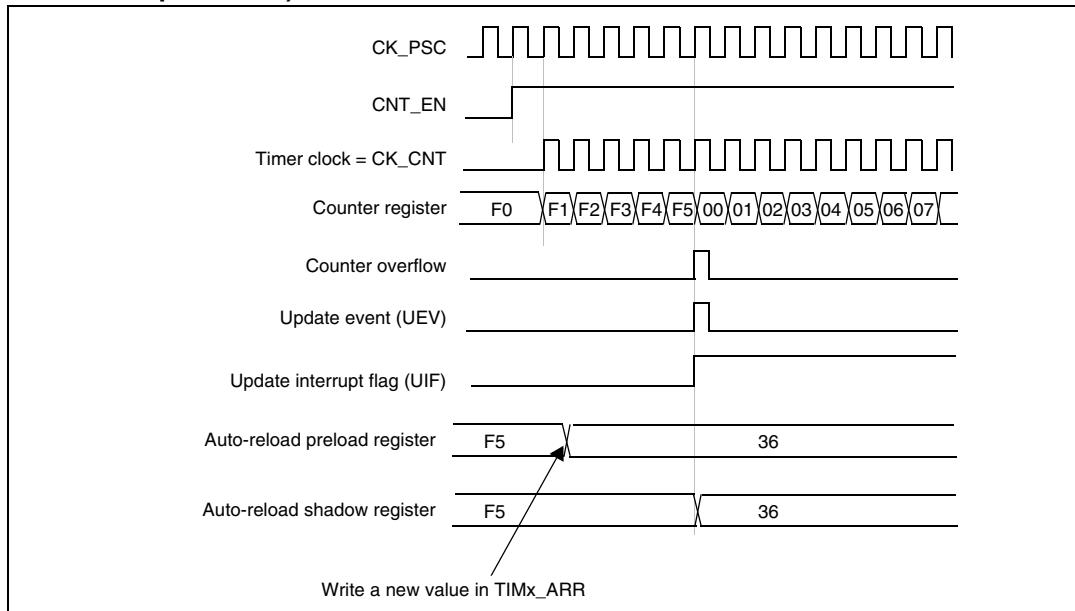
Figure 79. Counter timing diagram, internal clock divided by N**Figure 80. Counter timing diagram, Update event when ARPE=0 (TIMx_ARR not preloaded)**

Figure 81. 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 (contents 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.

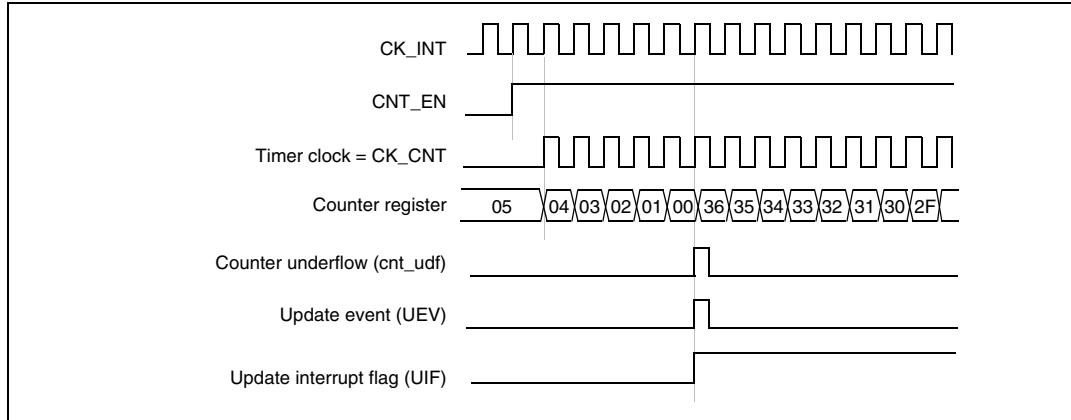
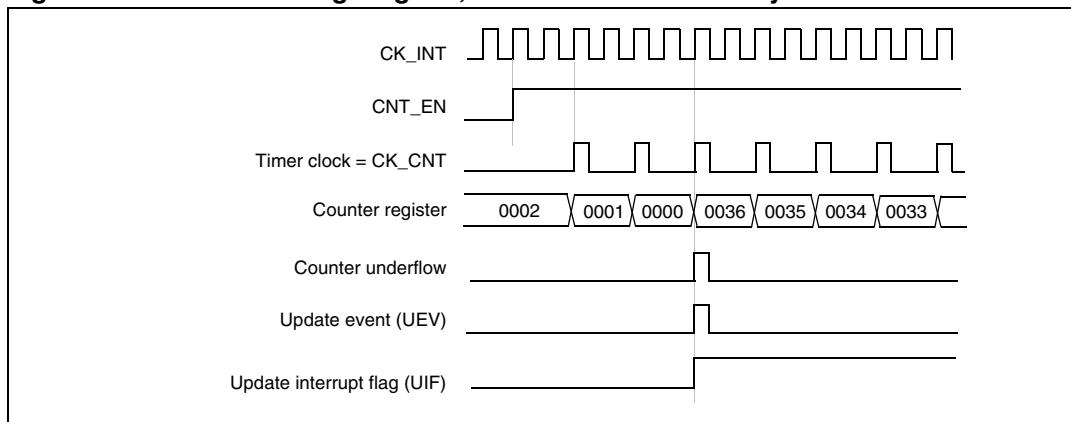
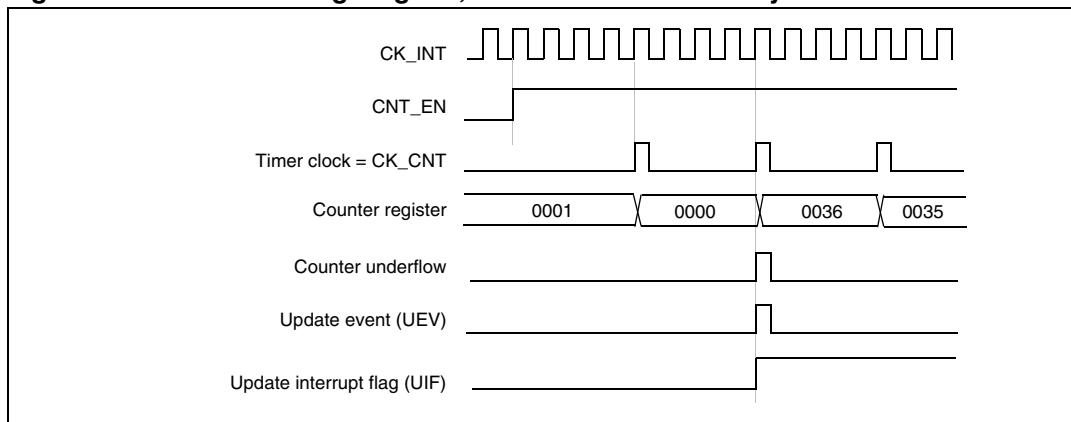
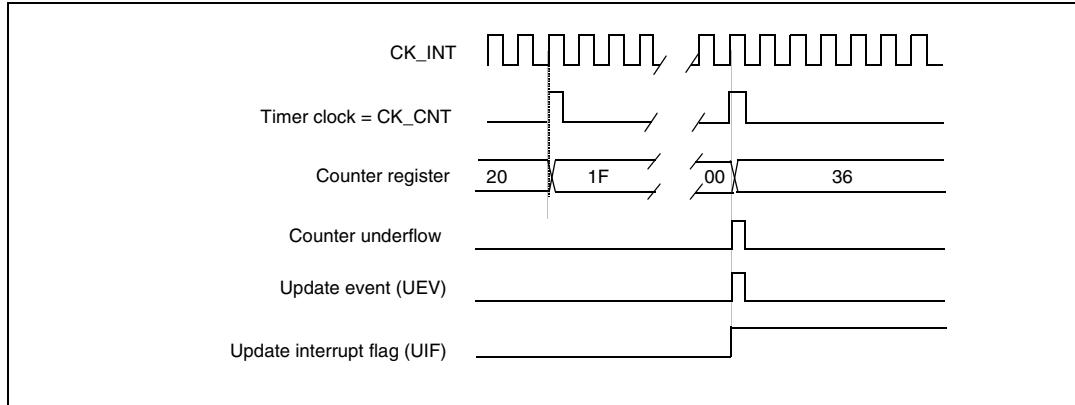
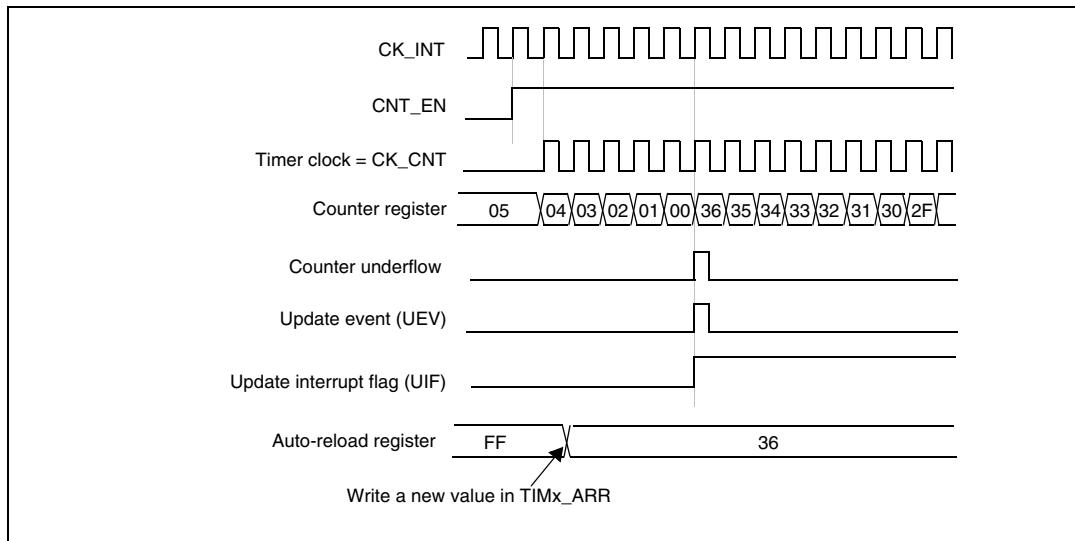
Figure 82. Counter timing diagram, internal clock divided by 1**Figure 83. Counter timing diagram, internal clock divided by 2****Figure 84. Counter timing diagram, internal clock divided by 4**

Figure 85. Counter timing diagram, internal clock divided by N**Figure 86. Counter timing diagram, Update event when repetition counter is not used**

Center-aligned mode (up/down counting)

In center-aligned mode, the counter counts from 0 to the auto-reload value (content of the TIMx_ARR register) – 1, generates a counter overflow event, then counts from the auto-reload value down to 1 and generates a counter underflow event. Then it restarts counting from 0.

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

The update event can be generated at each counter overflow and at each counter underflow or by setting the UG bit in the TIMx_EGR register (by software or by using the slave mode controller) also generates an update event. In this case, the counter restarts counting from 0, as well as the counter of the prescaler.

The UEV update event can be disabled by software by setting the UDIS bit in TIMx_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until the UDIS bit has been written to 0. However, the counter continues counting up and down, based on the current auto-reload value.

In addition, if the URS bit (update request selection) in TIMx_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupt when clearing the counter on the capture event.

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

- The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSC register).
- The auto-reload active register is updated with the preload value (contents 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 87. Counter timing diagram, internal clock divided by 1, TIMx_ARR=0x6

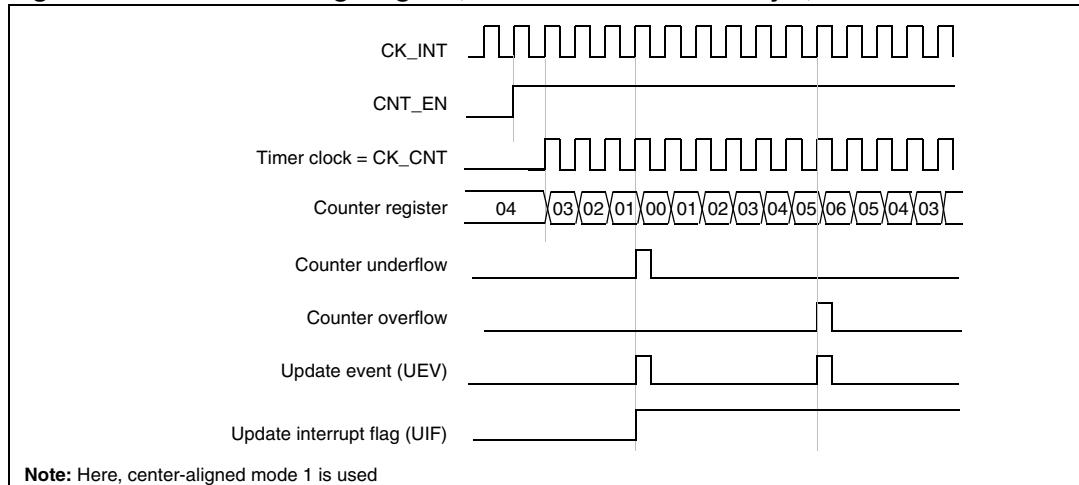


Figure 88. Counter timing diagram, internal clock divided by 2

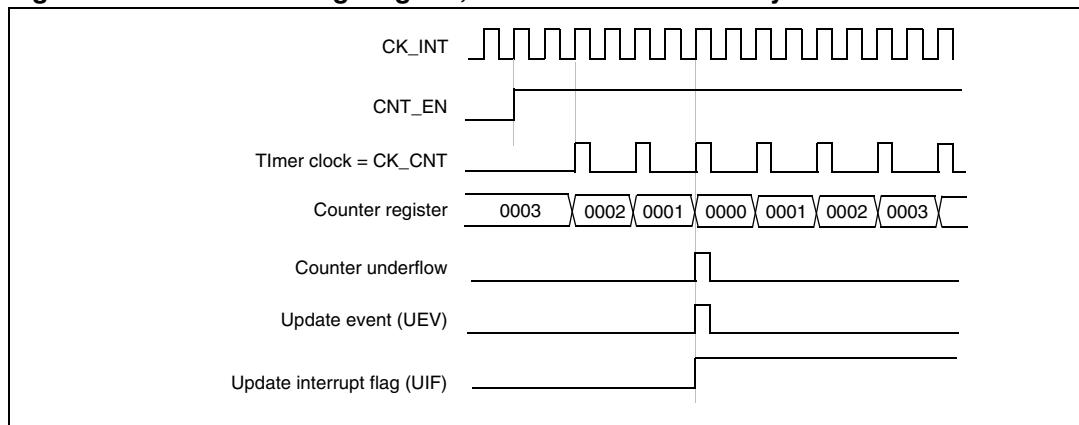


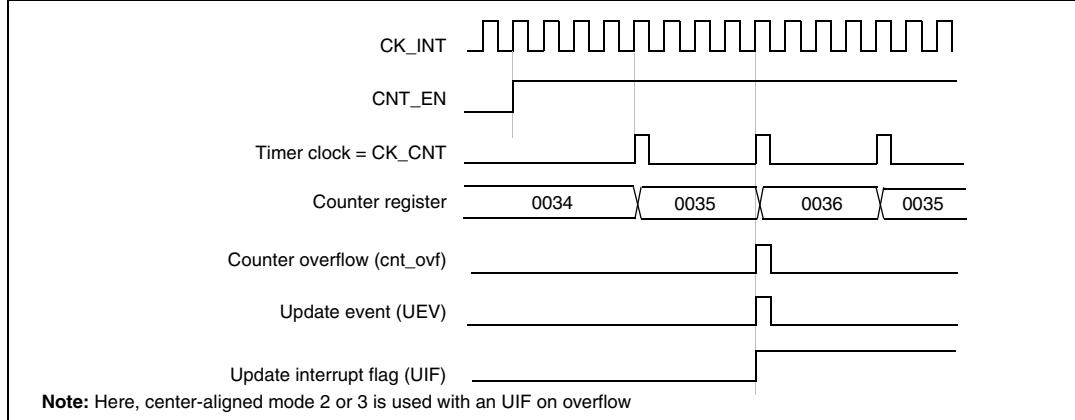
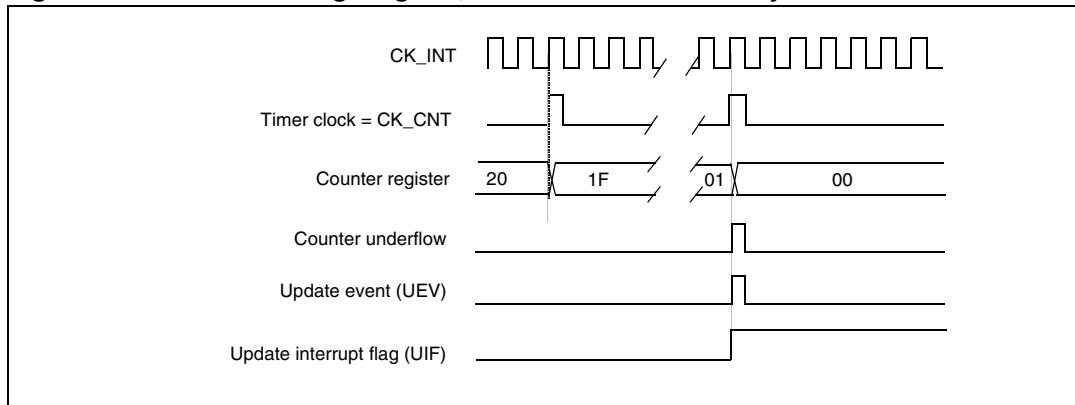
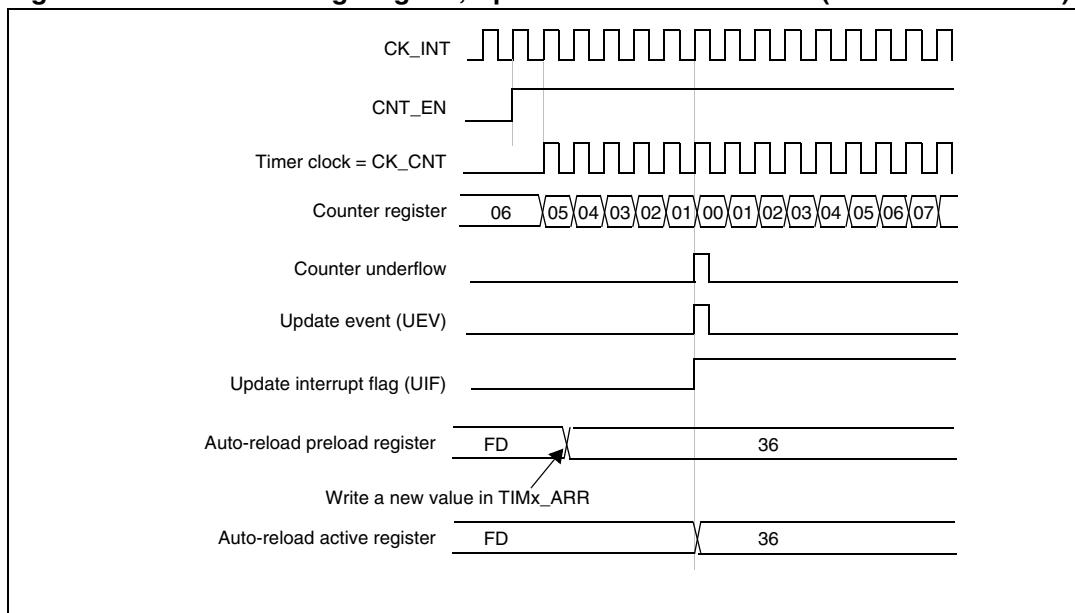
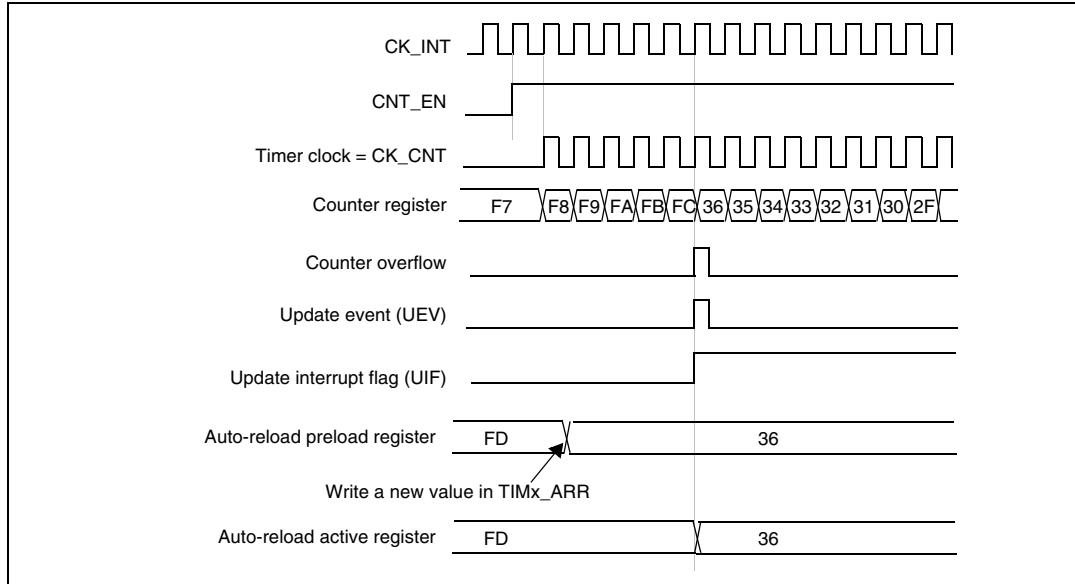
Figure 89. Counter timing diagram, internal clock divided by 4, TIMx_ARR=0x36**Figure 90. Counter timing diagram, internal clock divided by N****Figure 91. Counter timing diagram, Update event with ARPE=1 (counter underflow)**

Figure 92. Counter timing diagram, Update event with ARPE=1 (counter overflow)

13.4.3 Clock selection

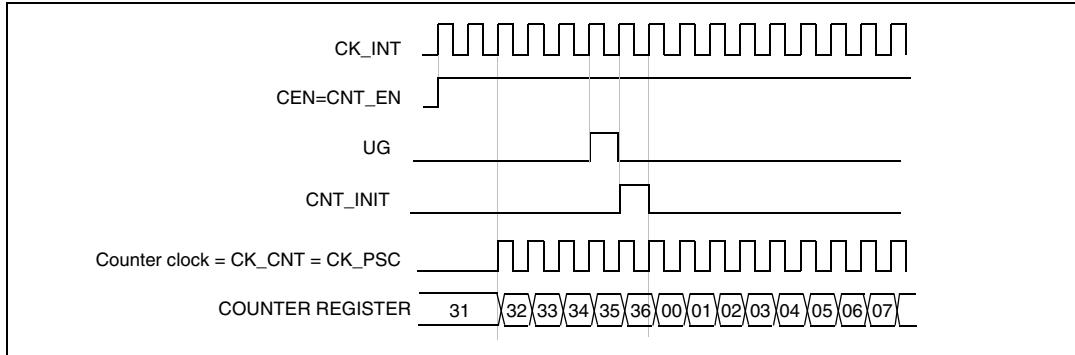
The counter clock can be provided by the following clock sources:

- Internal clock (CK_INT)
- External clock mode1: external input pin (TI_x)
- External clock mode2: external trigger input (ETR)
- Internal trigger inputs (ITRx): using one timer as prescaler for another timer, for example, you can configure Timer 1 to act as a prescaler for Timer 2. Refer to : [Using one timer as prescaler for the another on page 247](#) for more details.

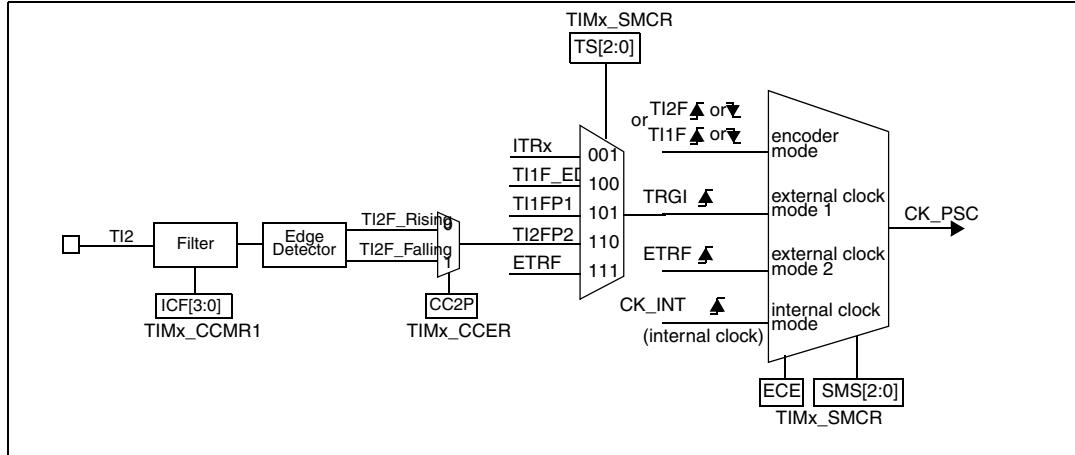
Internal clock source (CK_INT)

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 93 shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

Figure 93. 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 94. TI2 external clock connection example

For example, to configure the upcounter to count in response to a rising edge on the TI2 input, use the following procedure:

For example, to configure the upcounter to count in response to a rising edge on the TI2 input, use the following procedure:

1. Configure channel 2 to detect rising edges on the TI2 input by writing CC2S= '01' in the TIMx_CCMR1 register.
2. Configure the input filter duration by writing the IC2F[3:0] bits in the TIMx_CCMR1 register (if no filter is needed, keep IC2F=0000).

Note:

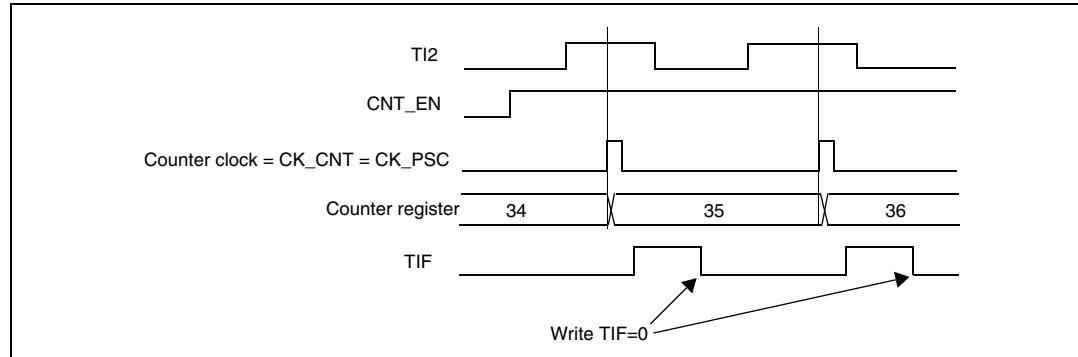
The capture prescaler is not used for triggering, so you don't need to configure it.

3. Select rising edge polarity by writing CC2P=0 in the TIMx_CCER register.
4. Configure the timer in external clock mode 1 by writing SMS=111 in the TIMx_SMCR register.
5. Select TI2 as the input source by writing TS=110 in the TIMx_SMCR register.
6. Enable the counter by writing CEN=1 in the TIMx_CR1 register.

When a rising edge occurs on TI2, the counter counts once and the TIF flag is set.

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

Figure 95. Control circuit in external clock mode 1



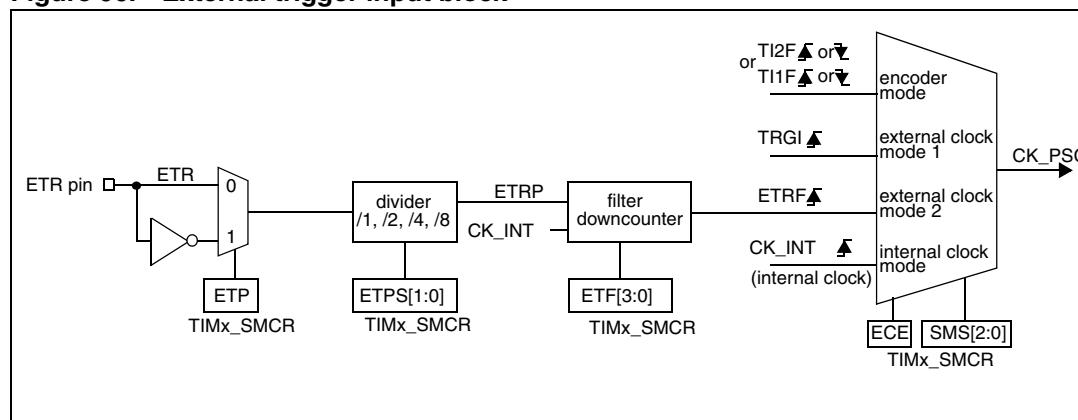
External clock source mode 2

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 96](#) gives an overview of the external trigger input block.

Figure 96. External trigger input block

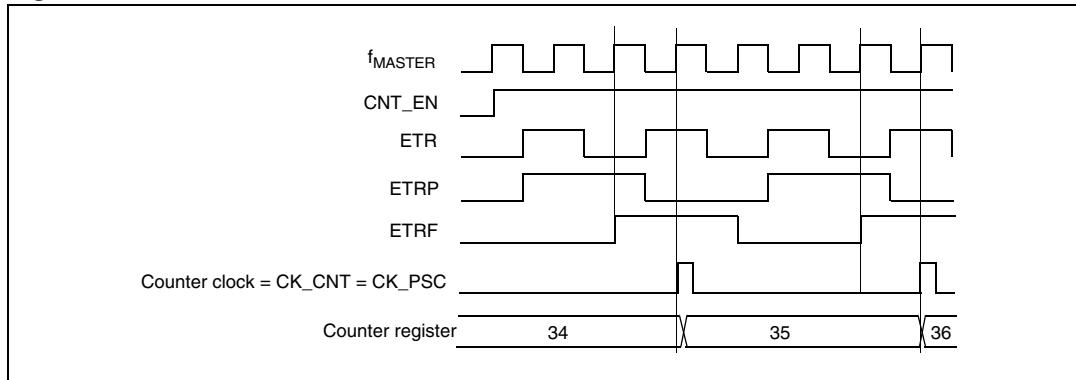


For example, to configure the upcounter to count each 2 rising edges on ETR, use the following procedure:

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 ETRP signal.

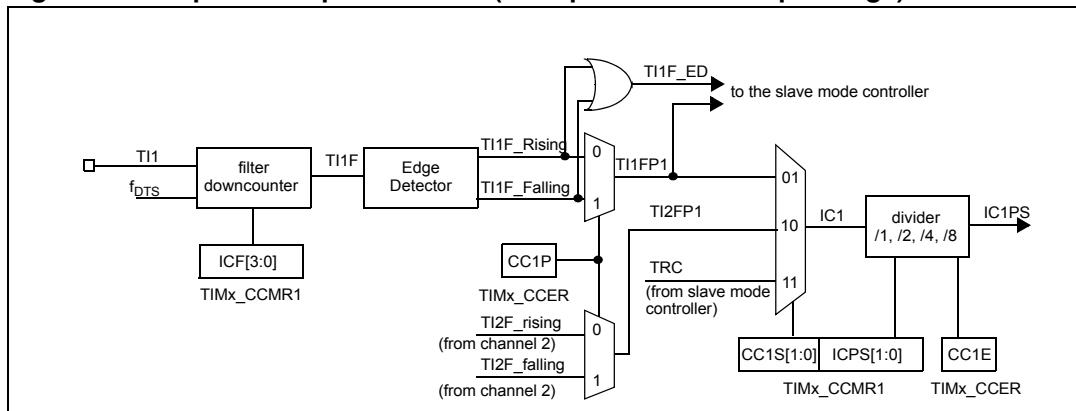
Figure 97. Control circuit in external clock mode 2

13.4.4 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 98. 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 99. Capture/compare channel 1 main circuit

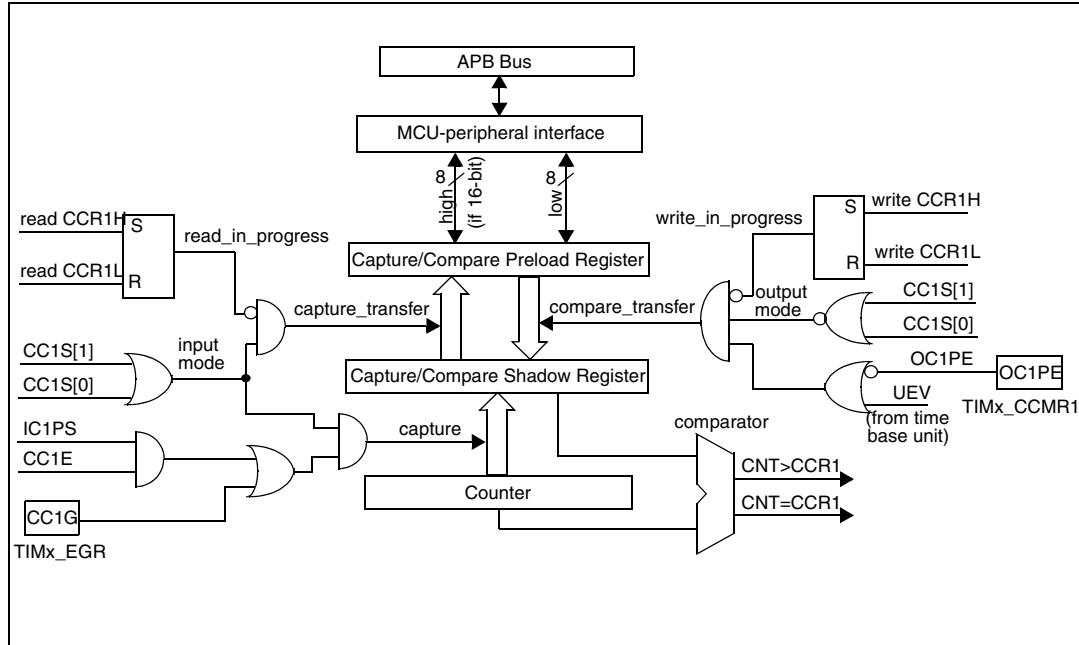
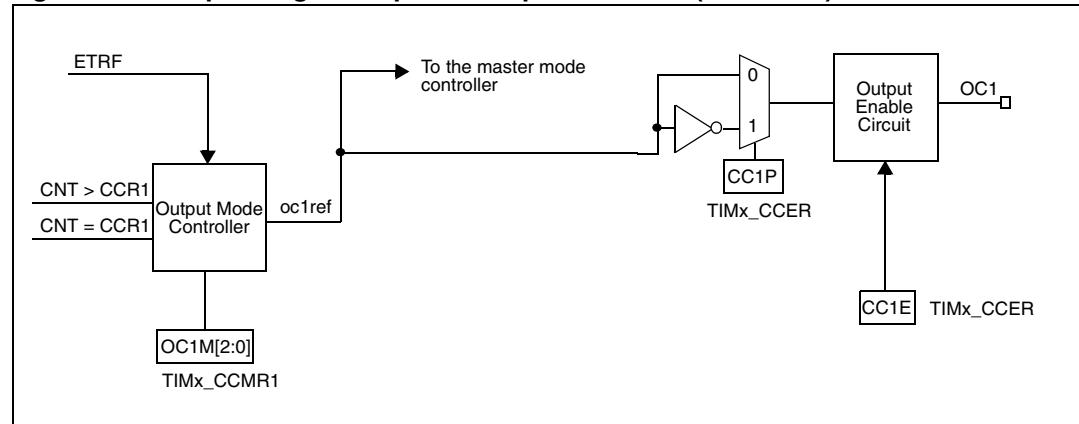


Figure 100. Output stage of capture/compare channel (channel 1)



The capture/compare block is made of one preload register and one shadow register. Write and read always access the preload register.

In capture mode, captures are actually done in the shadow register, which is copied into the preload register.

In compare mode, the content of the preload register is copied into the shadow register which is compared to the counter.

13.4.5 Input capture mode

In Input capture mode, the Capture/Compare Registers (TIMx_CCRx) are used to latch the value of the counter after a transition detected by the corresponding ICx signal. When a capture occurs, the corresponding CCXIF flag (TIMx_SR register) is set and an interrupt or a DMA request can be sent if they are enabled. If a capture occurs while the CCxIF flag was already high, then the over-capture flag CCxOF (TIMx_SR register) is set. CCxIF can be cleared by software by writing it to '0' or by reading the captured data stored in the TIMx_CCRx register. CCxOF is cleared when you write it to '0'.

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

- Select the active input: TIMx_CCR1 must be linked to the TI1 input, so write the CC1S bits to 01 in the TIMx_CCMR1 register. As soon as CC1S becomes different from 00, the channel is configured in input and the TIMx_CCR1 register becomes read-only.
- Program the input filter duration you need with respect to the signal you connect to the timer (when the input is one of the TIx (ICxF bits in the TIMx_CCMRx register). Let's imagine that, when toggling, the input signal is not stable during at 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.

When an input capture occurs:

- The TIMx_CCR1 register gets the value of the counter on the active transition.
- CC1IF flag is set (interrupt flag). CC1OF is also set if at least two consecutive captures occurred whereas the flag was not cleared.
- An interrupt is generated depending on the CC1IE bit.
- A DMA request is generated depending on the CC1DE bit.

In order to handle the overcapture, it is recommended to read the data before the overcapture flag. This is to avoid missing an overcapture which could happen after reading the flag and before reading the data.

Note:

IC interrupt and/or DMA requests can be generated by software by setting the corresponding CCxG bit in the TIMx_EGR register.

13.4.6 PWM input mode

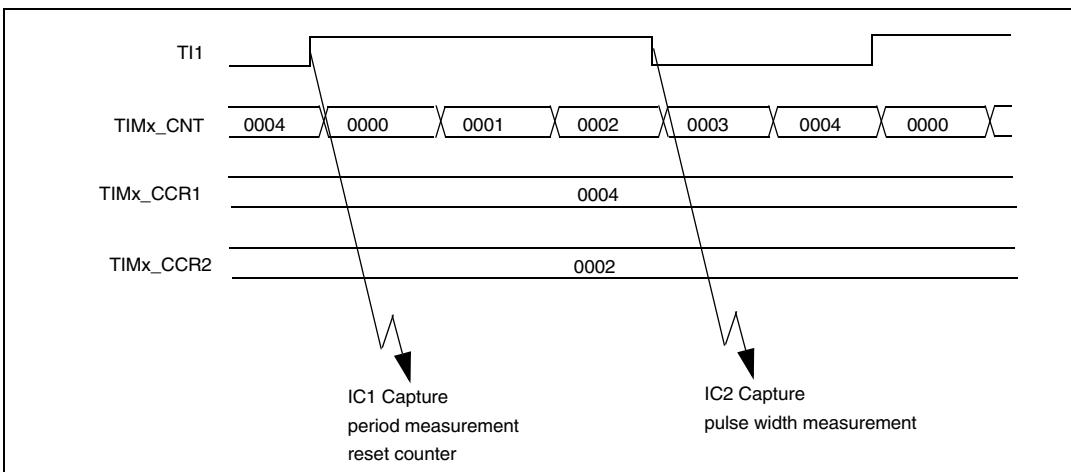
This mode is a particular case of input capture mode. The procedure is the same except:

- Two IC_x signals are mapped on the same TI_x input.
- These 2 IC_x signals are active on edges with opposite polarity.
- One of the two TI_xFP signals is selected as trigger input and the slave mode controller is configured in reset mode.

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

- Select the active input for TIMx_CCR1: write the CC1S bits to 01 in the TIMx_CCMR1 register (TI1 selected).
- Select the active polarity for TI1FP1 (used both for capture in TIMx_CCR1 and counter clear): write the CC1P bit to '0' (active on rising edge).
- Select the active input for TIMx_CCR2: write the CC2S bits to 10 in the TIMx_CCMR1 register (TI1 selected).
- Select the active polarity for TI1FP2 (used for capture in TIMx_CCR2): write the CC2P bit to '1' (active on falling edge).
- Select the valid trigger input: write the TS bits to 101 in the TIMx_SMCR register (TI1FP1 selected).
- Configure the slave mode controller in reset mode: write the SMS bits to 100 in the TIMx_SMCR register.
- Enable the captures: write the CC1E and CC2E bits to '1' in the TIMx_CCER register.

Figure 101. PWM input mode timing.



13.4.7 Forced output mode

In output mode (CC_xS bits = 00 in the TIMx_CCMRx register), each output compare signal (OC_xREF and then OC_x) 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/OC_x) to its active level, you just need to write 101 in the OC_xM bits in the corresponding TIMx_CCMRx register. Thus ocxref is forced high (OC_xREF is always active high) and OC_x get opposite value to CC_xP polarity bit.

e.g.: CCxP=0 (OCx active high) => OCx is forced to high level.

ocxref signal can be forced low by writing the OCxM bits to 100 in the TIMx_CCMRx register.

Anyway, the comparison between the TIMx_CCRx shadow register and the counter is still performed and allows the flag to be set. Interrupt and DMA requests can be sent accordingly. This is described in the Output Compare Mode section.

13.4.8 Output compare mode

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

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

- Assigns the corresponding output pin to a programmable value defined by the output compare mode (OCxM bits in the TIMx_CCMRx register) and the output polarity (CCxP bit in the TIMx_CCER register). The output pin can keep its level (OCXM=000), be set active (OCxM=001), be set inactive (OCxM=010) or can toggle (OCxM=011) on match.
- Sets a flag in the interrupt status register (CCxIF bit in the TIMx_SR register).
- Generates an interrupt if the corresponding interrupt mask is set (CCxIE bit in the TIMx_DIER register).
- Sends a DMA request if the corresponding enable bit is set (CCxDE bit in the TIMx_DIER register, CCDS bit in the TIMx_CR2 register for the DMA request selection).

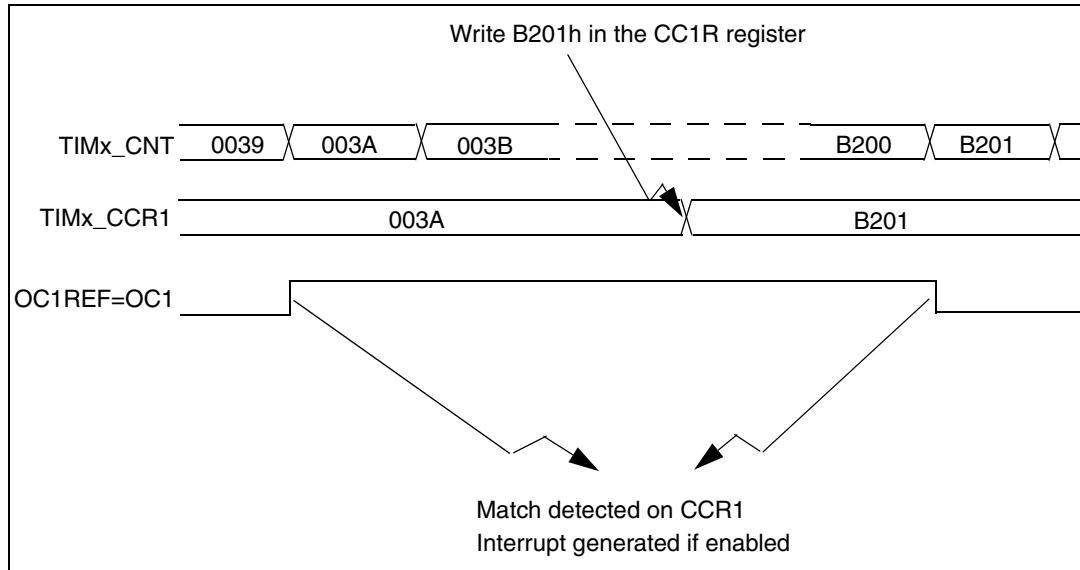
The TIMx_CCRx registers can be programmed with or without preload registers using the OCxPE bit in the TIMx_CCMRx register.

In output compare mode, the update event UEV has no effect on ocxref and OCx output. The timing resolution is one count of the counter. Output compare mode can also be used to output a single pulse (in One Pulse Mode).

Procedure:

1. Select the counter clock (internal, external, prescaler).
2. Write the desired data in the TIMx_ARR and TIMx_CCRx registers.
3. Set the CCxIE and/or CCxDE bits if an interrupt and/or a DMA request is to be generated.
4. Select the output mode. For example, you must write OCxM='011', OCxPE='0', CCxP='0' and CCxE='1' to toggle OCx output pin when CNT matches CCRx, CCRx preload is not used, OCx is enabled and active high.
5. Enable the counter by setting the CEN bit in the TIMx_CR1 register.

The TIMx_CCRx register can be updated at any time by software to control the output waveform, provided that the preload register is not enabled (OCxPE='0', else TIMx_CCRx shadow register is updated only at the next update event UEV). An example is given in [Figure 102](#).

Figure 102. Output compare mode, toggle on OC1.

13.4.9 PWM mode

Pulse Width Modulation mode allows you to generate a signal with a frequency determined by the value of the TIMx_ARR register and a duty cycle determined by the value of the TIMx_CC Rx register.

The PWM mode can be selected independently on each channel (one PWM per OCx output) by writing '110' (PWM mode 1) or '111' (PWM mode 2) in the OCxM bits in the TIMx_CCMRx register. You must enable the corresponding preload register by setting the OCxPE bit in the TIMx_CCMRx register, and eventually the auto-reload preload register (in upcounting or center-aligned modes) by setting the ARPE bit in the TIMx_CR1 register.

As the preload registers are transferred to the shadow registers only when an update event occurs, before starting the counter, you have to initialize all the registers by setting the UG bit in the TIMx_EGR register.

OCx polarity is software programmable using the CCxP bit in the TIMx_CCER register. It can be programmed as active high or active low. OCx output is enabled by the CCxE bit in the TIMx_CCER register. Refer to the TIMx_CCERx register description for more details.

In PWM mode (1 or 2), TIMx_CNT and TIMx_CC Rx are always compared to determine whether TIMx_CC Rx \leq TIMx_CNT or TIMx_CNT \leq TIMx_CC Rx (depending on the direction of the counter). However, to comply with the OCREF_CLR functionality (OCREF can be cleared by an external event through the ETR signal until the next PWM period), the OCREF signal is asserted only:

- When the result of the comparison changes, or
- When the output compare mode (OCxM bits in TIMx_CCMRx register) switches from the “frozen” configuration (no comparison, OCxM='000') to one of the PWM modes (OCxM='110' or '111').

This allows to force the PWM by software while running.

The timer is able to generate PWM in edge-aligned mode or center-aligned mode depending on the CMS bits in the TIMx_CR1 register.

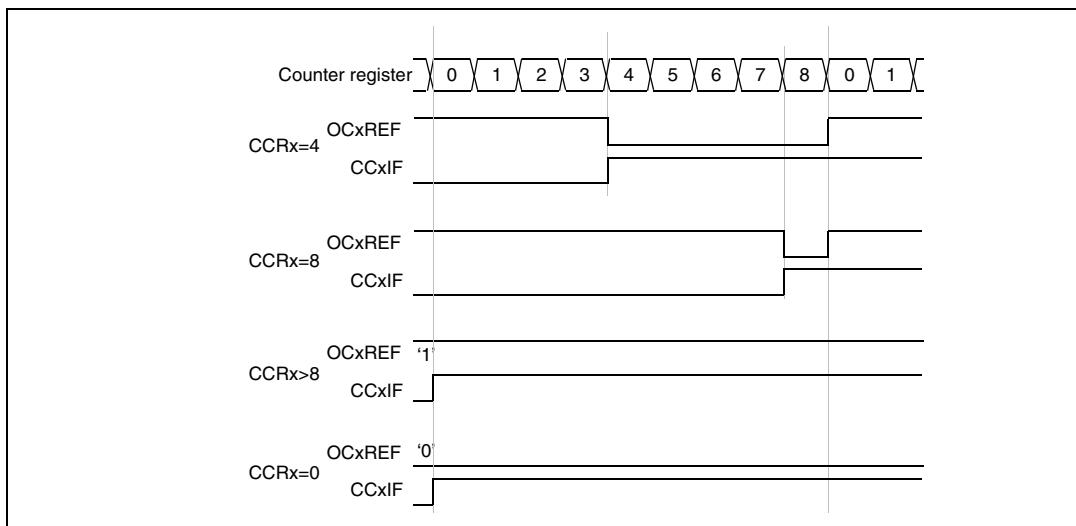
PWM edge-aligned mode

Upcounting configuration

Upcounting is active when the DIR bit in the TIMx_CR1 register is low. Refer to the [Section : upcounting mode on page 218](#).

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 103](#) shows some edge-aligned PWM waveforms in an example where TIMx_ARR=8.

Figure 103. Edge-aligned PWM waveforms (ARR=8)



Downcounting configuration

Downcounting is active when DIR bit in TIMx_CR1 register is high. Refer to [Downcounting mode on page 221](#)

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.

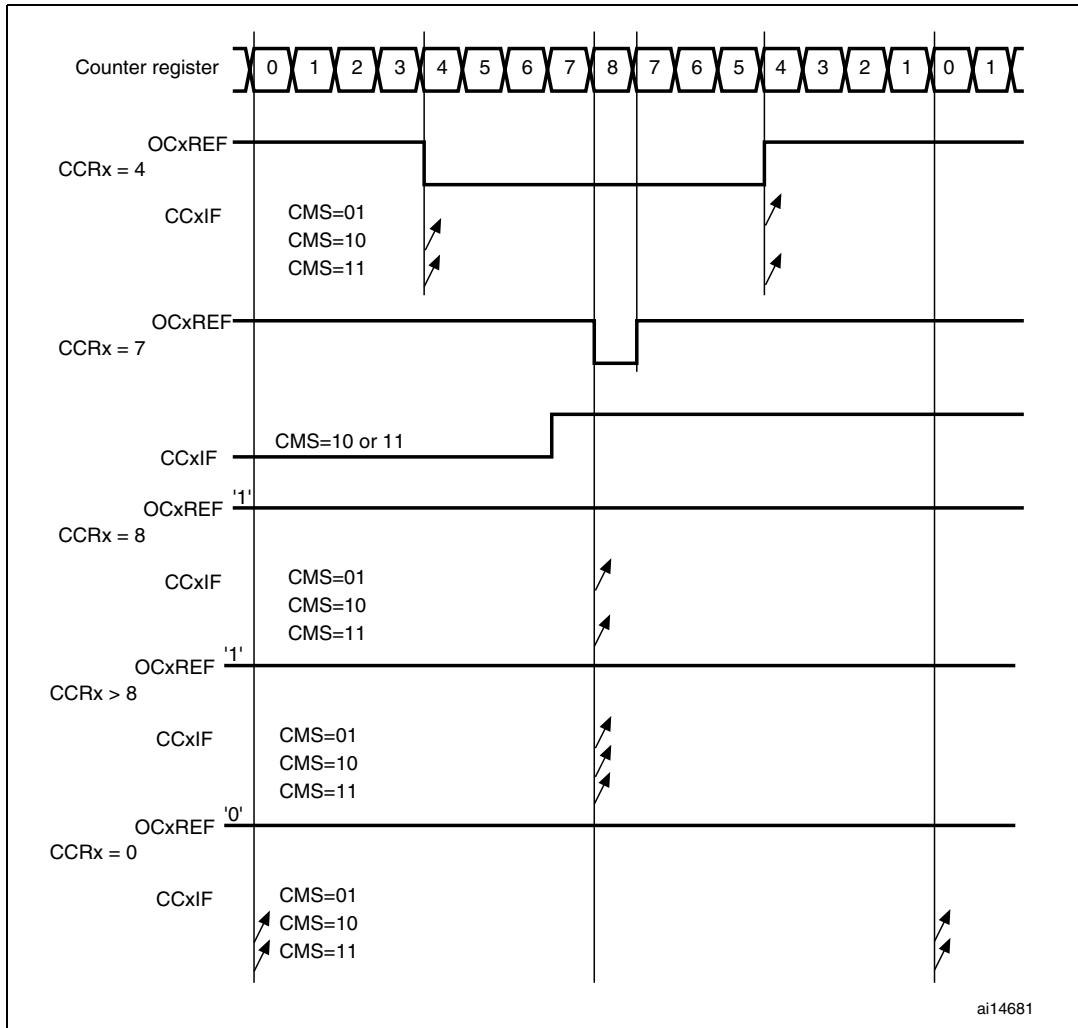
PWM center-aligned mode

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 223](#).

[Figure 104](#) 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 104. Center-aligned PWM waveforms (ARR=8)



ai14681

Hints on using center-aligned mode:

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

13.4.10 One pulse mode

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

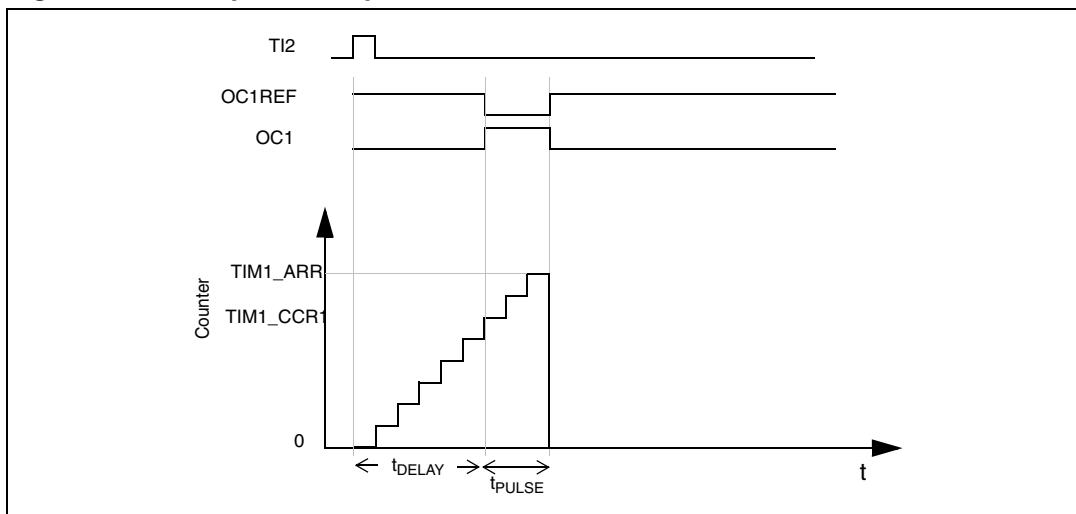
Starting the counter can be controlled through the slave mode controller. Generating the waveform can be done in output compare mode or PWM mode. You select One Pulse Mode by setting the OPM bit in the TIMx_CR1 register. This makes the counter stop automatically at the next update event UEV.

A pulse can be correctly generated only if the compare value is different from the counter initial value. Before starting (when the timer is waiting for the trigger), the configuration must be:

In upcounting: CNT<CCR_x&ARR (in particular, 0<CCR_x),

In downcounting: CNT>CCR_x.

Figure 105. Example of one pulse mode.



For example you may want to generate a positive pulse on OC1 with a length of t_{PULSE} and after a delay of t_{DELAY} as soon as a positive edge is detected on the TI2 input pin.

Let's use TI2FP2 as trigger 1:

- Map TI2FP2 on TI2 by writing IC2S='01' in the TIMx_CCMR1 register.
- TI2FP2 must detect a rising edge, write CC2P='0' in the TIMx_CCER register.
- Configure TI2FP2 as trigger for the slave mode controller (TRGI) by writing TS='110' in the TIMx_SMCR register.
- TI2FP2 is used to start the counter by writing SMS to '110' in the TIMx_SMCR register (trigger mode).

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

- The t_{DELAY} is defined by the value written in the TIMx_CCR1 register.
- The t_{PULSE} is defined by the difference between the auto-reload value and the compare value (TIMx_ARR - TIMx_CCR1).
- Let's say you want to build a waveform with a transition from '0' to '1' when a compare match occurs and a transition from '1' to '0' when the counter reaches the auto-reload value. To do this you enable PWM mode 2 by writing OC1M=111 in the TIMx_CCMR1 register. You can optionally enable the preload registers by writing OC1PE='1' in the TIMx_CCMR1 register and ARPE in the TIMx_CR1 register. In this case you have to write the compare value in the TIMx_CCR1 register, the auto-reload value in the TIMx_ARR register, generate an update by setting the UG bit and wait for external trigger event on TI2. CC1P is written to '0' in this example.

In our example, the DIR and CMS bits in the TIMx_CR1 register should be low.

You only want 1 pulse, so you write '1' in the OPM bit in the TIMx_CR1 register to stop the counter at the next update event (when the counter rolls over from the auto-reload value back to 0).

Particular case: OC_x fast enable:

In One Pulse Mode, the edge detection on 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 you want to output a waveform with the minimum delay, you can set the OC_xFE bit in the TIM_x_CCMR_x register. Then OC_xRef (and OC_x) is forced in response to the stimulus, without taking in account the comparison. Its new level is the same as if a compare match had occurred. OC_xFE acts only if the channel is configured in PWM1 or PWM2 mode.

13.4.11 Clearing the OC_xREF signal on an external event

The OC_xREF signal for a given channel can be reset by applying a High level on the ETRF input (OC_xCE enable bit of the corresponding TIM_x_CCMR_x register set to '1'). The OC_xREF remains low until the next update event, UEV, occurs.

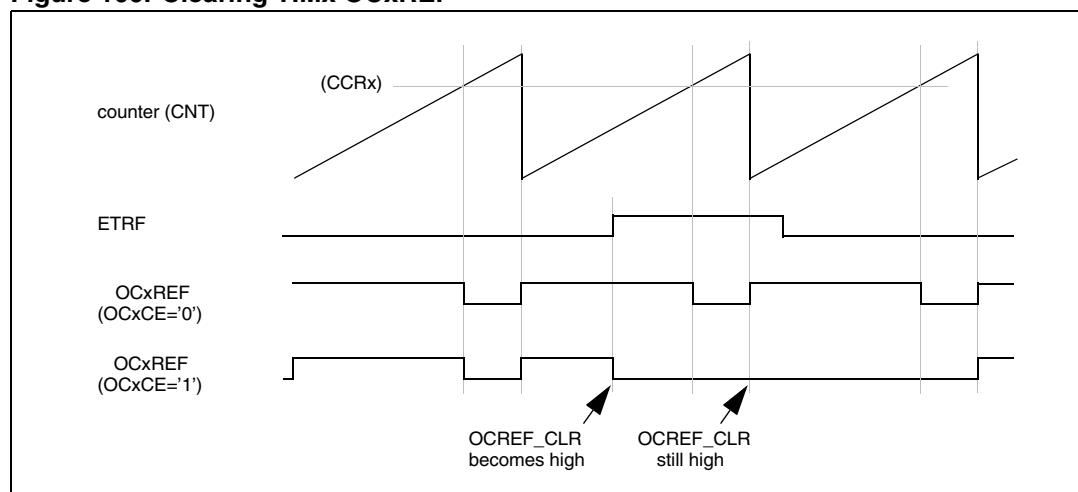
This function can be only used in output compare mode and PWM mode. It does not work in forced mode.

For example, the OC_xREF 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 TIM_x_SMCR register set to '00'.
2. The external clock mode 2 must be disabled: bit ECE of the TIM1_SMCR register set to '0'.
3. The external trigger polarity (ETP) and the external trigger filter (ETF) can be configured according to the user needs.

Figure 106 shows the behavior of the OC_xREF signal when the ETRF Input becomes High, for both values of the enable bit OC_xCE. In this example, the timer TIM_x is programmed in PWM mode.

Figure 106. Clearing TIM_x OC_xREF



13.4.12 Encoder interface mode

To select Encoder Interface mode write SMS='001' in the TIMx_SMCR register if the counter is counting on TI2 edges only, SMS='010' if it is counting on TI1 edges only and SMS='011' if it is counting on both TI1 and TI2 edges.

Select the TI1 and TI2 polarity by programming the CC1P and CC2P bits in the TIMx_CCER register. When needed, you can program the input filter as well.

The two inputs TI1 and TI2 are used to interface to an incremental encoder. Refer to [Table 40](#). The counter is clocked by each valid transition on TI1FP1 or TI2FP2 (TI1 and TI2 after input filter and polarity selection, TI1FP1=TI1 if not filtered and not inverted, TI2FP2=TI2 if not filtered and not inverted) assuming that it is enabled (CEN bit in TIMx_CR1 register written to '1'). The sequence of transitions of the two inputs is evaluated and generates count pulses as well as the direction signal. Depending on the sequence the counter counts up or down, the DIR bit in the TIMx_CR1 register is modified by hardware accordingly. The DIR bit is calculated at each transition on any input (TI1 or TI2), whatever the counter is counting on TI1 only, TI2 only or both TI1 and TI2.

Encoder interface mode acts simply as an external clock with direction selection. This means that the counter just counts continuously between 0 and the auto-reload value in the TIMx_ARR register (0 to ARR or ARR down to 0 depending on the direction). So you must configure TIMx_ARR before starting. In the same way, the capture, compare, prescaler, trigger output features continue to work as normal.

In this mode, the counter is modified automatically following the speed and the direction of the incremental encoder and its content, therefore, always represents the encoder's position. The count direction correspond to the rotation direction of the connected sensor. The table summarizes the possible combinations, assuming TI1 and TI2 don't switch at the same time.

Table 40. Counting direction versus encoder signals

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

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

The [Figure 107](#) gives an example of counter operation, showing count signal generation and direction control. It also shows how input jitter is compensated where both edges are

selected. This might occur if the sensor is positioned near to one of the switching points. For this example we assume that the configuration is the following:

- CC1S='01' (TIMx_CCMR1 register, IC1FP1 mapped on TI1).
- CC2S='01' (TIMx_CCMR2 register, IC2FP2 mapped on TI2).
- CC1P='0' (TIMx_CCER register, IC1FP1 non-inverted, IC1FP1=TI1).
- CC2P='0' (TIMx_CCER register, IC2FP2 non-inverted, IC2FP2=TI2).
- SMS='011' (TIMx_SMCR register, both inputs are active on both rising and falling edges).
- CEN='1' (TIMx_CR1 register, Counter is enabled).

Figure 107. Example of counter operation in encoder interface mode.

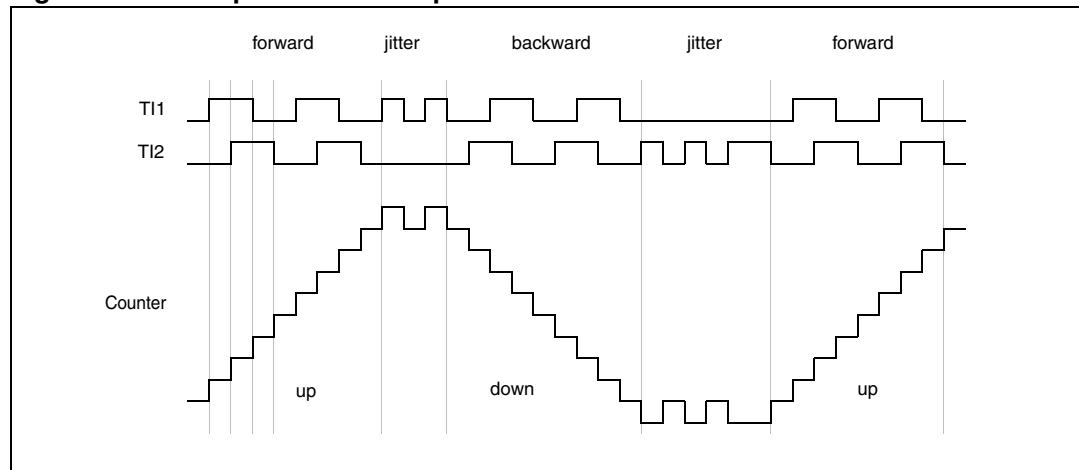
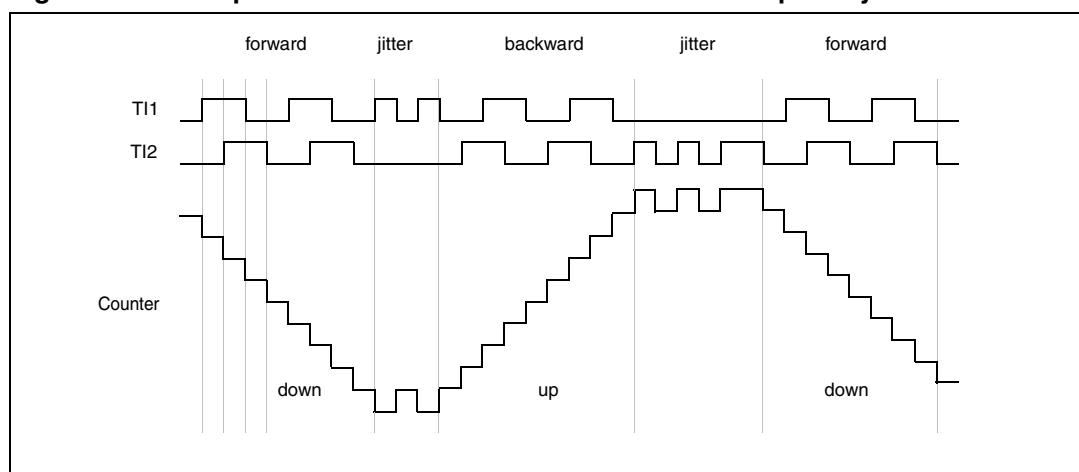


Figure 108 gives an example of counter behavior when IC1FP1 polarity is inverted (same configuration as above except CC1P='1').

Figure 108. Example of encoder interface mode with IC1FP1 polarity inverted.



The timer, when configured in Encoder Interface mode provides information on the sensor's current position. You can obtain dynamic information (speed, acceleration, deceleration) by measuring the period between two encoder events using a second timer configured in capture mode. The output of the encoder which indicates the mechanical zero can be used for this purpose. Depending on the time between two events, the counter can also be read

at regular times. You can do this by latching the counter value into a third input capture register if available (then the capture signal must be periodic and can be generated by another timer). When available, it is also possible to read its value through a DMA request generated by a Real-Time clock.

13.4.13 Timer input XOR function

The TI1S bit in the TIM1_CR2 register, allows the input filter of channel 1 to be connected to the output of a XOR gate, combining the three input pins TIMx_CH1 to TIMx_CH3.

The XOR output can be used with all the timer input functions such as trigger or input capture.

An example of this feature used to interface Hall sensors is given in [Section 12.4.18 on page 182](#).

13.4.14 Timers and external trigger synchronization

The TIMx Timers can be synchronized with an external trigger in several modes: Reset mode, Gated mode and Trigger mode.

Slave mode: Reset mode

The counter and its prescaler can be reinitialized in response to an event on a trigger input. Moreover, if the URS bit from the TIMx_CR1 register is low, an update event UEV is generated. Then all the preloaded registers (TIMx_ARR, TIMx_CCRx) are updated.

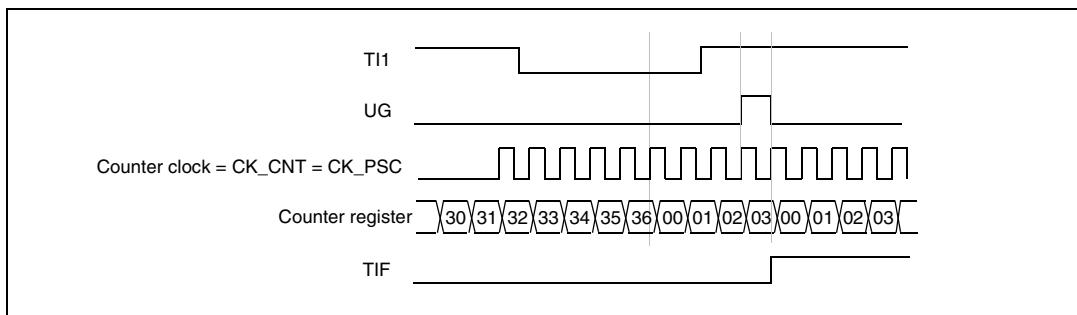
In the following example, the upcounter is cleared in response to a rising edge on TI1 input:

- Configure the channel 1 to detect rising edges on TI1. Configure the input filter duration (in this example, we don't need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so you don't need to configure it. The CC1S bits select the input capture source only, CC1S = 01 in the TIMx_CCMR1 register. Write CC1P=0 in TIMx_CCER register to validate the polarity (and detect rising edges only).
- Configure the timer in reset mode by writing SMS=100 in TIMx_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx_SMCR register.
- Start the counter by writing CEN=1 in the TIMx_CR1 register.

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

The following figure shows this behavior when the auto-reload register TIMx_ARR=0x36. The delay between the rising edge on TI1 and the actual reset of the counter is due to the resynchronization circuit on TI1 input.

Figure 109. Control circuit in reset mode



Slave mode: Gated mode

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

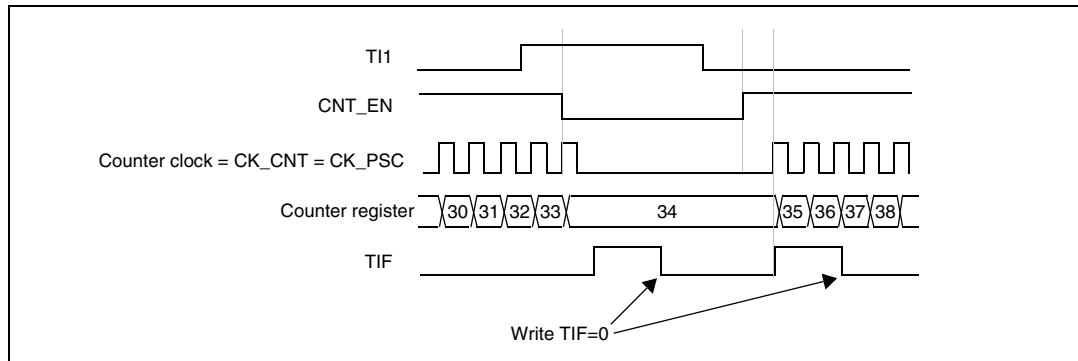
In the following example, the upcounter counts only when TI1 input is low:

- Configure the channel 1 to detect low levels on TI1. Configure the input filter duration (in this example, we don't need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so you don't need to configure it. The CC1S bits select the input capture source only, CC1S=01 in TIMx_CCMR1 register. Write CC1P=1 in TIMx_CCER register to validate the polarity (and detect low level only).
- Configure the timer in gated mode by writing SMS=101 in TIMx_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx_SMCR register.
- Enable the counter by writing CEN=1 in the TIMx_CR1 register (in gated mode, the counter doesn't start if CEN=0, whatever is the trigger input level).

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

The delay between the rising edge on TI1 and the actual stop of the counter is due to the resynchronization circuit on TI1 input.

Figure 110. Control circuit in gated mode



Slave mode: Trigger mode

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

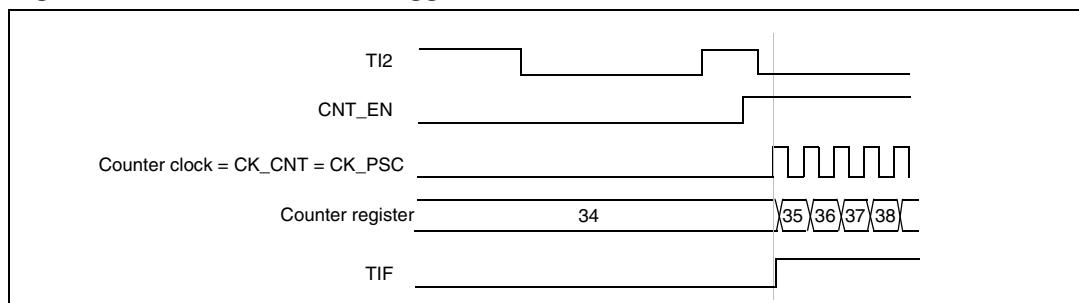
In the following example, the upcounter starts in response to a rising edge on TI2 input:

- Configure the channel 2 to detect rising edges on TI2. Configure the input filter duration (in this example, we don't need any filter, so we keep IC2F=0000). The capture prescaler is not used for triggering, so you don't need to configure it. CC2S bits are selecting the input capture source only, CC2s=01 in TIMx_CCMR1 register. Write CC2P=1 in TIMx_CCER register to validate the polarity (and detect low level only).
- Configure the timer in trigger mode by writing SMS=110 in TIMx_SMCR register. Select TI2 as the input source by writing TS=110 in TIMx_SMCR register.

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

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

Figure 111. 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.

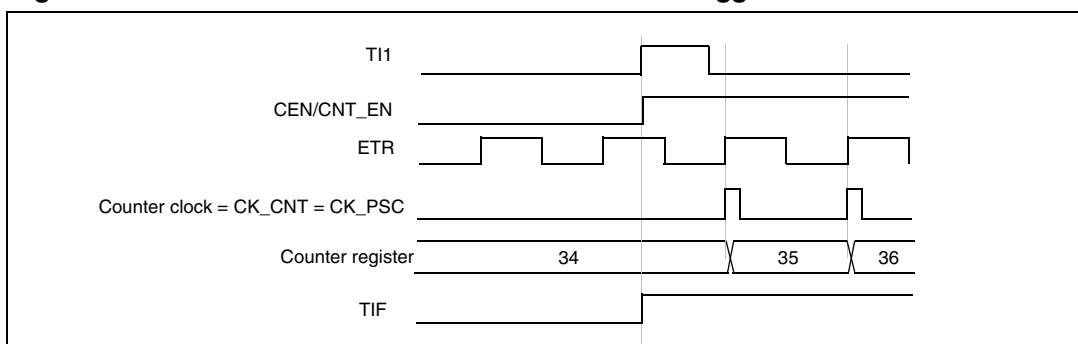
In the following example, the upcounter is incremented at each rising edge of the ETR signal as soon as a rising edge of TI1 occurs:

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=01 in TIMx_CCMR1 register to select only the input capture source
 - CC1P=0 in TIMx_CCER register to validate the polarity (and detect rising edge only).
3. Configure the timer in trigger mode by writing SMS=110 in TIMx_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx_SMCR register.

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

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

Figure 112. Control circuit in external clock mode 2 + trigger mode



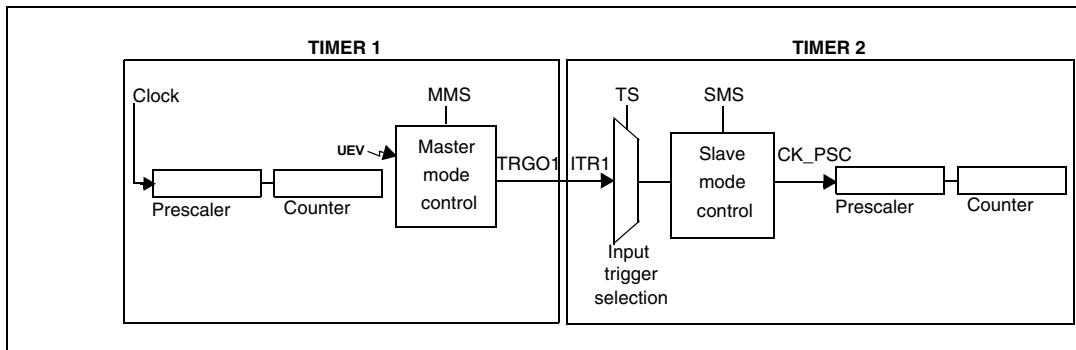
13.4.15 Timer synchronization

The TIMx timers are linked together internally for timer synchronization or chaining. When one Timer is configured in Master Mode, it can reset, start, stop or clock the counter of another Timer configured in Slave Mode.

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

Using one timer as prescaler for the another

Figure 113. Master/Slave timer example



For example, you can configure Timer 1 to act as a prescaler for Timer 2. Refer to [Figure 113](#). To do this:

- Configure Timer 1 in master mode so that it outputs a periodic trigger signal on each update event UEV. If you write MMS=010 in the TIM1_CR2 register, a rising edge is output on TRGO1 each time an update event is generated.
- To connect the TRGO1 output of Timer 1 to Timer 2, Timer 2 must be configured in slave mode using ITR1 as internal trigger. You select this through the TS bits in the TIM2_SMCR register (writing TS=001).
- Then you put the slave mode controller in external clock mode 1 (write SMS=111 in the TIM2_SMCR register). This causes Timer 2 to be clocked by the rising edge of the periodic Timer 1 trigger signal (which correspond to the timer 1 counter overflow).
- Finally both timers must be enabled by setting their respective CEN bits (TIMx_CR1 register).

Note: If OCx is selected on Timer 1 as trigger output (MMS=1xx), its rising edge is used to clock the counter of timer 2.

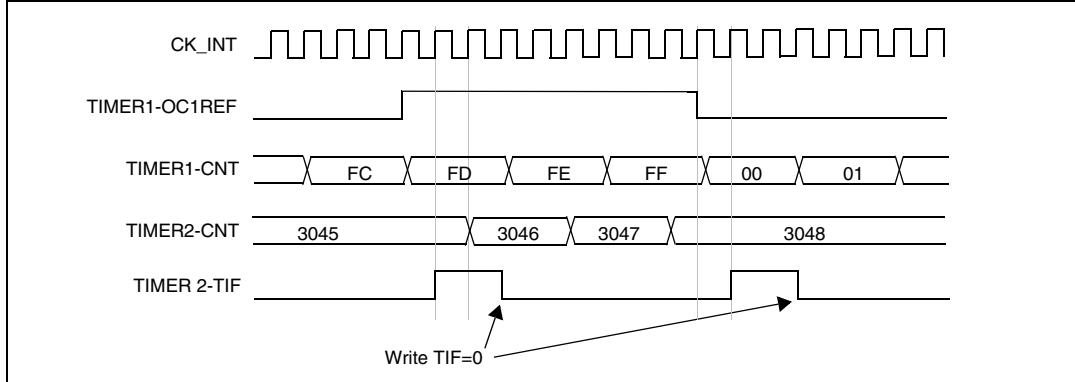
Using one timer to enable another timer

In this example, we control the enable of Timer 2 with the output compare 1 of Timer 1. Refer to [Figure 113](#) for connections. Timer 2 counts on the divided internal clock only when OC1REF of Timer 1 is high. Both counter clock frequencies are divided by 3 by the prescaler compared to CK_INT ($f_{CK_CNT} = f_{CK_INT}/3$).

- Configure Timer 1 master mode to send its Output Compare 1 Reference (OC1REF) signal as trigger output (MMS=100 in the TIM1_CR2 register).
- Configure the Timer 1 OC1REF waveform (TIM1_CCMR1 register).
- Configure Timer 2 to get the input trigger from Timer 1 (TS=001 in the TIM2_SMCR register).
- Configure Timer 2 in gated mode (SMS=101 in TIM2_SMCR register).
- Enable Timer 2 by writing '1' in the CEN bit (TIM2_CR1 register).
- Start Timer 1 by writing '1' in the CEN bit (TIM1_CR1 register).

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

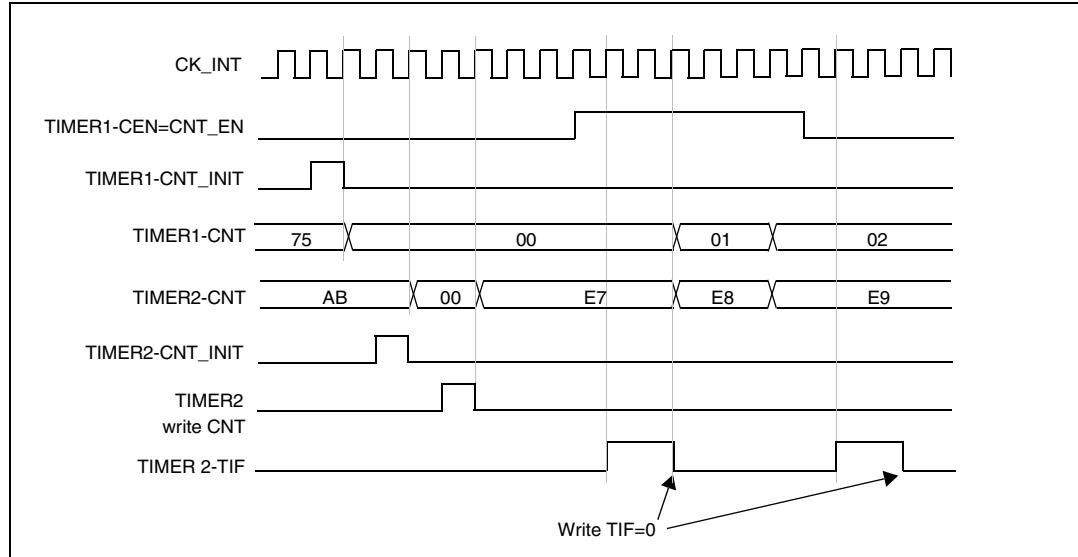
Figure 114. Gating timer 2 with OC1REF of timer 1



In the example in [Figure 114](#), the Timer 2 counter and prescaler are not initialized before being started. So they start counting from their current value. It is possible to start from a given value by resetting both timers before starting Timer 1. You can then write any value you want in the timer counters. The timers can easily be reset by software using the UG bit in the TIMx_EGR registers.

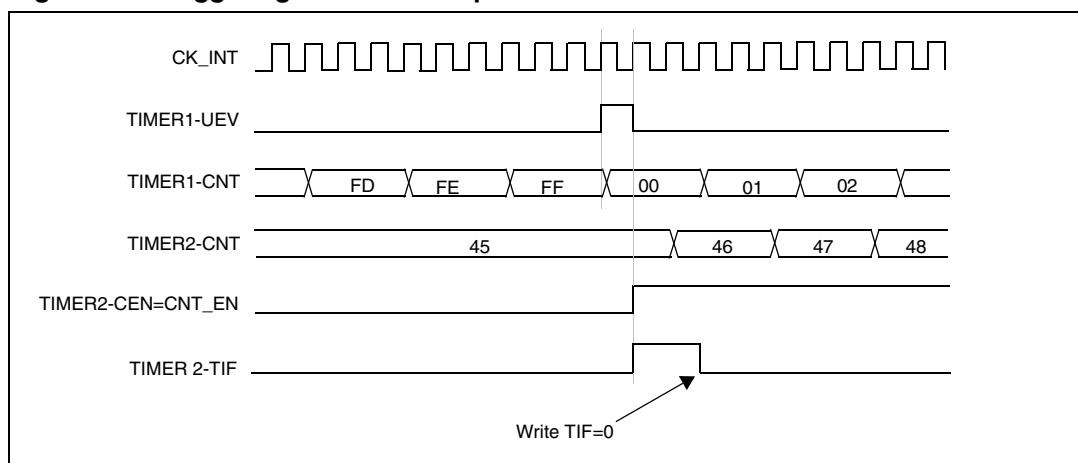
In the next example, we synchronize Timer 1 and Timer 2. Timer 1 is the master and starts from 0. Timer 2 is the slave and starts from 0xE7. The prescaler ratio is the same for both timers. Timer 2 stops when Timer 1 is disabled by writing '0' to the CEN bit in the TIM1_CR1 register:

- Configure Timer 1 master mode to send its Output Compare 1 Reference (OC1REF) signal as trigger output (MMS=100 in the TIM1_CR2 register).
- Configure the Timer 1 OC1REF waveform (TIM1_CCMR1 register).
- Configure Timer 2 to get the input trigger from Timer 1 (TS=001 in the TIM2_SMCR register).
- Configure Timer 2 in gated mode (SMS=101 in TIM2_SMCR register).
- Reset Timer 1 by writing '1' in UG bit (TIM1_EGR register).
- Reset Timer 2 by writing '1' in UG bit (TIM2_EGR register).
- Initialize Timer 2 to 0xE7 by writing '0xE7' in the timer 2 counter (TIM2_CNTL).
- Enable Timer 2 by writing '1' in the CEN bit (TIM2_CR1 register).
- Start Timer 1 by writing '1' in the CEN bit (TIM1_CR1 register).
- Stop Timer 1 by writing '0' in the CEN bit (TIM1_CR1 register).

Figure 115. Gating timer 2 with Enable of timer 1**Using one timer to start another timer**

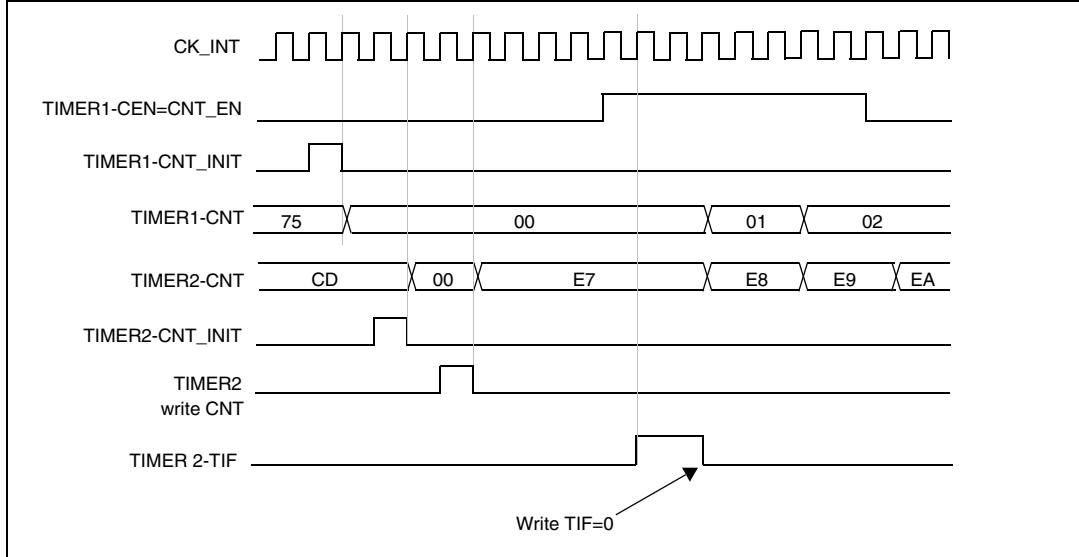
In this example, we set the enable of Timer 2 with the update event of Timer 1. Refer to [Figure 113](#) for connections. Timer 2 starts counting from its current value (which can be non-zero) on the divided internal clock as soon as the update event is generated by Timer 1. When Timer 2 receives the trigger signal its CEN bit is automatically set and the counter counts until we write '0' to the CEN bit in the TIM2_CR1 register. Both counter clock frequencies are divided by 3 by the prescaler compared to CK_INT ($f_{CK_CNT} = f_{CK_INT}/3$).

- Configure Timer 1 master mode to send its Update Event (UEV) as trigger output (MMS=010 in the TIM1_CR2 register).
- Configure the Timer 1 period (TIM1_ARR registers).
- Configure Timer 2 to get the input trigger from Timer 1 (TS=001 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 116. Triggering timer 2 with Update of timer 1

As in the previous example, you can initialize both counters before starting counting. [Figure 117](#) shows the behavior with the same configuration as in [Figure 116](#) but in trigger mode instead of gated mode (SMS=110 in the TIM2_SMCR register).

Figure 117. Triggering timer 2 with Enable of timer 1



Using one timer as prescaler for another timer

For example, you can configure Timer 1 to act as a prescaler for Timer 2. Refer to [Figure 113](#) for connections. To do this:

- Configure Timer 1 master mode to send its Update Event (UEV) as trigger output (MMS=010 in the TIM1_CR2 register). then it outputs a periodic signal on each counter overflow.
- Configure the Timer 1 period (TIM1_ARR registers).
- Configure Timer 2 to get the input trigger from Timer 1 (TS=001 in the TIM2_SMCR register).
- Configure Timer 2 in external clock mode 1 (SMS=111 in TIM2_SMCR register).
- Start Timer 2 by writing '1' in the CEN bit (TIM2_CR1 register).
- Start Timer 1 by writing '1' in the CEN bit (TIM1_CR1 register).

Starting 2 timers synchronously in response to an external trigger

In this example, we set the enable of timer 1 when its TI1 input rises, and the enable of Timer 2 with the enable of Timer 1. Refer to [Figure 113](#) 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):

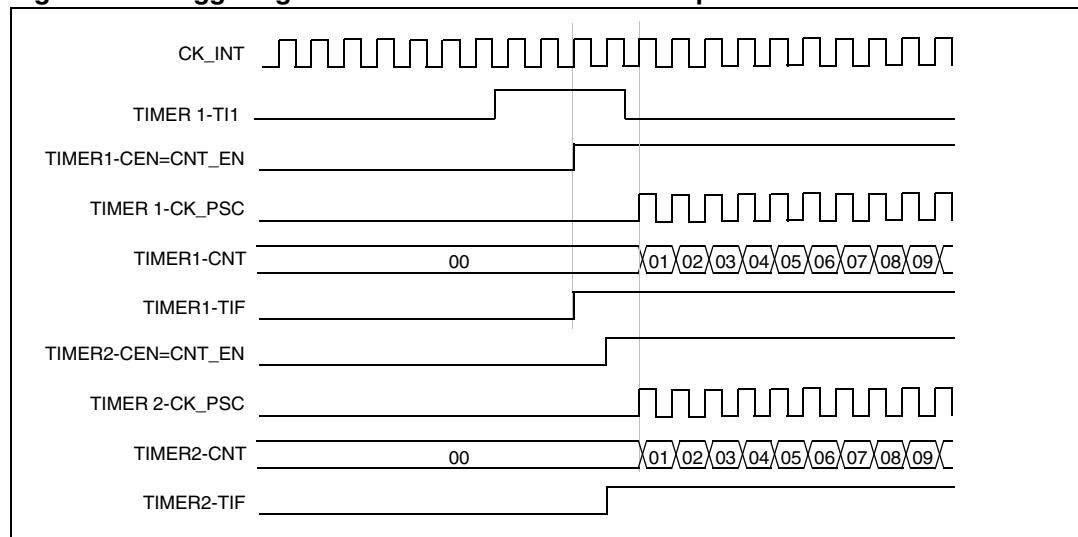
- Configure Timer 1 master mode to send its Enable as trigger output (MMS=001 in the TIM1_CR2 register).
- Configure Timer 1 slave mode to get the input trigger from TI1 (TS=100 in the TIM1_SMCR register).
- Configure Timer 1 in trigger mode (SMS=110 in the TIM1_SMCR register).
- Configure the Timer 1 in Master/Slave mode by writing MSM='1' (TIM1_SMCR register).
- Configure Timer 2 to get the input trigger from Timer 1 (TS=001 in the TIM2_SMCR register).
- Configure Timer 2 in trigger mode (SMS=110 in the TIM2_SMCR register).

When a rising edge occurs on TI1 (Timer 1), both counters starts counting synchronously on the internal clock and both TIF flags are set.

Note:

In this example both timers are initialized before starting (by setting their respective UG bits). Both counters starts from 0, but you can easily insert an offset between them by writing any of the counter registers (TIMx_CNT). You can see that the master/slave mode insert a delay between CNT_EN and CK_PSC on timer 1.

Figure 118. Triggering timer 1 and 2 with timer 1 TI1 input.



13.4.16 Debug mode

When the microcontroller enters debug mode (Cortex-M3 core - halted), the TIMx counter either continues to work normally or stops, depending on DBG_TIMx_STOP configuration bit in DBG module. For more details, refer to [Section 20.15.2: Debug support for timers, watchdog, bxCAN and I2C](#).

13.5 TIMx register description

Refer to [Section 1.1 on page 23](#) for a list of abbreviations used in register descriptions.

13.5.1 Control register 1 (TIMx_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN		
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		

Bits 15:10 Reserved, always read as 0

Bits 9:8 **CKD**: *Clock Division*.

This bit-field indicates the division ratio between the timer clock (CK_INT) frequency and sampling clock used by the digital filters (ETR, TIx),

00: $t_{DTS} = t_{CK_INT}$

01: $t_{DTS} = 2 \times t_{CK_INT}$

10: $t_{DTS} = 4 \times t_{CK_INT}$

11: Reserved

Bit 7 **ARPE**: *Auto-Reload Preload enable*.

0: TIMx_ARR register is not buffered.

1: TIMx_ARR register is buffered.

Bits 6:5 **CMS**: *Center-aligned Mode Selection*.

00: Edge-aligned mode. The counter counts up or down depending on the direction bit (DIR).

01: Center-aligned mode 1. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set only when the counter is counting down.

10: Center-aligned mode 2. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set only when the counter is counting up.

11: Center-aligned mode 3. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set both when the counter is counting up or down.

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

Bit 4 **DIR**: *Direction*.

0: Counter used as upcounter.

1: Counter used as downcounter.

Note: This bit is read only when the timer is configured in Center-aligned mode or Encoder mode.

Bit 3 **OPM**: *One Pulse Mode*.

0: Counter is not stopped at update event

1: Counter stops counting at the next update event (clearing the bit CEN).

Bit 2 URS: *Update Request Source.*

This bit is set and cleared by software to select the UEV event sources.

0: Any of the following events generate an update interrupt or DMA request if enabled. These events can be:

- Counter overflow/underflow
 - Setting the UG bit
 - Update generation through the slave mode controller
- 1: Only counter overflow/underflow generates an update interrupt or DMA request if enabled.

Bit 1 UDIS: *Update Disable.*

This bit is set and cleared by software to enable/disable UEV event generation.

0: UEV enabled. The Update (UEV) event is generated by one of the following events:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

Buffered registers are then loaded with their preload values.

1: UEV disabled. The Update event is not generated, shadow registers keep their value (ARR, PSC, CCRx). However the counter and the prescaler are reinitialized if the UG bit is set or if a hardware reset is received from the slave mode controller.

Bit 0 CEN: *Counter enable.*

- 0: Counter disabled
1: Counter enabled

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

CEN is cleared automatically in one pulse mode, when an update event occurs.

13.5.2 Control register 2 (TIMx_CR2)

Address offset: 0x04

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TI1S	MMS[2:0]			CCDS	Reserved		
								rw	rw	rw	rw	rw			

Bits 15:8 Reserved, always read as 0.

Bit 7 **TI1S**: *TI1 Selection*.

0: The TIMx_CH1 pin is connected to TI1 input.

1: The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

See also [Section 12.4.18: Interfacing with Hall sensors on page 182](#)

Bits 6:4 **MMS**: *Master Mode Selection*.

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

000: **Reset** - the UG bit from the TIMx_EGR register is used as trigger output (TRGO). If the reset is generated by the trigger input (slave mode controller configured in reset mode) then the signal on TRGO is delayed compared to the actual reset.

001: **Enable** - the Counter enable signal, CNT_EN, is used as trigger output (TRGO). It is useful to start several timers at the same time or to control a window in which a slave timer is enabled. The Counter Enable signal is generated by a logic OR between CEN control bit and the trigger input when configured in gated mode.

When the Counter Enable signal is controlled by the trigger input, there is a delay on TRGO, except if the master/slave mode is selected (see the MSM bit description in TIMx_SMCR register).

010: **Update** - The update event is selected as trigger output (TRGO). For instance a master timer can then be used as a prescaler for a slave timer.

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

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

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

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

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

Bit 3 **CCDS**: *Capture/Compare DMA Selection*.

0: CCx DMA request sent when CCx event occurs

1: CCx DMA requests sent when update event occurs

Bits 2:0 Reserved, always read as 0

13.5.3 Slave mode control register (TIMx_SMCR)

Address offset: 0x08

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ETP	ECE	ETPS[1:0]		ETF[3:0]			MSM	TS[2:0]			Res.	SMS[2:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	

Bit 15 **ETP: External Trigger Polarity.**

This bit selects whether ETR or \overline{ETR} is used for trigger operations

0: ETR is non-inverted, active at high level or rising edge.

1: ETR is inverted, active at low level or falling edge.

Bit 14 **ECE: External Clock enable.**

This bit enables External clock mode 2.

0: External clock mode 2 disabled.

1: External clock mode 2 enabled. The counter is clocked by any active edge on the ETRF signal.

Note 1: Setting the ECE bit has the same effect as selecting external clock mode 1 with TRGI connected to ETRF (SMS=111 and TS=111).

Note 2: It is possible to simultaneously use external clock mode 2 with the following slave modes: reset mode, gated mode and trigger mode. Nevertheless, TRGI must not be connected to ETRF in this case (TS bits must not be 111).

Note 3: If external clock mode 1 and external clock mode 2 are enabled at the same time, the external clock input is ETRF.

Bits 13:12 **ETPS: External Trigger Prescaler.**

External trigger signal ETRP frequency must be at most 1/4 of CK_INT frequency. A prescaler can be enabled to reduce ETRP frequency. It is useful when inputting fast external clocks.

00: Prescaler OFF.

01: ETRP frequency divided by 2.

10: ETRP frequency divided by 4.

11: ETRP frequency divided by 8.

Bits 11:8 ETF[3:0]: External Trigger Filter.

This bit-field then defines the frequency used to sample ETRP signal and the length of the digital filter applied to ETRP. The digital filter is made of an event counter in which N events are needed to validate a transition on the output:

- 0000: No filter, sampling is done at f_{DTS} .
- 0001: $f_{SAMPLING}=f_{CK_INT}$, N=2.
- 0010: $f_{SAMPLING}=f_{CK_INT}$, N=4.
- 0011: $f_{SAMPLING}=f_{CK_INT}$, N=8.
- 0100: $f_{SAMPLING}=f_{DTS}/2$, N=6.
- 0101: $f_{SAMPLING}=f_{DTS}/2$, N=8.
- 0110: $f_{SAMPLING}=f_{DTS}/4$, N=6.
- 0111: $f_{SAMPLING}=f_{DTS}/4$, N=8.
- 1000: $f_{SAMPLING}=f_{DTS}/8$, N=6.
- 1001: $f_{SAMPLING}=f_{DTS}/8$, N=8.
- 1010: $f_{SAMPLING}=f_{DTS}/16$, N=5.
- 1011: $f_{SAMPLING}=f_{DTS}/16$, N=6.
- 1100: $f_{SAMPLING}=f_{DTS}/16$, N=8.
- 1101: $f_{SAMPLING}=f_{DTS}/32$, N=5.
- 1110: $f_{SAMPLING}=f_{DTS}/32$, N=6.
- 1111: $f_{SAMPLING}=f_{DTS}/32$, N=8.

Bit 7 MSM: Master/Slave mode.

- 0: No action
- 1: The effect of an event on the trigger input (TRGI) is delayed to allow a perfect synchronization between the current timer and its slaves (through TRGO). It is useful if we want to synchronize several timers on a single external event.

Bits 6:4 TS: Trigger Selection.

This bit-field selects the trigger input to be used to synchronize the counter.

- 000: Internal Trigger 0 (ITR0). TIM1
- 001: Internal Trigger 1 (ITR1). TIM2
- 010: Internal Trigger 2 (ITR2). TIM3
- 011: Internal Trigger 3 (ITR3). TIM4
- 100: TI1 Edge Detector (TI1F_ED).
- 101: Filtered Timer Input 1 (TI1FP1).
- 110: Filtered Timer Input 2 (TI2FP2).
- 111: External Trigger input (ETRF).

Note: These bits must be changed only when they are not used (e.g. when SMS=000) to avoid wrong edge detections at the transition.

Bit 3 Reserved, always read as 0.

Bits 2:0 SMS: Slave Mode Selection.

When external signals are selected the active edge of the trigger signal (TRGI) is linked to the polarity selected on the external input (see Input Control register and Control Register description).

000: Slave mode disabled - if CEN = '1' then the prescaler is clocked directly by the internal clock.

001: Encoder mode 1 - Counter counts up/down on TI2FP2 edge depending on TI1FP1 level.

010: Encoder mode 2 - Counter counts up/down on TI1FP1 edge depending on TI2FP2 level.

011: Encoder mode 3 - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input.

100: Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter and generates an update of the registers.

101: Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high. The counter stops (but is not reset) as soon as the trigger becomes low. Both start and stop of the counter are controlled.

110: Trigger Mode - The counter starts at a rising edge of the trigger TRGI (but it is not reset). Only the start of the counter is controlled.

111: External Clock Mode 1 - Rising edges of the selected trigger (TRGI) clock the counter.

Note: The gated mode must not be used if TI1F_ED is selected as the trigger input (TS='100'). Indeed, TI1F_ED outputs 1 pulse for each transition on TI1F, whereas the gated mode checks the level of the trigger signal.

13.5.4 DMA/Interrupt enable register (TIMx_DIER)

Address offset: 0x0C

Reset value: 0x0000

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res	CC4 DE	CC3 DE	CC2 DE	CC1 DE	UDE	Res.	TIE	Res	CC4IE	CC3IE	CC2IE	CC1IE	UIE	
	rw		rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw	

Bit 15 Reserved, always read as 0.

Bit 14 **TDE**: Trigger DMA request enable.

- 0: Trigger DMA request disabled.
- 1: Trigger DMA request enabled.

Bit 13 Reserved, always read as 0

Bit 12 **CC4DE**: Capture/Compare 4 DMA request enable.

- 0: CC4 DMA request disabled.
- 1: CC4 DMA request enabled.

Bit 11 **CC3DE**: Capture/Compare 3 DMA request enable.

- 0: CC3 DMA request disabled.
- 1: CC3 DMA request enabled.

Bit 10 **CC2DE**: Capture/Compare 2 DMA request enable.

- 0: CC2 DMA request disabled.
- 1: CC2 DMA request enabled.

Bit 9 **CC1DE**: Capture/Compare 1 DMA request enable.

- 0: CC1 DMA request disabled.
- 1: CC1 DMA request enabled.

Bit 8 **UDE**: Update DMA request enable.

- 0: Update DMA request disabled.
- 1: Update DMA request enabled.

Bit 7 Reserved, always read as 0.

Bit 6 **TIE**: Trigger interrupt enable.

- 0: Trigger interrupt disabled.
- 1: Trigger interrupt enabled.

Bit 5 Reserved, always read as 0.

Bit 4 **CC4IE**: Capture/Compare 4 interrupt enable.

- 0: CC4 interrupt disabled.
- 1: CC4 interrupt enabled.

Bit 3 **CC3IE**: Capture/Compare 3 interrupt enable.

- 0: CC3 interrupt disabled.
- 1: CC3 interrupt enabled.

Bit 2 **CC2IE**: Capture/Compare 2 interrupt enable.

- 0: CC2 interrupt disabled.
- 1: CC2 interrupt enabled.

Bit 1 **CC1IE**: Capture/Compare 1 interrupt enable.

- 0: CC1 interrupt disabled.
- 1: CC1 interrupt enabled.

Bit 0 **UIE**: Update interrupt enable.

- 0: Update interrupt disabled.
- 1: Update interrupt enabled.

13.5.5 Status register (TIMx_SR)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		CC4 OF	CC3 OF	CC2 OF	CC1 OF	Reserved	TIF	Res	CC4IF	CC3IF	CC2IF	CC1IF	UIF		
	rc	rc	rc	rc			rc		rc	rc	rc	rc		rc	rc

Bit 15:13 Reserved, always read as 0.

Bit 12 **CC4OF**: Capture/Compare 4 Overcapture Flag.

refer to CC1OF description

Bit 11 **CC3OF**: Capture/Compare 3 Overcapture Flag.

refer to CC1OF description

Bit 10 **CC2OF**: Capture/Compare 2 Overcapture Flag.

refer to CC1OF description

Bit 9 **CC1OF**: Capture/Compare 1 Overcapture Flag.

This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to '0'.

0: No overcapture has been detected.

1: The counter value has been captured in TIMx_CCR1 register while CC1IF flag was already set

Bits 8:7 Reserved, always read as 0.

Bit 6 **TIF**: Trigger interrupt Flag.

This flag is set by hardware on trigger event (active edge detected on TRGI input when the slave mode controller is enabled in all modes but gated mode, both edges in case gated mode is selected). It is cleared by software.

0: No trigger event occurred.

1: Trigger interrupt pending.

Bit 5 Reserved, always read as 0

Bit 4 **CC4IF**: Capture/Compare 4 interrupt Flag.

refer to CC1IF description

Bit 3 **CC3IF**: Capture/Compare 3 interrupt Flag.

refer to CC1IF description

Bit 2 **CC2IF**: Capture/Compare 2 interrupt Flag.

refer to CC1IF description

Bit 1 **CC1IF**: *Capture/compare 1 interrupt Flag.*

If channel CC1 is configured as output:

This flag is set by hardware when the counter matches the compare value, with some exception in center-aligned mode (refer to the CMS bits in the TIMx_CR1 register description). It is cleared by software.

0: No match.

1: The content of the counter TIMx_CNT has matched the content of the TIMx_CCR1 register.

If channel CC1 is configured as input:

This bit is set by hardware on a capture. It is cleared by software or by reading the TIMx_CCR1 register.

0: No input capture occurred.

1: The counter value has been captured in TIMx_CCR1 register (An edge has been detected on IC1 which matches the selected polarity).

Bit 0 **UIF**: *Update interrupt flag.*

This bit is set by hardware on an update event. It is cleared by software.

0: No update occurred.

1: Update interrupt pending. This bit is set by hardware when the registers are updated:

- At overflow or underflow regarding the repetition downcounter value (update if REP_CNT=0) and if the UDIS=0 in the TIMx_CR1 register.
- When CNT is reinitialized by software using the UG bit in TIMx_EGR register, if URS=0 and UDIS=0 in the TIMx_CR1 register.
- When CNT is reinitialized by a trigger event (refer to the synchro control register description), if URS=0 and UDIS=0 in the TIMx_CR1 register.

13.5.6 Event generation register (TIMx_EGR)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
									TG	Res.	CC4G	CC3G	CC2G	CC1G	UG
Reserved								w	Res.	w	w	w	w	w	w

Bits 15:7 Reserved, always read as 0.

Bit 6 **TG**: *Trigger generation*.

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action.

1: The TIF flag is set in TIMx_SR register. Related interrupt or DMA transfer can occur if enabled.

Bit 5 Reserved, always read as 0.

Bit 4 **CC4G**: *Capture/compare 4 generation*.

refer to CC1G description

Bit 3 **CC3G**: *Capture/compare 3 generation*.

refer to CC1G description

Bit 2 **CC2G**: *Capture/compare 2 generation*.

refer to CC1G description

Bit 1 **CC1G**: *Capture/compare 1 generation*.

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action.

1: A capture/compare event is generated on channel 1:

If channel CC1 is configured as output:

CC1IF flag is set, Corresponding interrupt or DMA request is sent if enabled.

If channel CC1 is configured as input:

The current value of the counter is captured in TIMx_CCR1 register. The CC1IF flag is set, the corresponding interrupt or DMA request is sent if enabled. The CC1OF flag is set if the CC1IF flag was already high.

Bit 0 **UG**: *Update generation*.

This bit can be set by software, it is automatically cleared by hardware.

0: No action.

1: Re-initialize the counter and generates an update of the registers. Note that the prescaler counter is cleared too (anyway the prescaler ratio is not affected). The counter is cleared if the center-aligned mode is selected or if DIR=0 (upcounting), else it takes the auto-reload value (TIMx_ARR) if DIR=1 (downcounting).

13.5.7 Capture/compare mode register 1 (TIMx_CCMR1)

Address offset: 0x18

Reset value: 0x0000

The channels can be used in input (capture mode) or in output (compare mode). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OCxx describes its function when the channel is configured in output, ICxx describes its function when the channel is configured in input. So you must take care that the same bit can have a different meaning for the input stage and for the output stage.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
OC2 CE	OC2M[2:0]			OC2 PE	OC2 FE	CC2S[1:0]	OC1 CE	OC1M[2:0]			OC1 PE	OC1 FE	CC1S[1:0]				
IC2F[3:0]			IC2PSC[1:0]				IC1F[3:0]			IC1PSC[1:0]							
rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw		rw	rw		

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.

001: Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIMx_CNT matches the capture/compare register 1 (TIMx_CCR1).

010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx_CNT matches the capture/compare register 1 (TIMx_CCR1).

011: Toggle - OC1REF toggles when TIMx_CNT=TIMx_CCR1.

100: Force inactive level - OC1REF is forced low.

101: Force active level - OC1REF is forced high.

110: PWM mode 1 - In upcounting, channel 1 is active as long as TIMx_CNT<TIMx_CCR1 else inactive. In downcounting, channel 1 is inactive (OC1REF='0') as long as TIMx_CNT>TIMx_CCR1 else active (OC1REF='1').

111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIMx_CNT<TIMx_CCR1 else active. In downcounting, channel 1 is active as long as TIMx_CNT>TIMx_CCR1 else inactive.

Note 1: These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx_BDTR register) and CC1S='00' (the channel is configured in output).

Note 2: In PWM mode 1 or 2, the OCREF level changes only when the result of the comparison changes or when the output compare mode switches from “frozen” mode to “PWM” mode.

Bit 3 OC1PE: Output Compare 1 Preload enable.

0: Preload register on TIMx_CCR1 disabled. TIMx_CCR1 can be written at anytime, the new value is taken in account immediately.

1: Preload register on TIMx_CCR1 enabled. Read/Write operations access the preload register. TIMx_CCR1 preload value is loaded in the active register at each update event.

Note 1: These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx_BDTR register) and CC1S='00' (the channel is configured in output).

Note 2: The PWM mode can be used without validating the preload register only in one pulse mode (OPM bit set in TIMx_CR1 register). Else the behavior is not guaranteed.

Bit 2 OC1FE: Output Compare 1 Fast enable.

This bit is used to accelerate the effect of an event on the trigger in input on the CC output.

0: CC1 behaves normally depending on counter and CCR1 values even when the trigger is ON. The minimum delay to activate CC1 output when an edge occurs on the trigger input is 5 clock cycles.

1: An active edge on the trigger input acts like a compare match on CC1 output. Then, OC is set to the compare level independently from the result of the comparison. Delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. 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 events are needed to validate a transition on the output:

0000: No filter, sampling is done at f_{DTS} .

0001: $f_{SAMPLING} = f_{CK_INT}$, N=2.

0010: $f_{SAMPLING} = f_{CK_INT}$, N=4.

0011: $f_{SAMPLING} = f_{CK_INT}$, N=8.

0100: $f_{SAMPLING} = f_{DTS}/2$, N=6.

0101: $f_{SAMPLING} = f_{DTS}/2$, N=8.

0110: $f_{SAMPLING} = f_{DTS}/4$, N=6.

0111: $f_{SAMPLING} = f_{DTS}/4$, N=8.

1000: $f_{SAMPLING} = f_{DTS}/8$, N=6.

1001: $f_{SAMPLING} = f_{DTS}/8$, N=8.

1010: $f_{SAMPLING} = f_{DTS}/16$, N=5.

1011: $f_{SAMPLING} = f_{DTS}/16$, N=6.

1100: $f_{SAMPLING} = f_{DTS}/16$, N=8.

1101: $f_{SAMPLING} = f_{DTS}/32$, N=5.

1110: $f_{SAMPLING} = f_{DTS}/32$, N=6.

1111: $f_{SAMPLING} = f_{DTS}/32$, N=8.

Note: In current silicon revision, f_{DTS} is replaced in the formula by CK_INT when ICxF[3:0]= 1, 2 or 3.

Bits 3:2 **IC1PSC**: *Input Capture 1 Prescaler*.

This bit-field defines the ratio of the prescaler acting on CC1 input (IC1).

The prescaler is reset as soon as CC1E='0' (TIMx_CCER register).

00: no prescaler, capture is done each time an edge is detected on the capture input.

01: capture is done once every 2 events.

10: capture is done once every 4 events.

11: capture is done once every 8 events.

Bits 1:0 **CC1S**: *Capture/Compare 1 Selection*.

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output.

01: CC1 channel is configured as input, IC1 is mapped on TI1.

10: CC1 channel is configured as input, IC1 is mapped on TI2.

11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC1S bits are writable only when the channel is OFF (CC1E = '0' in TIMx_CCER).

13.5.8 Capture/compare mode register 2 (TIMx_CCMR2)

Address offset: 0x1C

Reset value: 0x0000

Refer to the above CCMR1 register description.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
OC4 CE	OC4M[2:0]			OC4 PE	OC4 FE	CC4S[1:0]	OC3 CE.	OC3M[2:0]			OC3 PE	OC3 FE	CC3S[1:0]			
IC4F[3:0]			IC4PSC[1:0]				IC3F[3:0]			IC3PSC[1:0]						
rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	

Output Compare mode

Bit 15 **OC4CE**: Output Compare 4 Clear Enable

Bits 14:12 **OC4M**: Output Compare 4 Mode.

Bit 11 **OC4PE**: Output Compare 4 Preload enable.

Bit 10 **OC4FE**: Output Compare 4 Fast enable.

Bits 9:8 **CC4S**: Capture/Compare 4 Selection.

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC4 channel is configured as output.

01: CC4 channel is configured as input, IC4 is mapped on TI4.

10: CC4 channel is configured as input, IC4 is mapped on TI3.

11: CC4 channel is configured as input, IC4 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC4S bits are writable only when the channel is OFF (CC4E = '0' in TIMx_CCER).

Bit 7 **OC3CE**: Output Compare 3 Clear Enable

Bits 6:4 **OC3M**: Output Compare 3 Mode.

Bit 3 **OC3PE**: Output Compare 3 Preload enable.

Bit 2 **OC3FE**: Output Compare 3 Fast enable.

Bits 1:0 **CC3S**: Capture/Compare 3 Selection.

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC3 channel is configured as output.

01: CC3 channel is configured as input, IC3 is mapped on TI3.

10: CC3 channel is configured as input, IC3 is mapped on TI4.

11: CC3 channel is configured as input, IC3 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC3S bits are writable only when the channel is OFF (CC3E = '0' in TIMx_CCER).

Input capture mode

Bits 15:12 **IC4F**: *Input Capture 4 Filter.*

Bits 11:10 **IC4PSC**: *Input Capture 4 Prescaler.*

Bits 9:8 **CC4S**: *Capture/Compare 4 Selection.*

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC4 channel is configured as output.

01: CC4 channel is configured as input, IC4 is mapped on TI4.

10: CC4 channel is configured as input, IC4 is mapped on TI3.

11: CC4 channel is configured as input, IC4 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC4S bits are writable only when the channel is OFF (CC4E = '0' in TIMx_CCER).

Bits 7:4 **IC3F**: *Input Capture 3 Filter.*

Bits 3:2 **IC3PSC**: *Input Capture 3 Prescaler.*

Bits 1:0 **CC3S**: *Capture/Compare 3 Selection.*

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC3 channel is configured as output.

01: CC3 channel is configured as input, IC3 is mapped on TI3.

10: CC3 channel is configured as input, IC3 is mapped on TI4.

11: CC3 channel is configured as input, IC3 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC3S bits are writable only when the channel is OFF (CC3E = '0' in TIMx_CCER).

13.5.9 Capture/compare enable register (TIMx_CCER)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	CC4P	CC4E	Reserved	CC3P	CC3E	Reserved	CC2P	CC2E	Reserved	CC1P	CC1E				
	rw	rw		rw	rw										

Bits 15:14 Reserved, always read as 0.

Bit 13 **CC4P**: *Capture/Compare 4 output Polarity.*

refer to CC1P description

Bit 12 **CC4E**: *Capture/Compare 4 output enable.*

refer to CC1E description

Bits 11:10 Reserved, always read as 0.

Bit 9 **CC3P**: *Capture/Compare 3 output Polarity.*

refer to CC1P description

Bit 8 **CC3E**: *Capture/Compare 3 output enable.*

refer to CC1E description

Bits 7:6 Reserved, always read as 0.

Bit 5 **CC2P**: *Capture/Compare 2 output Polarity.*

refer to CC1P description

Bit 4 **CC2E**: *Capture/Compare 2 output enable.*

refer to CC1E description

Bits 3:2 Reserved, always read as 0.

Bit 1 **CC1P**: *Capture/Compare 1 output Polarity.*

CC1 channel configured as output:

0: OC1 active high.

1: OC1 active low.

CC1 channel configured as input:

This bit selects whether IC1 or IC1 is used for trigger or capture operations.

0: non-inverted: capture is done on a rising edge of IC1. When used as external trigger, IC1 is non-inverted.

1: inverted: capture is done on a falling edge of IC1. When used as external trigger, IC1 is inverted.

Bit 0 **CC1E**: *Capture/Compare 1 output enable.*

CC1 channel configured as output:

0: Off - OC1 is not active.

1: On - OC1 signal is output on the corresponding output pin.

CC1 channel configured as input:

This bit determines if a capture of the counter value can actually be done into the input capture/compare register 1 (TIMx_CCR1) or not.

0: Capture disabled.

1: Capture enabled.

Table 41. Output control bit for standard OCx channels

CCxE bit	OCx output state
0	Output Disabled (OCx=0, OCx_EN=0)
1	OCx=OCxREF + Polarity, OCx_EN=1

Note: The state of the external I/O pins connected to the standard OCx channels depends on the OCx channel state and the GPIO and AFIO registers.

13.5.10 Counter (TIMx_CNT)

Address offset: 0x24

Reset value: 0x0000

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

Bits 15:0 **CNT[15:0]: Counter Value.**

13.5.11 Prescaler (TIMx_PSC)

Address offset: 0x28

Reset value: 0x0000

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

Bits 15:0 **PSC[15:0]: Prescaler Value.**

The counter clock frequency CK_CNT is equal to $f_{CK_PSC} / (PSC[15:0] + 1)$.

PSC contains the value to be loaded in the active prescaler register at each update event.

13.5.12 Auto-reload register (TIMx_ARR)

Address offset: 0x2C

Reset value: 0x0000

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

ARR[15:0]: Prescaler Value.

ARR is the value to be loaded in the actual auto-reload register.

Bits 15:0 Refer to the [Section 13.4.1: Time base unit on page 216](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

13.5.13 Capture/compare register 1 (TIMx_CCR1)

Address offset: 0x34

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR1[15:0]: Capture/Compare 1 Value.**

If channel CC1 is configured as output:

CCR1 is the value to be loaded in the actual capture/compare 1 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx_CCMR1 register (bit OC1PE). Else the preload value is copied in the active capture/compare 1 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx_CNT and signaled on OC1 output.

If channel CC1 is configured as input:

CCR1 is the counter value transferred by the last input capture 1 event (IC1).

13.5.14 Capture/compare register 2 (TIMx_CCR2)

Address offset: 0x38

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR2[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR2[15:0]: Capture/Compare 2 Value.**

If channel CC2 is configured as output:

CCR2 is the value to be loaded in the actual capture/compare 2 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx_CCMR2 register (bit OC2PE). Else the preload value is copied in the active capture/compare 2 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx_CNT and signalled on OC2 output.

If channel CC2 is configured as input:

CCR2 is the counter value transferred by the last input capture 2 event (IC2).

13.5.15 Capture/compare register 3 (TIMx_CCR3)

Address offset: 0x3C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR3[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR3[15:0]: Capture/Compare Value.**

If channel CC3 is configured as output:

CCR3 is the value to be loaded in the actual capture/compare 3 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx_CCMR3 register (bit OC3PE). Else the preload value is copied in the active capture/compare 3 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx_CNT and signalled on OC3 output.

If channel CC3 is configured as input:

CCR3 is the counter value transferred by the last input capture 3 event (IC3).

13.5.16 Capture/compare register 4 (TIMx_CCR4)

Address offset: 0x40

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR4[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR4[15:0]: Capture/Compare Value.**

1/ if CC4 channel is configured as output (CC4S bits):

CCR4 is the value to be loaded in the actual capture/compare 4 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx_CCMR4 register (bit OC4PE). Else the preload value is copied in the active capture/compare 4 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx_CNT and signalled on OC4 output.

2/ if CC4 channel is configured as input (CC4S bits in TIMx_CCMR4 register):

CCR4 is the counter value transferred by the last input capture 4 event (IC4).

13.5.17 DMA control register (TIMx_DCR)

Address offset: 0x48

Reset value: 0x0000

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

Bits 15:13 Reserved, always read as 0

Bits 12:8 **DBL[4:0]: DMA Burst Length.**

This 5-bits vector defines the length of DMA transfers in burst mode (the timer recognizes a burst transfer when a read or a write access is done to the TIMx_DMAR address), i.e. the number of bytes to be transferred.

00000: 1 byte,

00001: 2 bytes,

00010: 3 bytes,

...

10001: 18 bytes.

Bits 7:5 Reserved, always read as 0

Bits 4:0 **DBA[4:0]: DMA Base Address.**

This 5-bit vector defines the base-address for DMA transfers in burst mode (when read/write access are done through the TIMx_DMAR address). DBA is defined as an offset starting from the address of the TIMx_CR1 register.

Example:

00000: TIMx_CR1,

00001: TIMx_CR2,

00010: TIMx_SMCR,

...

13.5.18 DMA address for burst mode (TIMx_DMAR)

Address offset: 0x4C

Reset value: 0x0000

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

Bits 15:0 **DMAB[15:0]: DMA register for burst accesses.**

A read or write access to the DMAR register accesses the register located at the address:

"(TIMx_CR1 address) + DBA + (DMA index)" in which:

TIMx_CR1 address is the address of the control register 1,

DBA is the DMA base address configured in the TIMx_DCR register,

DMA index is the offset automatically controlled by the DMA transfer, depending on the length of the transfer DBL in the TIMx_DCR register.

13.6 TIMx register map

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

Table 42. TIMx - register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
00h	TIMx_CR1 Reset value																																	
04h	TIMx_CR2 Reset value																																	
08h	TIMx_SMCR Reset value																																	
0Ch	TIMx_DIER Reset value																																	
10h	TIMx_SR Reset value																																	
14h	TIMx_EGR Reset value																																	
18h	TIMx_CCMR1 <i>Output Compare mode</i> Reset value																																	
	TIMx_CCMR1 <i>Input Capture mode</i> Reset value																																	
1Ch	TIMx_CCMR2 <i>Output Compare mode</i> Reset value																																	
	TIMx_CCMR2 <i>Input Capture mode</i> Reset value																																	
20h	TIMx_CCER Reset value																																	
24h	TIMx_CNT Reset value																																	
28h	TIMx_PSC Reset value																																	
2Ch	TIMx_ARR Reset value																																	
30h																																		
34h	TIMx_CCR1 Reset value																																	

Table 42. TIMx - register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
38h	TIMx_CCR2																																		
	Reset value																																		
3Ch	TIMx_CCR3																																		
	Reset value																																		
40h	TIMx_CCR4																																		
	Reset value																																		
44h																																			
48h	TIMx_DCR																		DBL[4:0]		Reserved			DBA[4:0]											
	Reset value																		0 0 0 0 0				0 0 0 0 0												
4Ch	TIMx_DMAR																								DMAB[15:0]										
	Reset value																		0 0 0 0 0				0 0 0 0 0												

Refer to [Table 1 on page 27](#) for the register boundary addresses.

[Table 1 on page 27](#)

14 Controller area network (bxCAN)

14.1 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.

14.2 Main features

- Supports CAN protocol version 2.0 A, B Active
- Bit rates up to 1 Mbit/s
- Supports the Time Triggered Communication option

Transmission

- Three transmit mailboxes
- Configurable transmit priority
- Time Stamp on SOF transmission

Reception

- Two receive FIFOs with three stages
- 14 scalable filter banks/CAN cell - shared between CAN cells
- Identifier list feature
- Configurable FIFO overrun
- Time Stamp on SOF reception

Time-triggered communication option

- Disable automatic retransmission mode
- 16-bit free running timer
- Configurable timer resolution
- Time Stamp sent in last two data bytes

Management

- Maskable interrupts
- Software-efficient mailbox mapping at a unique address space

Note:

The USB and CAN share a dedicated 512-byte SRAM memory for data transmission and reception, and so they cannot be used concurrently (the shared SRAM is accessed through CAN and USB exclusively). The USB and CAN can be used in the same application but not at the same time.

14.3 General description

In today's CAN applications, the number of nodes in a network is increasing and often several networks are linked together via gateways. Typically the number of messages in the system (and thus to be handled by each node) has significantly increased. In addition to the application messages, Network Management and Diagnostic messages have been introduced.

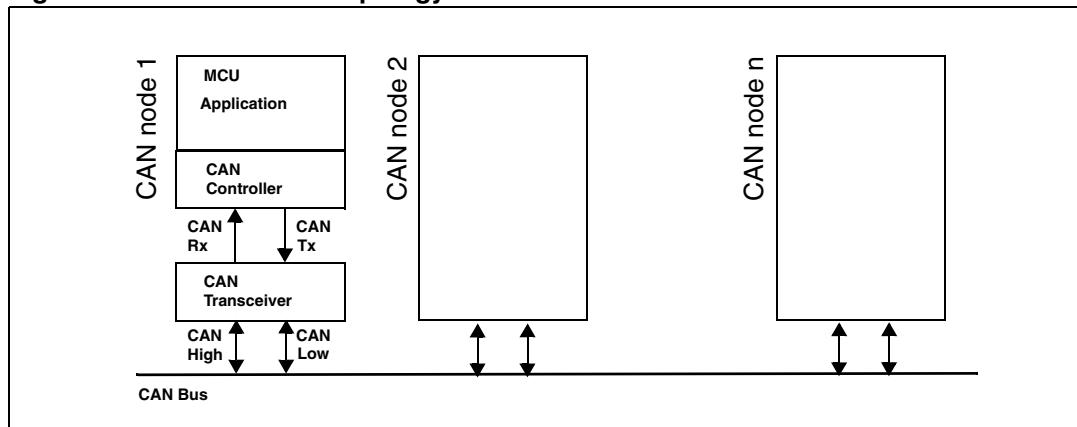
- An enhanced filtering mechanism is required to handle each type of message.

Furthermore, application tasks require more CPU time, therefore real-time constraints caused by message reception have to be reduced.

- A receive FIFO scheme allows the CPU to be dedicated to application tasks for a long time period without losing messages.

The standard HLP (Higher Layer Protocol) based on standard CAN drivers requires an efficient interface to the CAN controller.

Figure 119. CAN network topology



14.3.1 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.

14.3.2 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

14.3.3 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.

14.3.4 Acceptance filters

The bxCAN provides 14 scalable/configurable identifier filter banks for selecting the incoming messages the software needs and discarding the others.

14.3.5 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 120. CAN block diagram

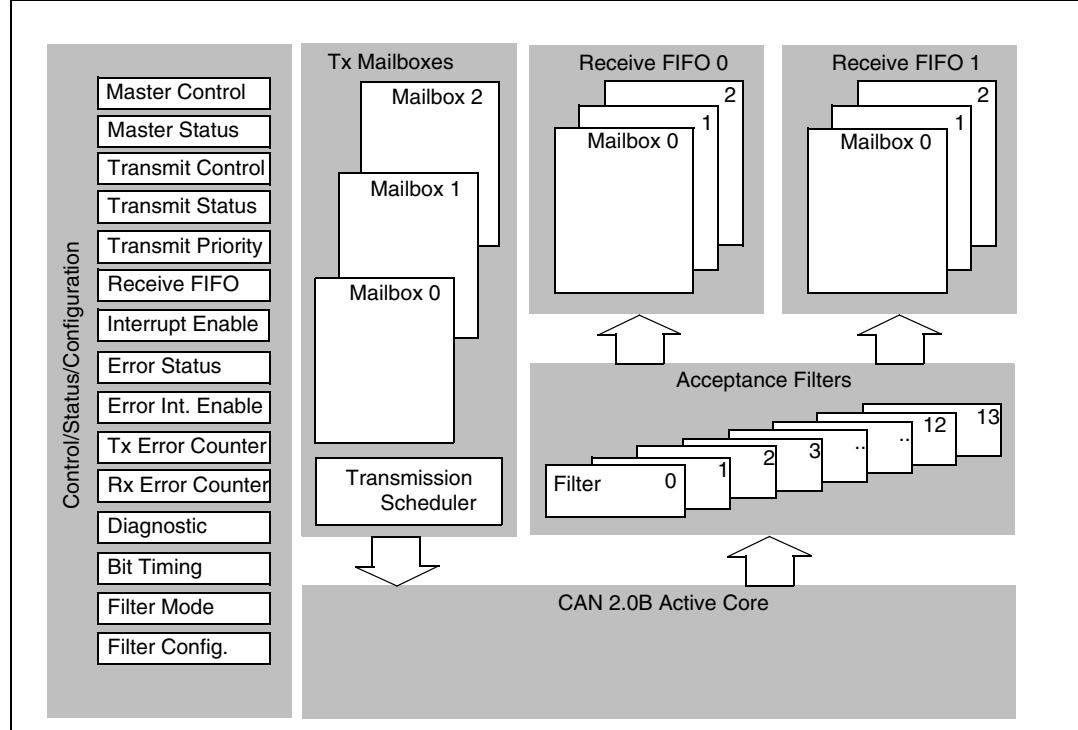
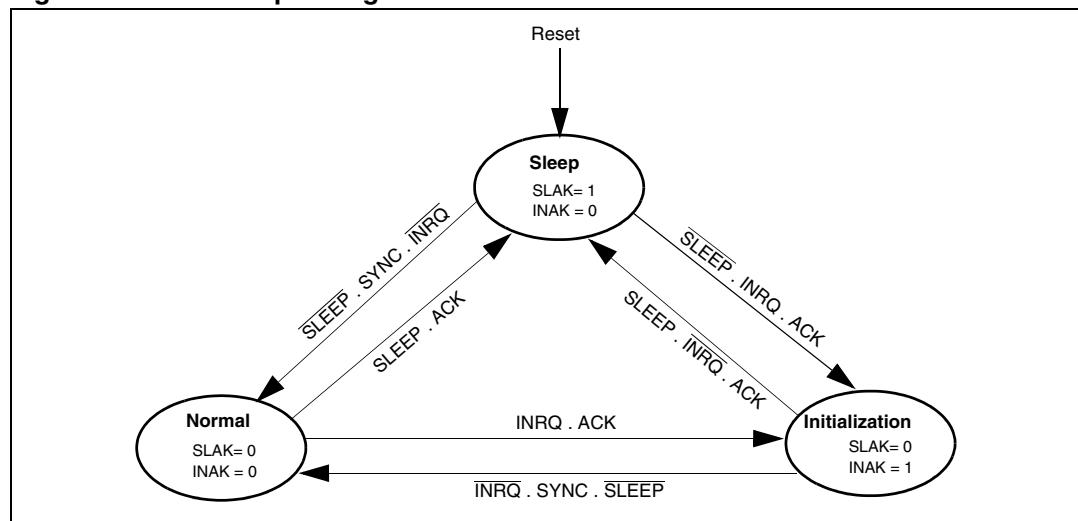


Figure 121. bxCAN operating modes



- Note:*
- 1 *ACK = The wait state during which hardware confirms a request by setting the INAK or SLAK bits in the CAN_MSR register*
 - 2 *SYNC = The state during which bxCAN waits until the CAN bus is idle, meaning 11 consecutive recessive bits have been monitored on CANRX*

14.4 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.

14.4.1 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).

14.4.2 Normal mode

Once the initialization has been done, the software must request the hardware to enter Normal mode, to synchronize on the CAN bus and start reception and transmission. Entering Normal mode is done by clearing the INRQ bit in the CAN_MCR register and waiting until the hardware has confirmed the request by clearing the INAK bit in the CAN_MSR register. Afterwards, the bxCAN synchronizes with the data transfer on the CAN bus by waiting for the occurrence of a sequence of 11 consecutive recessive bits (≡ Bus Idle) before it can take part in bus activities and start message transfer.

The initialization of the filter values is independent from Initialization Mode but must be done while the filter is not active (corresponding FACTx bit cleared). The filter scale and mode configuration must be configured before entering Normal Mode.

14.4.3 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 wakeup sequence by clearing the SLEEP bit if the AWUM bit in the CAN_MCR register is set. If the AWUM bit is cleared, software has to clear the SLEEP bit when a wakeup interrupt occurs, in order to exit from Sleep mode.

Note: *If the wakeup interrupt is enabled (WKUIE bit set in CAN_IER register) a wakeup interrupt will be generated on detection of CAN bus activity, even if the bxCAN automatically performs the wakeup sequence.*

After the SLEEP bit has been cleared, Sleep mode is exited once bxCAN has synchronized with the CAN bus, refer to [Figure 121: bxCAN operating modes](#). The Sleep mode is exited once the SLAK bit has been cleared by hardware.

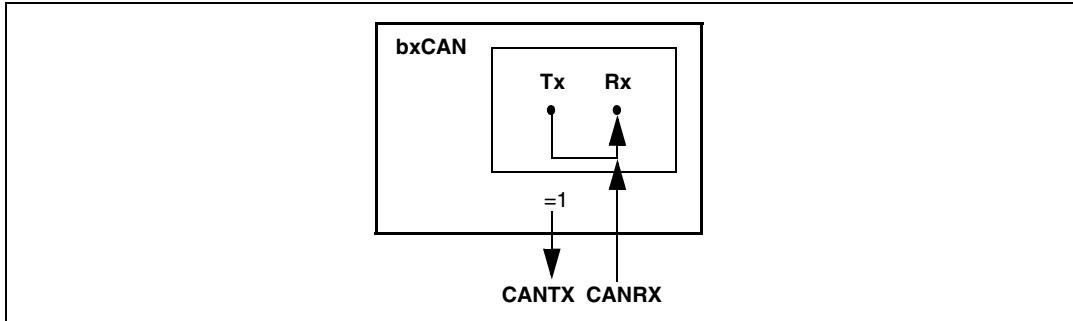
14.4.4 Test mode

Test mode can be selected by the SILM and LBKM bits in the CAN_BTR register. These bits must be configured while bxCAN is in Initialization mode. Once test mode has been selected, the INRQ bit in the CAN_MCR register must be reset to enter Normal mode.

14.4.5 Silent mode

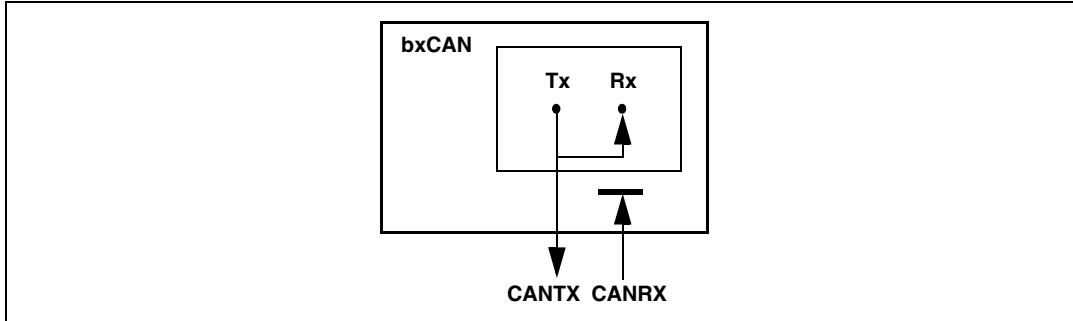
The bxCAN can be put in Silent mode by setting the SILM bit in the CAN_BTR register.

In Silent mode, the bxCAN is able to receive valid data frames and valid remote frames, but it sends only recessive bits on the CAN bus and it cannot start a transmission. If the bxCAN has to send a dominant bit (ACK bit, overload flag, active error flag), the bit is rerouted internally so that the CAN Core monitors this dominant bit, although the CAN bus may remain in recessive state. Silent mode can be used to analyze the traffic on a CAN bus without affecting it by the transmission of dominant bits (Acknowledge Bits, Error Frames).

Figure 122. bxCAN in silent mode

14.4.6 Loop back mode

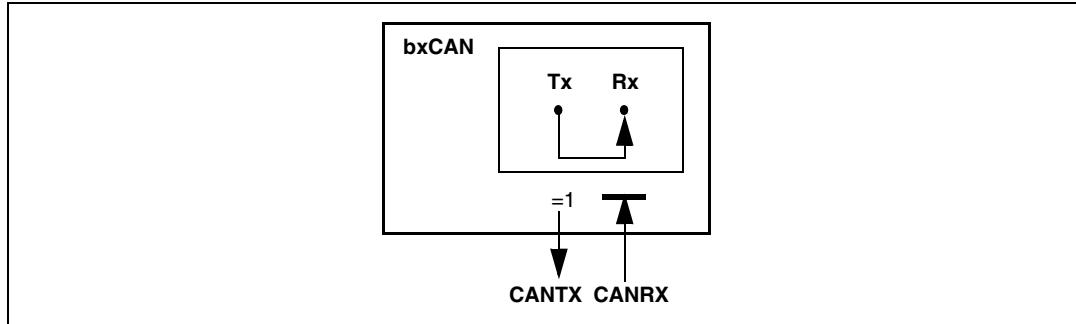
The bxCAN can be set in Loop Back Mode by setting the LBKM bit in the CAN_BTR register. In Loop Back Mode, the bxCAN treats its own transmitted messages as received messages and stores them (if they pass acceptance filtering) in a Receive mailbox.

Figure 123. bxCAN in loop back mode

This mode is provided for self-test functions. To be independent of external events, the CAN Core ignores acknowledge errors (no dominant bit sampled in the acknowledge slot of a data / remote frame) in Loop Back Mode. In this mode, the bxCAN performs an internal feedback from its Tx output to its Rx input. The actual value of the CANRX input pin is disregarded by the bxCAN. The transmitted messages can be monitored on the CANTX pin.

14.4.7 Loop back combined with silent mode

It is also possible to combine Loop Back mode and Silent mode by setting the LBKM and SLM 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 124. bxCAN in combined mode

14.5 Functional description

14.5.1 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 will be **Scheduled** for transmission. The transmission of the message of the scheduled mailbox will start (enter **transmit** state) when the CAN bus becomes idle. Once the mailbox has been successfully transmitted, it will become **empty** again. The hardware indicates a successful transmission by setting the RQCP and TXOK bits in the CAN_TSR register.

If the transmission fails, the cause is indicated by the ALST bit in the CAN_TSR register in case of an Arbitration Lost, and/or the TERR bit, in case of transmission error detection.

Transmit priority

By identifier:

When more than one transmit mailbox is pending, the transmission order is given by the identifier of the message stored in the mailbox. The message with the lowest identifier value has the highest priority according to the arbitration of the CAN protocol. If the identifier values are equal, the lower mailbox number will be scheduled first.

By transmit request order:

The transmit mailboxes can be configured as a transmit FIFO by setting the TXFP bit in the CAN_MCR register. In this mode the priority order is given by the transmit request order.

This mode is very useful for segmented transmission.

Abort

A transmission request can be aborted by the user setting the ABRQ bit in the CAN_TSR register. In **Pending** or **Scheduled** state, the mailbox is aborted immediately. An abort request while the mailbox is in **transmit** state can have two results. If the mailbox is transmitted successfully the mailbox becomes **empty** with the TXOK bit set in the CAN_TSR register. If the transmission fails, the mailbox becomes **Scheduled**, the

transmission is aborted and becomes **empty** with TXOK cleared. In all cases the mailbox will become **empty** again at least at the end of the current transmission.

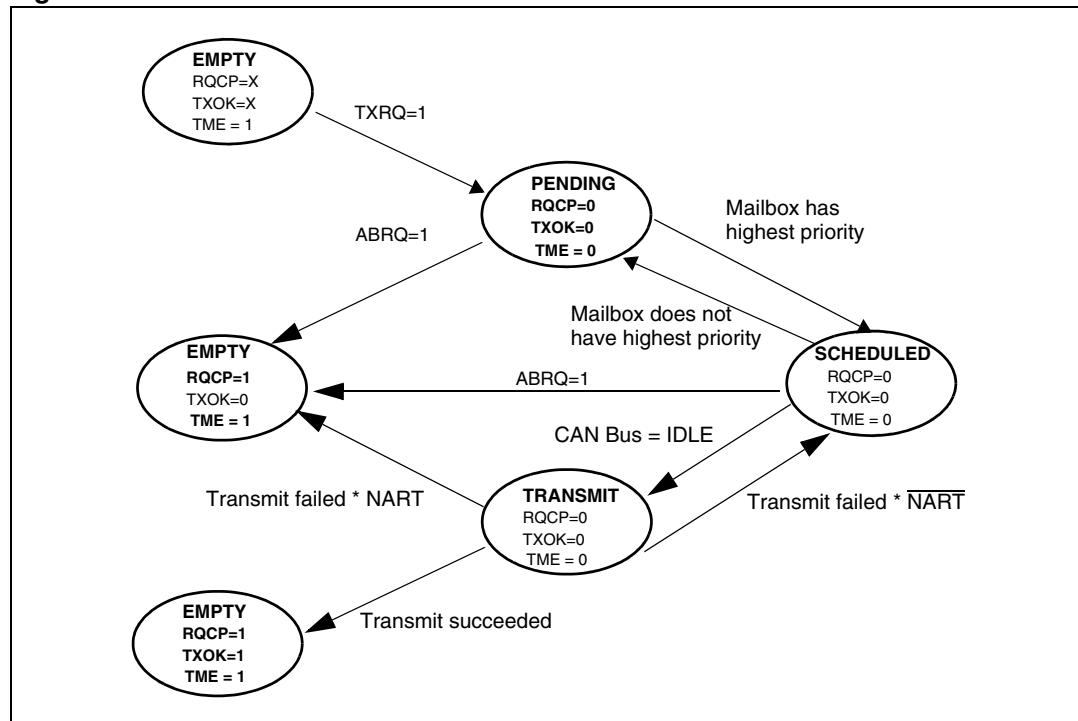
Non-automatic retransmission mode

This mode has been implemented in order to fulfil the requirement of the Time Triggered Communication option of the CAN standard. To configure the hardware in this mode the NART bit in the CAN_MCR register must be set.

In this mode, each transmission is started only once. If the first attempt fails, due to an arbitration loss or an error, the hardware will not automatically restart the message transmission.

At the end of the first transmission attempt, the hardware considers the request as completed and sets the RQCP bit in the CAN_TSR register. The result of the transmission is indicated in the CAN_TSR register by the TXOK, ALST and TERR bits.

Figure 125. Transmit mailbox states



14.5.2 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 captured on the sample point of the Start Of Frame bit in both reception and transmission.

14.5.3 Reception handling

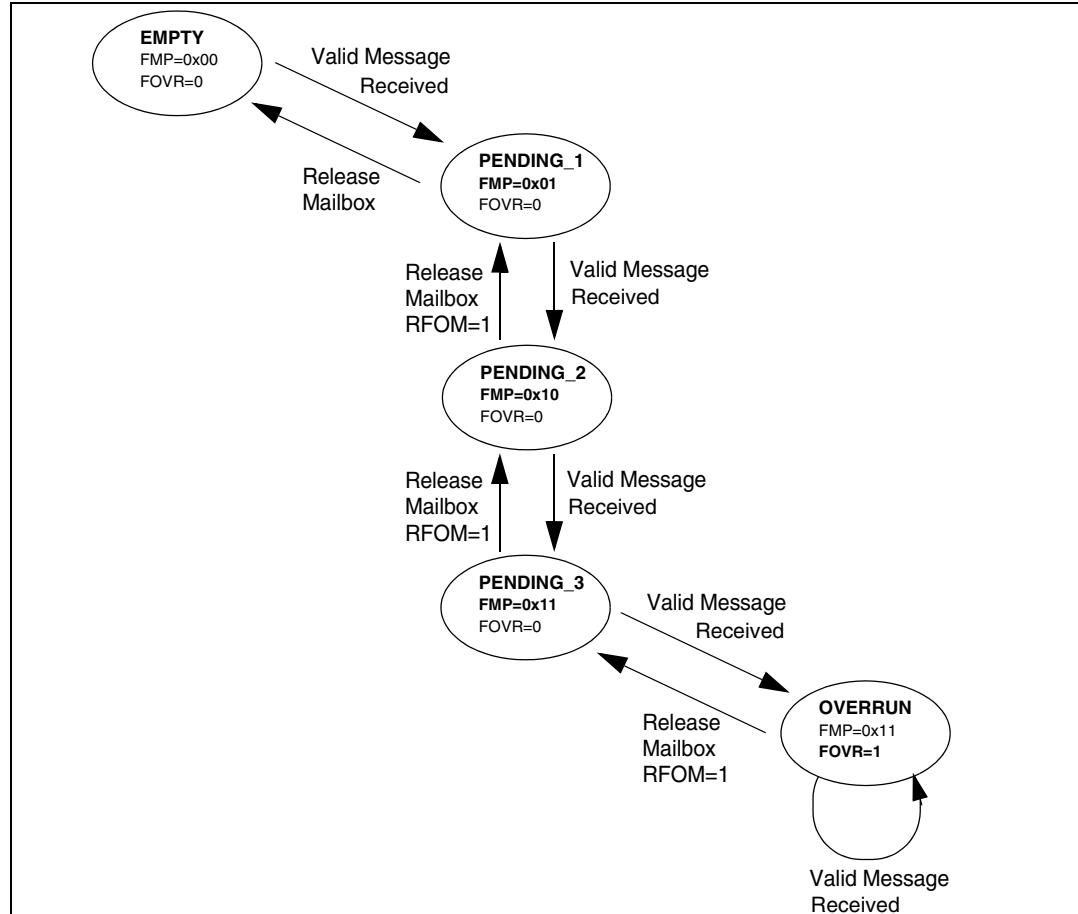
For the reception of CAN messages, three mailboxes organized as a FIFO are provided. In order to save CPU load, simplify the software and guarantee data consistency, the FIFO is

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 14.5.4: Identifier filtering](#).

Figure 126. 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 CRFR 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 CRFR register. The FIFO becomes **empty** again. If a new valid message has been received in the meantime, the FIFO stays in **pending_1** state and the new message is available in the output mailbox.

If the application does not release the mailbox, the next valid message will be stored in the FIFO which enters **pending_2** state (FMP[1:0] = 10b). The storage process is repeated for the next valid message putting the FIFO into **pending_3** state (FMP[1:0] = 11b). At this point, the software must release the output mailbox by setting the RFOM bit, so that a

mailbox is free to store the next valid message. Otherwise the next valid message received will cause a loss of message.

Refer also to [Section 14.5.5: Message storage](#)

Overrun

Once the FIFO is in **pending_3** state (i.e. the three mailboxes are full) the next valid message reception will lead to an **overrun** and a message will be lost. The hardware signals the overrun condition by setting the FOVR bit in the CRFR register. Which message is lost depends on the configuration of the FIFO:

- If the FIFO lock function is disabled (RFLM bit in the CAN_MCR register cleared) the last message stored in the FIFO will be overwritten by the new incoming message. In this case the latest messages will be always available to the application.
- If the FIFO lock function is enabled (RFLM bit in the CAN_MCR register set) the most recent message will be discarded and the software will have the three oldest messages in the FIFO available.

Reception related interrupts

Once a message has been stored in the FIFO, the FMP[1:0] bits are updated and an interrupt request is generated if the FMPIE bit in the CAN_IER register is set.

When the FIFO becomes full (i.e. a third message is stored) the FULL bit in the CRFR 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.

14.5.4 Identifier filtering

In the CAN protocol the identifier of a message is not associated with the address of a node but related to the content of the message. Consequently a transmitter broadcasts its message to all receivers. On message reception a receiver node decides - depending on the identifier value - whether the software needs the message or not. If the message is needed, it is copied into the SRAM. If not, the message must be discarded without intervention by the software.

To fulfil this requirement, the bxCAN Controller provides 14 configurable and scalable filter banks (13-0) to the application, in order to receive only the messages the software needs. This hardware filtering saves CPU resources which would be otherwise needed to perform filtering by software. Each filter bank x consists of two 32-bit registers, CAN_FxR0 and CAN_FxR1.

Scalable width

To optimize and adapt the filters to the application needs, each filter bank can be scaled independently. Depending on the filter scale a filter bank provides:

- One 32-bit filter for the STDID[10:0], EXTID[17:0], IDE and RTR bits.
- Two 16-bit filters for the STDID[10:0], RTR, IDE and EXTID[17:15] bits.

Refer to [Figure 127](#).

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 CFMR 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 CFSCR register, refer to [Figure 127](#). The **identifier list** or **identifier mask** mode for the corresponding Mask/Identifier registers is configured by means of the FBMx bits in the CFMR register.

To filter a group of identifiers, configure the Mask/Identifier registers in mask mode.

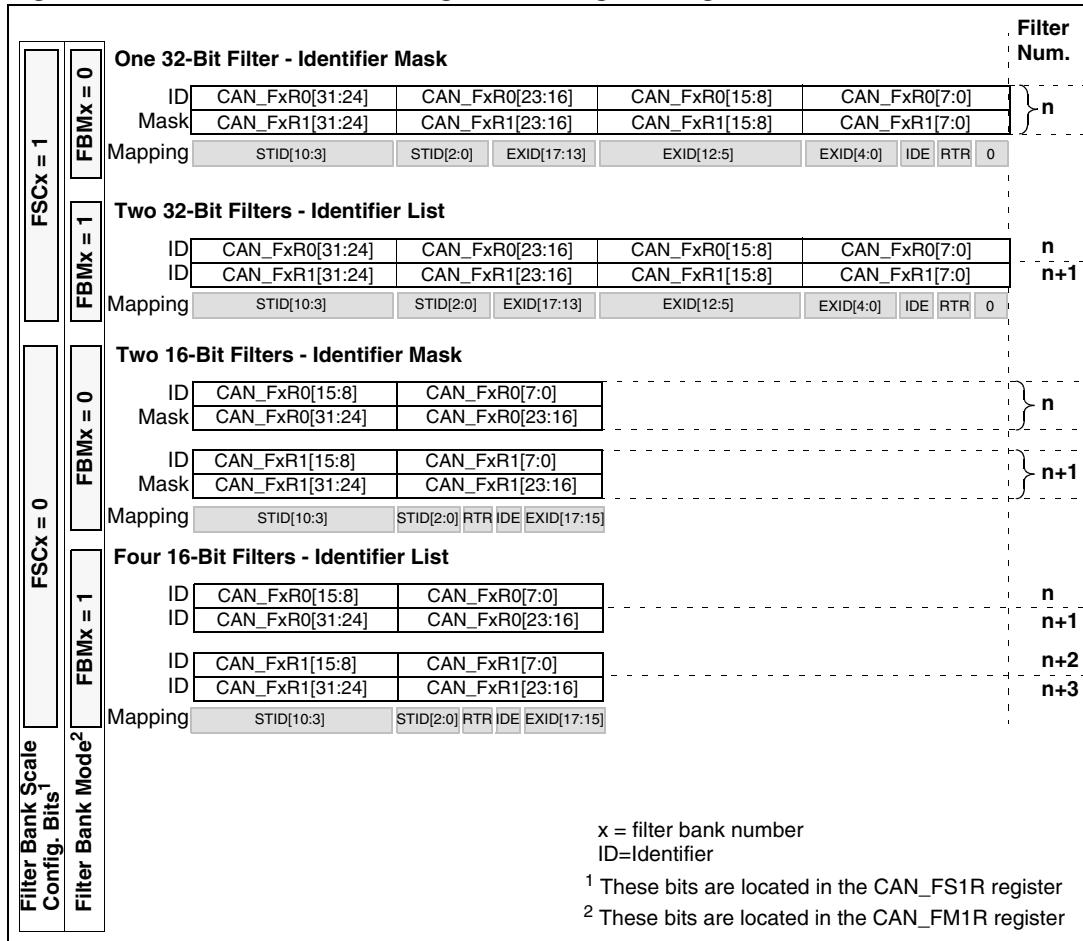
To select single identifiers, configure the Mask/Identifier registers in identifier list mode.

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 14 filter banks.

Concerning the filter configuration, refer to [Figure 127](#).

Figure 127. 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 128](#) for an example.

Figure 128. Example of filter numbering

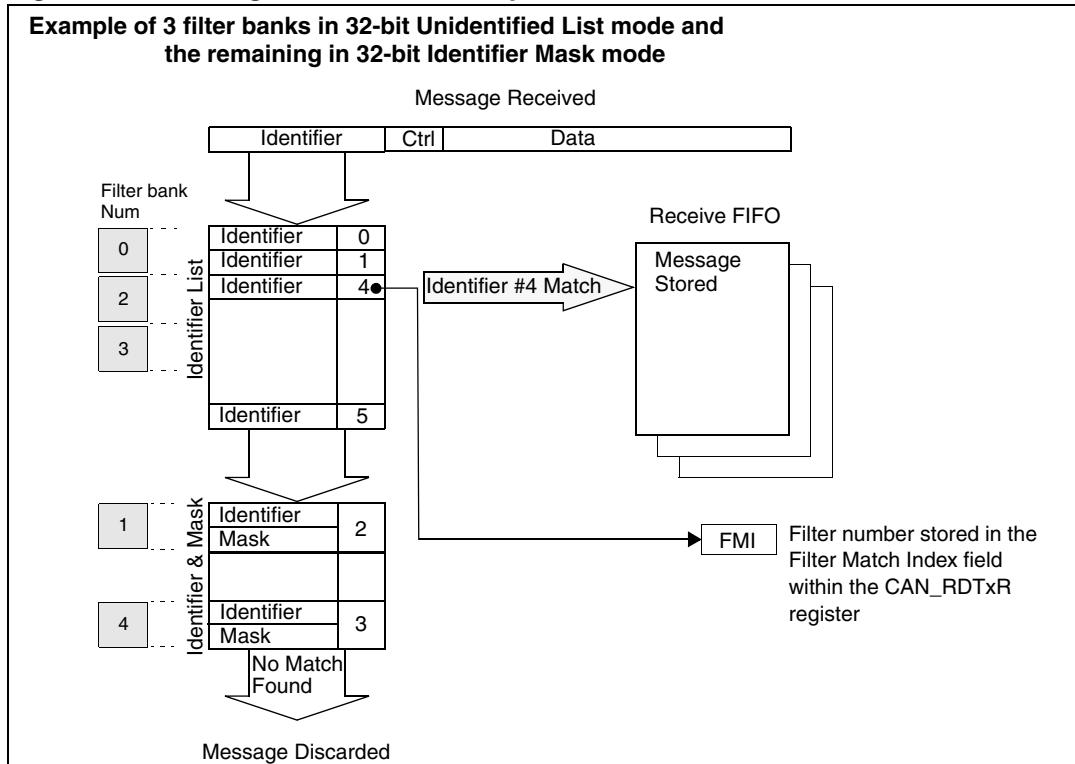
Filter Bank	FIFO0	Filter Num.	Filter Bank	FIFO1	Filter Num.
0	ID List (32-bit)	0 1	2	ID Mask (16-bit)	0 1
1	ID Mask (32-bit)	2	4	ID List (32-bit)	2 3
3	ID List (16-bit)	3 4 5 6	7	Deactivated ID Mask (16-bit)	4 5
5	Deactivated ID List (32-bit)	7 8	8	ID Mask (16-bit)	6 7
6	ID Mask (16-bit)	9 10	10	Deactivated 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

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 129. 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.

14.5.5 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 43. Transmit mailbox mapping

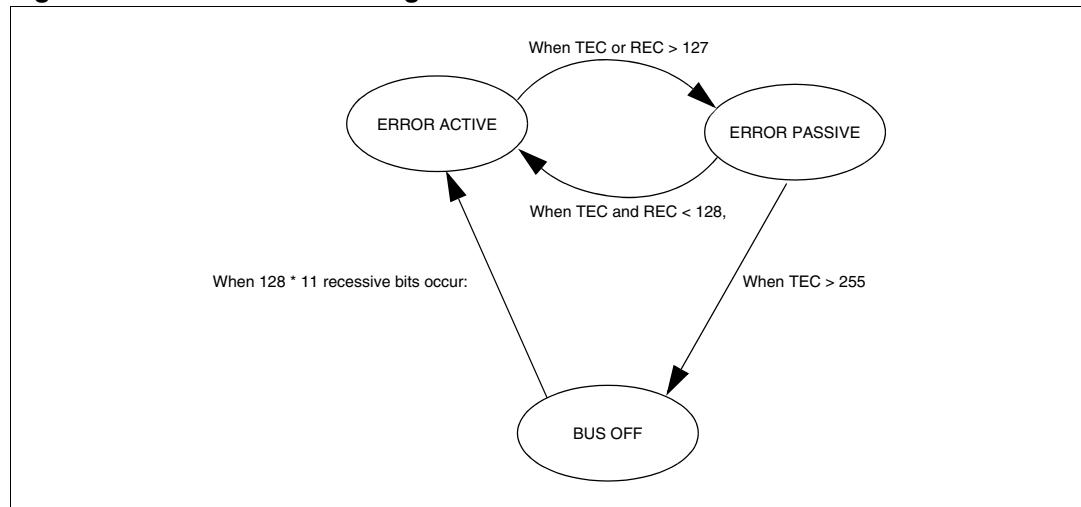
Offset to transmit mailbox base address	Register name
0	CAN_TIxR
4	CAN_TDTxR
8	CAN_TDLxR
12	CAN_TDhxR

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 CRFR 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 44. 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 130. CAN error state diagram

14.5.6 Error management

The error management as described in the CAN protocol is handled entirely by hardware using a Transmit Error Counter (TEC value, in CAN_ESR register) and a Receive Error Counter (REC value, in the CAN_ESR register), which get incremented or decremented according to the error condition. For detailed information about TEC and REC management, please refer to the CAN standard.

Both of them may be read by software to determine the stability of the network. Furthermore, the CAN hardware provides detailed information on the current error status in CAN_ESR register. By means of the CAN_IER register (ERRIE bit, etc.), the software can configure the interrupt generation on error detection in a very flexible way.

Bus-Off recovery

The Bus-Off state is reached when TEC is greater than 255, this state is indicated by BOFF bit in CAN_ESR register. In Bus-Off state, the bxCAN is no longer able to transmit and receive messages.

Depending on the ABOM bit in the CAN_MCR register bxCAN will recover from Bus-Off (become error active again) either automatically or on software request. But in both cases the bxCAN has to wait at least for the recovery sequence specified in the CAN standard (128 occurrences of 11 consecutive recessive bits monitored on CANRX).

If ABOM is set, the bxCAN will start the recovering sequence automatically after it has entered Bus-Off state.

If ABOM is cleared, the software must initiate the recovering sequence by requesting bxCAN to enter and to leave initialization mode.

Note:

In initialization mode, bxCAN does not monitor the CANRX signal, therefore it cannot complete the recovery sequence. To recover, bxCAN must be in normal mode.

14.5.7 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 ($1 \times t_{CAN}$).
- **Bit segment 1 (BS1):** defines the location of the sample point. It includes the PROP_SEG and PHASE_SEG1 of the CAN standard. Its duration is programmable between 1 and 16 time quanta but may be automatically lengthened to compensate for positive phase drifts due to differences in the frequency of the various nodes of the network.
- **Bit segment 2 (BS2):** defines the location of the transmit point. It represents the PHASE_SEG2 of the CAN standard. Its duration is programmable between 1 and 8 time quanta but may also be automatically shortened to compensate for negative phase drifts.

The resynchronization Jump Width (SJW) defines an upper bound to the amount of lengthening or shortening of the bit segments. It is programmable between 1 and 4 time quanta.

A valid edge is defined as the first transition in a bit time from dominant to recessive bus level provided the controller itself does not send a recessive bit.

If a valid edge is detected in BS1 instead of SYNC_SEG, BS1 is extended by up to SJW so that the sample point is delayed.

Conversely, if a valid edge is detected in BS2 instead of SYNC_SEG, BS2 is shortened by up to SJW so that the transmit point is moved earlier.

As a safeguard against programming errors, the configuration of the Bit Timing Register (CAN_BTR) is only possible while the device is in Standby mode.

Note: For a detailed description of the CAN bit timing and resynchronization mechanism, please refer to the ISO 11898 standard.

Figure 131. Bit timing

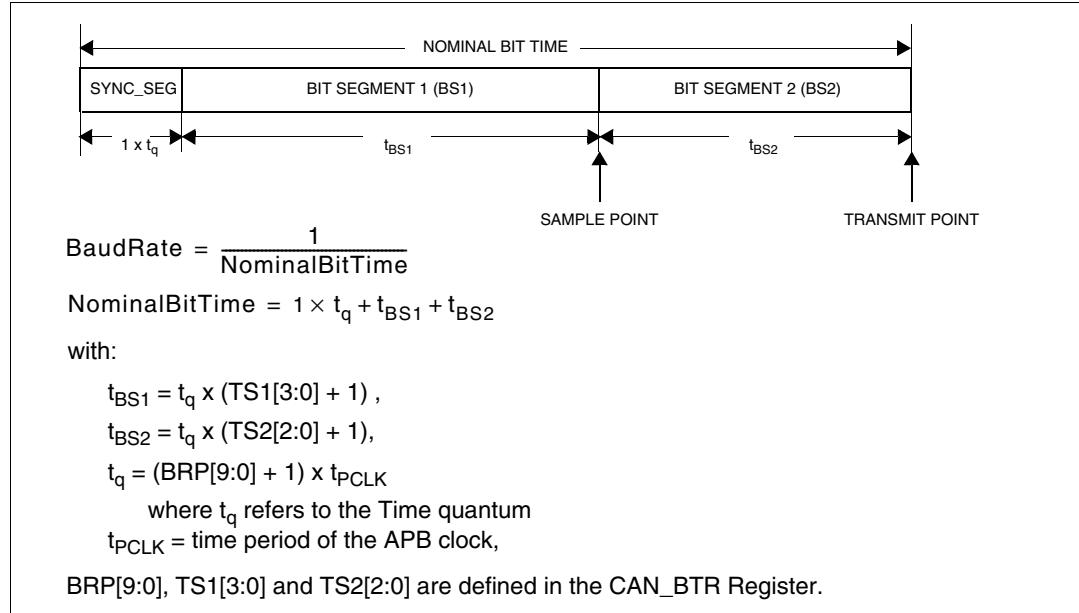
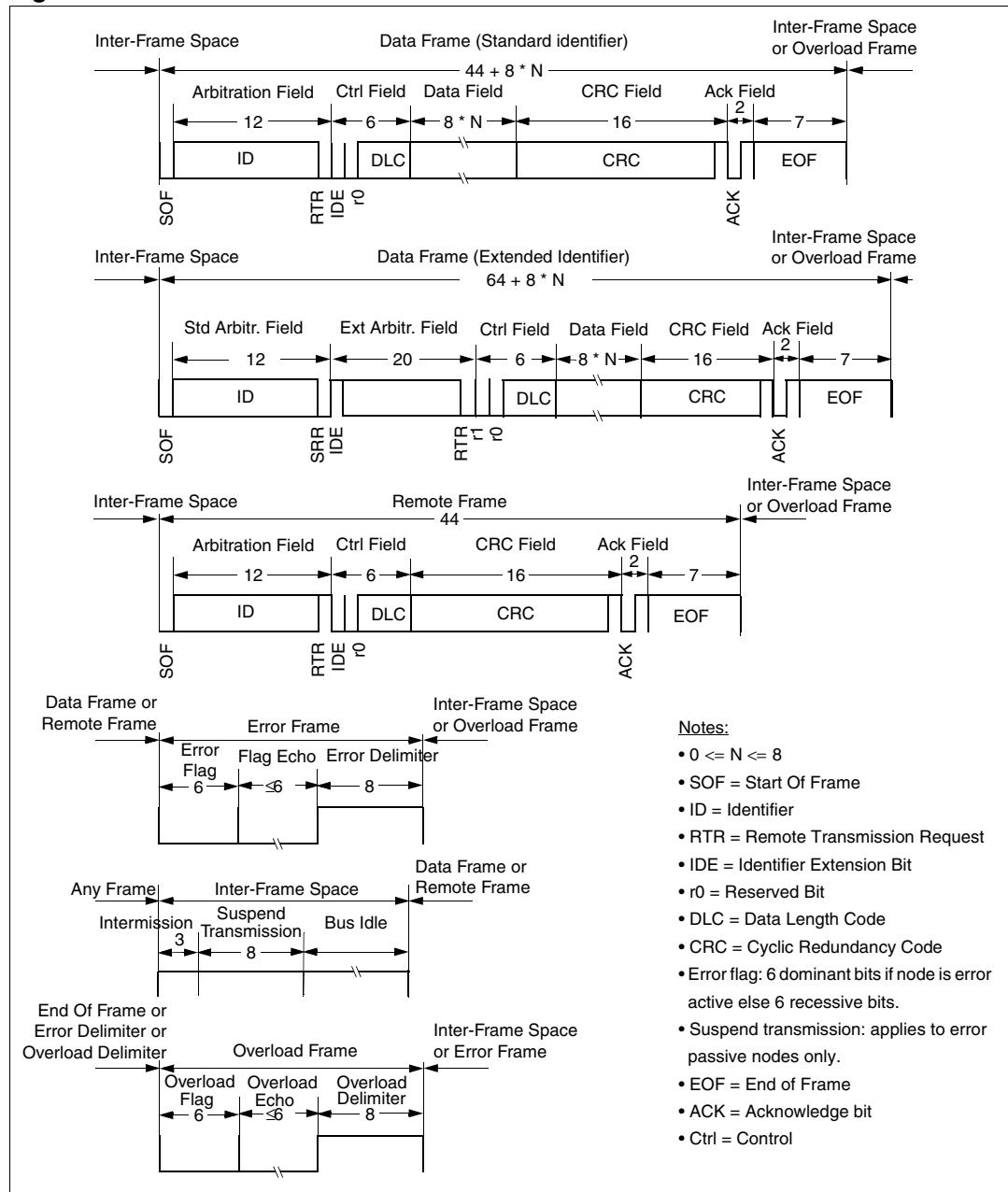
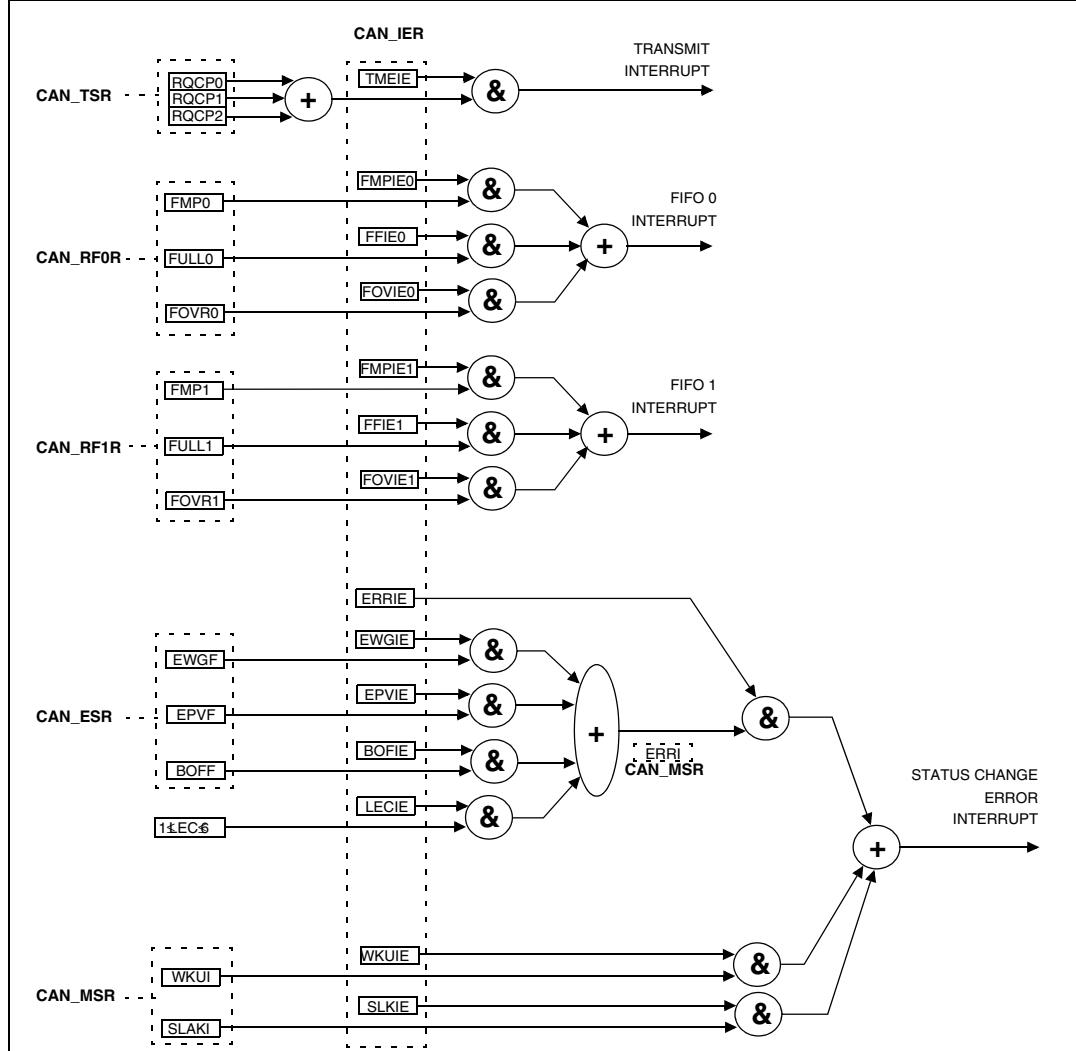


Figure 132. CAN frames

14.6 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 133. Event flags and interrupt generation



- The **transmit interrupt** can be generated by the following events:
 - Transmit mailbox 0 becomes empty, RQCP0 bit in the CAN_TSR register set.
 - Transmit mailbox 1 becomes empty, RQCP1 bit in the CAN_TSR register set.
 - Transmit mailbox 2 becomes empty, RQCP2 bit in the CAN_TSR register set.
- The **FIFO 0 interrupt** can be generated by the following events:
 - Reception of a new message, FMP0 bits in the CAN_RF0R register are not '00'.
 - FIFO0 full condition, FULL0 bit in the CAN_RF0R register set.
 - FIFO0 overrun condition, FOVR0 bit in the CAN_RF0R register set.
- The **FIFO 1 interrupt** can be generated by the following events:
 - Reception of a new message, FMP1 bits in the CAN_RF1R register are not '00'.
 - FIFO1 full condition, FULL1 bit in the CAN_RF1R register set.
 - FIFO1 overrun condition, FOVR1 bit in the CAN_RF1R register set.

- The **error and status change interrupt** can be generated by the following events:
 - Error condition, for more details on error conditions please refer to the CAN Error Status register (CAN_ESR).
 - Wakeup condition, SOF monitored on the CAN Rx signal.
 - Entry into Sleep mode.

14.7 Register access protection

Erroneous access to certain configuration registers can cause the hardware to temporarily disturb the whole CAN network. Therefore the CAN_BTR register can be modified by software only while the CAN hardware is in initialization mode.

Although the transmission of incorrect data will not cause problems at the CAN network level, it can severely disturb the application. A transmit mailbox can be only modified by software while it is in empty state, refer to [Figure 125: Transmit mailbox states](#).

The filters 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.

14.8 CAN register description

Refer to [Section 1.1 on page 23](#) for a list of abbreviations used in register descriptions.

14.8.1 Control and status registers

CAN master control register (CAN_MCR)

Address offset: 0x00

Reset value: 0x0001 0002

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RE SET		Reserved				TTCM	ABOM	AWUM	NART	RFLM	TXFP	SLEEP	INRQ		
rs						rw	rw	rw	rw						

Bits 31:16 Reserved, forced by hardware to 0.

Bit 15 **RESET:** *bxCAN software master reset*

0: Normal operation.

1: Force a master reset of the bxCAN -> Sleep mode activated after reset (FMP bits and CAN_MCR register are initialized to the reset values). This bit is automatically reset to 0.

Bits 14:8 Reserved, forced by hardware to 0.

Bit 7 **TTCM:** *Time Triggered Communication Mode*

0: Time Triggered Communication mode disabled.

1: Time Triggered Communication mode enabled

Note: For more information on Time Triggered Communication mode, please refer to [Section 14.5.2: Time triggered communication mode](#).

Bit 6 **ABOM:** *Automatic Bus-Off Management*

This bit controls the behavior of the CAN hardware on leaving the Bus-Off state.

0: The Bus-Off state is left on software request, once 128 occurrences of 11 recessive bits have been monitored and the software has first set and cleared the INRQ bit of the CAN_MCR register.

1: The Bus-Off state is left automatically by hardware once 128 occurrences of 11 recessive bits have been monitored.

For detailed information on the Bus-Off state please refer to [Section 14.5.6: Error management](#).

Bit 5 **AWUM:** *Automatic Wakeup Mode*

This bit controls the behavior of the CAN hardware on message reception during Sleep mode.

0: The Sleep mode is left on software request by clearing the SLEEP bit of the CAN_MCR register.

1: The Sleep mode is left automatically by hardware on CAN message detection.

The SLEEP bit of the CAN_MCR register and the SLAK bit of the CAN_MSR register are cleared by hardware.

Bit 4 **NART:** *No Automatic Retransmission*

0: The CAN hardware will automatically retransmit the message until it has been successfully transmitted according to the CAN standard.

1: A message will be transmitted only once, independently of the transmission result (successful, error or arbitration lost).

Bit 3 **RFLM:** *Receive FIFO Locked Mode*

0: Receive FIFO not locked on overrun. Once a receive FIFO is full the next incoming message will overwrite the previous one.

1: Receive FIFO locked against overrun. Once a receive FIFO is full the next incoming message will be discarded.

Bit 2 **TXFP:** *Transmit FIFO Priority*

This bit controls the transmission order when several mailboxes are pending at the same time.

0: Priority driven by the identifier of the message

1: Priority driven by the request order (chronologically)

Bit 1 **SLEEP:** *Sleep Mode Request*

This bit is set by software to request the CAN hardware to enter the Sleep mode. Sleep mode will be entered as soon as the current CAN activity (transmission or reception of a CAN frame) has been completed.

This bit is cleared by software to exit Sleep mode.

This bit is cleared by hardware when the AWUM bit is set and a SOF bit is detected on the CAN Rx signal.

This bit is set after reset - CAN starts in Sleep mode.

Bit 0 INRQ: Initialization Request

The software clears this bit to switch the hardware into normal mode. Once 11 consecutive recessive bits have been monitored on the Rx signal the CAN hardware is synchronized and ready for transmission and reception. Hardware signals this event by clearing the INAK bit in the CAN_MSR register.

Software sets this bit to request the CAN hardware to enter initialization mode. Once software has set the INRQ bit, the CAN hardware waits until the current CAN activity (transmission or reception) is completed before entering the initialization mode. Hardware signals this event by setting the INAK bit in the CAN_MSR register.

CAN master status register (CAN_MSR)

Address offset: 0x04

Reset value: 0x0000 0C02

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		RX	SAMP	RXM	TXM	Reserved		SLAKI	WKUI	ERRI	SLAK	INAK			rw
		r	r	r	r			rc_w1	rc_w1	rc_w1	r	r			

Bits 31:12 Reserved, forced by hardware to 0.

Bit 11 RX: CAN Rx Signal

Monitors the actual value of the CAN_RX Pin.

Bit 10 SAMP: Last Sample Point

The value of RX on the last sample point (current received bit value).

Bit 9 RXM: Receive Mode

The CAN hardware is currently receiver.

Bit 8 TXM: Transmit Mode

The CAN hardware is currently transmitter.

Bits 7:5 Reserved, forced by hardware to 0.

Bit 4 SLAKI: Sleep Acknowledge Interrupt

When SLKIE=1, this bit is set by hardware to signal that the bxCAN has entered Sleep Mode. When set, this bit generates a status change interrupt if the SLKIE bit in the CAN_IER register is set. This bit is cleared by software or by hardware, when SLAK is cleared.

Note: When SLKIE=0, no polling on SLAKI is possible. In this case the SLAK bit can be polled.

Bit 3 WKUI: Wakeup Interrupt

This bit is set by hardware to signal that a SOF bit has been detected while the CAN hardware was in Sleep mode. Setting this bit generates a status change interrupt if the WKUIE bit in the CAN_IER register is set.

This bit is cleared by software.

Bit 2 ERRI: Error Interrupt

This bit is set by hardware when a bit of the CAN_ESR has been set on error detection and the corresponding interrupt in the CAN_IER is enabled. Setting this bit generates a status change interrupt if the ERRIE bit in the CAN_IER register is set.

This bit is cleared by software.

Bit 1 SLAK: Sleep Acknowledge

This bit is set by hardware and indicates to the software that the CAN hardware is now in Sleep mode. This bit acknowledges the Sleep mode request from the software (set SLEEP bit in CAN_MCR register).

This bit is cleared by hardware when the CAN hardware has left Sleep mode (to be synchronized on the CAN bus). To be synchronized the hardware has to monitor a sequence of 11 consecutive recessive bits on the CAN RX signal.

Note: The process of leaving Sleep mode is triggered when the SLEEP bit in the CAN_MCR register is cleared. Please refer to the AWUM bit of the CAN_MCR register description for detailed information for clearing SLEEP bit

Bit 0 INAK: Initialization Acknowledge

This bit is set by hardware and indicates to the software that the CAN hardware is now in initialization mode. This bit acknowledges the initialization request from the software (set INRQ bit in CAN_MCR register).

This bit is cleared by hardware when the CAN hardware has left the initialization mode (to be synchronized on the CAN bus). To be synchronized the hardware has to monitor a sequence of 11 consecutive recessive bits on the CAN RX signal.

CAN transmit status register (CAN_TSR)

Address offset: 0x08

Reset value: 0x1C00 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LOW2	LOW1	LOW0	TME2	TME1	TME0	CODE[1:0]	ABRQ 2		Reserved		TERR 2	ALST2	TXOK 2	RQCP 2	
r	r	r	r	r	r	r	r	rs			rc_w1	rc_w1	rc_w1	rc_w1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABRQ 1	Reserved		TERR 1	ALST1	TXOK 1	RQCP 1	ABRQ 0		Reserved		TERR 0	ALST0	TXOK 0	RQCP 0	
rs			rc_w1	rc_w1	rc_w1	rc_w1	rs				rc_w1	rc_w1	rc_w1	rc_w1	

Bit 31 LOW2: Lowest Priority Flag for Mailbox 2

This bit is set by hardware when more than one mailbox are pending for transmission and mailbox 2 has the lowest priority.

Bit 30 LOW1: Lowest Priority Flag for Mailbox 1

This bit is set by hardware when more than one mailbox are pending for transmission and mailbox 1 has the lowest priority.

Bit 29 LOW0: Lowest Priority Flag for Mailbox 0

This bit is set by hardware when more than one mailbox are pending for transmission and mailbox 0 has the lowest priority.

Note: The LOW[2:0] bits are set to zero when only one mailbox is pending.

Bit 28 TME2: Transmit Mailbox 2 Empty

This bit is set by hardware when no transmit request is pending for mailbox 2.

Bit 27 TME1: Transmit Mailbox 1 Empty

This bit is set by hardware when no transmit request is pending for mailbox 1.

Bit 26 TME0: Transmit Mailbox 0 Empty

This bit is set by hardware when no transmit request is pending for mailbox 0.

Bits 25:24 **CODE[1:0]: Mailbox Code**

In case at least one transmit mailbox is free, the code value is equal to the number of the next transmit mailbox free.

In case all transmit mailboxes are pending, the code value is equal to the number of the transmit mailbox with the lowest priority.

Bit 23 **ABRQ2: Abort Request for Mailbox 2**

Set by software to abort the transmission request for the corresponding mailbox.

Cleared by hardware when the mailbox becomes empty.

Setting this bit has no effect when the mailbox is not pending for transmission.

Bits 22:20 Reserved, forced by hardware to 0.

Bit 19 **TERR2: Transmission Error of Mailbox 2**

This bit is set when the previous TX failed due to an error.

Bit 18 **ALST2: Arbitration Lost for Mailbox 2**

This bit is set when the previous TX failed due to an arbitration lost.

Bit 17 **TXOK2: Transmission OK of Mailbox 2**

The hardware updates this bit after each transmission attempt.

0: The previous transmission failed

1: The previous transmission was successful

This bit is set by hardware when the transmission request on mailbox 2 has been completed successfully. Please refer to [Figure 125](#).

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, forced by hardware to 0.

Bit 11 **TERR1: Transmission Error of Mailbox1**

This bit is set when the previous TX failed due to an error.

Bit 10 **ALST1: Arbitration Lost for Mailbox1**

This bit is set when the previous TX failed due to an arbitration lost.

Bit 9 **TXOK1: Transmission OK of Mailbox1**

The hardware updates this bit after each transmission attempt.

0: The previous transmission failed

1: The previous transmission was successful

This bit is set by hardware when the transmission request on mailbox 1 has been completed successfully. Please refer to [Figure 125](#)

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_TMR1R register).

Clearing this bit clears all the status bits (TXOK1, ALST1 and TERR1) for Mailbox 1.

Bit 7 ABRQ0: Abort Request for Mailbox0

Set by software to abort the transmission request for the corresponding mailbox.

Cleared by hardware when the mailbox becomes empty.

Setting this bit has no effect when the mailbox is not pending for transmission.

Bits 6:4 Reserved, forced by hardware to 0.

Bit 3 TERR0: Transmission Error of Mailbox0

This bit is set when the previous TX failed due to an error.

Bit 2 ALST0: Arbitration Lost for Mailbox0

This bit is set when the previous TX failed due to an arbitration lost.

Bit 1 TXOK0: Transmission OK of Mailbox0

The hardware updates this bit after each transmission attempt.

0: The previous transmission failed

1: The previous transmission was successful

This bit is set by hardware when the transmission request on mailbox 1 has been completed successfully. Please refer to [Figure 125](#)

Bit 0 RQCP0: Request Completed Mailbox0

Set by hardware when the last request (transmit or abort) has been performed.

Cleared by software writing a “1” or by hardware on transmission request (TXRQ0 set in CAN_TI0R register).

Clearing this bit clears all the status bits (TXOK0, ALST0 and TERR0) for Mailbox 0.

CAN receive FIFO 0 register (CAN_RF0R)

Address offset: 0x0C

Reset value: 0x00

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reserved																
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reserved								RFOM0	FOVR0	FULL0	Res.	FMP0[1:0]			r	r

Bit 31:6 Reserved, forced by hardware to 0.

Bit 5 RFOM0: Release FIFO 0 Output Mailbox

Set by software to release the output mailbox of the FIFO. The output mailbox can only be released when at least one message is pending in the FIFO. Setting this bit when the FIFO is empty has no effect. If at least two messages are pending in the FIFO, the software has to release the output mailbox to access the next message.

Cleared by hardware when the output mailbox has been released.

Bit 4 FOVR0: FIFO 0 Overrun

This bit is set by hardware when a new message has been received and passed the filter while the FIFO was full.

This bit is cleared by software.

Bit 3 FULL0: FIFO 0 Full

Set by hardware when three messages are stored in the FIFO.

This bit is cleared by software.

Bit 2 Reserved, forced by hardware to 0.

Bits 1:0 FMP0[1:0]: FIFO 0 Message Pending

These bits indicate how many messages are pending in the receive FIFO.

FMP is increased each time the hardware stores a new message in to the FIFO. FMP is decreased each time the software releases the output mailbox by setting the RFOM0 bit.

CAN receive FIFO 1 register (CAN_RF1R)

Address offset: 0x10

Reset value: 0x00

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										RFOM1	FOVR1	FULL1	Res.	FMP1[1:0]	
										rs	rc_w1	rc_w1		r	r

Bits 31:6 Reserved, forced by hardware to 0.

Bit 5 RFOM1: Release FIFO 1 Output Mailbox

Set by software to release the output mailbox of the FIFO. The output mailbox can only be released when at least one message is pending in the FIFO. Setting this bit when the FIFO is empty has no effect. If at least two messages are pending in the FIFO, the software has to release the output mailbox to access the next message.

Cleared by hardware when the output mailbox has been released.

Bit 4 FOVR1: FIFO 1 Overrun

This bit is set by hardware when a new message has been received and passed the filter while the FIFO was full.

This bit is cleared by software.

Bit 3 FULL1: FIFO 1 Full

Set by hardware when three messages are stored in the FIFO.

This bit is cleared by software.

Bit 2 Reserved, forced by hardware to 0.

Bits 1:0 FMP1[1:0]: FIFO 1 Message Pending

These bits indicate how many messages are pending in the receive FIFO1.

FMP1 is increased each time the hardware stores a new message in to the FIFO1. FMP is decreased each time the software releases the output mailbox by setting the RFOM1 bit.

CAN interrupt enable register (CAN_IER)

Address offset: 0x14

Reset value: 0x00

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved														SLKIE	WKUIE
rw														rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERRIE	Reserved		LEC IE	BOF IE	EPV IE	EWG IE	Res.	FOV IE1	FF IE1	FMP IE1	FOV IE0	FF IE0	FMP IE0	TME IE	
rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:18 Reserved, forced by hardware to 0.

Bit 17 SLKIE: Sleep Interrupt Enable

0: No interrupt when SLAKI bit is set.

1: Interrupt generated when SLAKI bit is set.

Bit 16 WKUIE: Wakeup Interrupt Enable

0: No interrupt when WKUI is set.

1: Interrupt generated when WKUI bit is set.

Bit 15 ERRIE: Error Interrupt Enable

0: No interrupt will be generated when an error condition is pending in the CAN_ESR.

1: An interrupt will be generated when an error condition is pending in the CAN_ESR.

Bits 14:12 Reserved, forced by hardware to 0.

Bit 11 LECIE: Last Error Code Interrupt Enable

0: ERRI bit will not be set when the error code in LEC[2:0] is set by hardware on error detection.

1: ERRI bit will be set when the error code in LEC[2:0] is set by hardware on error detection.

Bit 10 BOFIE: Bus-Off Interrupt Enable

0: ERRI bit will not be set when BOFF is set.

1: ERRI bit will be set when BOFF is set.

Bit 9 EPVIE: Error Passive Interrupt Enable

0: ERRI bit will not be set when EPVF is set.

1: ERRI bit will be set when EPVF is set.

Bit 8 EWGIE: Error Warning Interrupt Enable

0: ERRI bit will not be set when EWGF is set.

1: ERRI bit will be set when EWGF is set.

Bit 7 Reserved, forced by hardware to 0.

Bit 6 FOVIE1: FIFO Overrun Interrupt Enable

0: No interrupt when FOVR is set.

1: Interrupt generated 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 14.6: Interrupts](#).

CAN error status register (CAN_ESR)

Address offset: 0x18

Reset value: 0x00

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
REC[7:0]								TEC[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								LEC[2:0]			Res.	BOFF	EPVF	EWGF	
								rw	rw	rw		r	r	r	

Bits 31:24 REC[7:0]: Receive Error Counter

The implementing part of the fault confinement mechanism of the CAN protocol. In case of an error during reception, this counter is incremented by 1 or by 8 depending on the error condition as defined by the CAN standard. After every successful reception the counter is decremented by 1 or reset to 120 if its value was higher than 128. When the counter value exceeds 127, the CAN controller enters the error passive state.

Bits 23:16 TEC[7:0]: least significant byte of the 9-bit Transmit Error Counter

The implementing part of the fault confinement mechanism of the CAN protocol.

Bits 15:7 Reserved, forced by hardware to 0.

Bits 6:4 LEC[2:0]: Last Error Code

This field is set by hardware and holds a code which indicates the error condition of the last error detected on the CAN bus. If a message has been transferred (reception or transmission) without error, this field will be cleared to '0'.

Code 7 is unused and may be written by the hardware to check for an update

000: No Error

001: Stuff Error

010: Form Error

011: Acknowledgment Error

100: Bit recessive Error

101: Bit dominant Error

110: CRC Error

111: Set by software

Bit 3 Reserved, forced by hardware to 0.

Bit 2 BOFF: Bus-Off Flag

This bit is set by hardware when it enters the bus-off state. The bus-off state is entered on TEC overflow, greater than 255, refer to [Section 14.5.6 on page 289](#).

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 \geq 96).

CAN bit timing register (CAN_BTR)

Address offset: 0x1C

Reset value: 0x0123 0000

Note: *This register can only be accessed by the software when the CAN hardware is in initialization mode.*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SILM	LBKM	Reserved				SJW[1:0]	Res.	TS2[2:0]				TS1[3:0]			
rw	rw					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	Reserved				BRP[9:0]										
rw					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **SILM:** *Silent Mode (Debug)*

0: Normal operation

1: Silent Mode

Bit 30 **LBKM:** *Loop Back Mode (Debug)*

0: Loop Back Mode disabled

1: Loop Back Mode enabled

Bits 29:26 Reserved, forced by hardware to 0.

Bits 25:24 **SJW[1:0]: Resynchronization Jump Width**

These bits define the maximum number of time quanta the CAN hardware is allowed to lengthen or shorten a bit to perform the resynchronization.

$$t_{RJW} = t_{CAN} \times (SJW[1:0] + 1)$$

Bit 23 Reserved, forced by hardware to 0.

Bits 22:20 **TS2[2:0]: Time Segment 2**

These bits define the number of time quanta in Time Segment 2.

$$t_{BS2} = t_{CAN} \times (TS2[2:0] + 1)$$

Bits 19:16 **TS1[3:0]: Time Segment 1**

These bits define the number of time quanta in Time Segment 1

$$t_{BS1} = t_{CAN} \times (TS1[3:0] + 1)$$

For more information on bit timing, please refer to [Section 14.5.7: Bit timing on page 289](#).

Bits 15:10 Reserved, forced by hardware to 0.

Bits 9:0 **BRP[9:0]: Baud Rate Prescaler**

These bits define the length of a time quanta.

$$t_q = (BRP[9:0]+1) \times t_{PCLK}$$

14.8.2 Mailbox registers

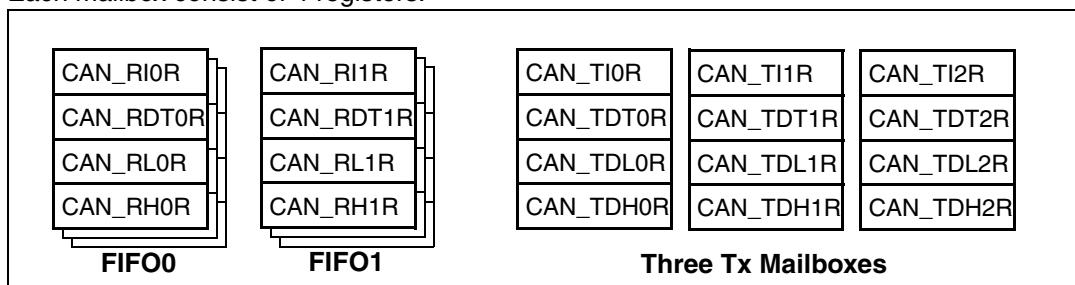
This chapter describes the registers of the transmit and receive mailboxes. Refer to [Section 14.5.5: Message storage on page 287](#) for detailed register mapping.

Transmit and receive mailboxes have the same registers except:

- The FMI field in the CAN_RDTxR register.
- A receive mailbox is always write protected.
- A transmit mailbox is write-enabled only while empty, corresponding TME bit in the CAN_TSR register set.

There are 3 TX Mailboxes and 2 RX Mailboxes. Each RX Mailbox allows access to a 3 level depth FIFO, the access being offered only to the oldest received message in the FIFO.

Each mailbox consist of 4 registers.



TX mailbox identifier register (CAN_TIxR) (x=0..2)

Address offsets: 0x180, 0x190, 0x1A0

Reset value: 0XX where X is undefined (except bit 0, TXRQ = 0)

- Note:
- 1 All TX registers are write protected when the mailbox is pending transmission (TME_x reset).
 - 2 This register also implements the TX request control (bit 0) - reset value 0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
STID[10:0]												EXID[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]: Standard Identifier**

The standard part of the identifier.

Bit 20:3 **EXID[17:0]: Extended Identifier**

The extended part of the identifier.

Bit 2 **IDE: Identifier Extension**

This bit defines the identifier type of message in the mailbox.

0: Standard identifier.

1: Extended identifier.

Bit 1 **RTR:** *Remote Transmission Request*

- 0: Data frame
- 1: Remote frame

Bit 0 **TXRQ:** *Transmit Mailbox Request*

Set by software to request the transmission for the corresponding mailbox.
Cleared by hardware when the mailbox becomes empty.

Mailbox data length control and time stamp register (CAN_TDTxR) (x=0..2)

All bits of this register are write protected when the mailbox is not in empty state.

Address offsets: 0x184, 0x194, 0x1A4

Reset value: 0XX where X is undefined

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

Bits 31:16 **TIME[15:0]: Message Time Stamp**

This field contains the 16-bit timer value captured at the SOF transmission.

Bits 15:9 Reserved

Bit 8 **TGT:** *Transmit Global Time*

This bit is active only when the hardware is in the Time Trigger Communication mode, TTCM bit of the CAN_MCR register is set.

0: Time stamp TIME[15:0] is not sent.

1: Time stamp TIME[15:0] value is sent in the last two data bytes of the 8-byte message: TIME[7:0] in data byte 6 and TIME[15:8] in data byte 7, replacing the data written in CAN_TDHxR[31:16] register (DATA6[7:0] and DATA7[7:0]). DLC must be programmed as 8 in order these two bytes to be sent over the CAN bus.

Bits 7:4 Reserved

Bits 3:0 **DLC[3:0]: Data Length Code**

This field defines the number of data bytes a data frame contains or a remote frame request.

A message can contain from 0 to 8 data bytes, depending on the value in the DLC field.

Mailbox data low register (CAN_TDLxR) (x=0..2)

All bits of this register are write protected when the mailbox is not in empty state.

Address offsets: 0x188, 0x198, 0x1A8

Reset value: 0XX where X is undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA3[7:0]								DATA2[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA1[7:0]								DATA0[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **DATA3[7:0]: Data Byte 3**

Data byte 3 of the message.

Bits 23:16 **DATA2[7:0]: Data Byte 2**

Data byte 2 of the message.

Bits 15:8 **DATA1[7:0]: Data Byte 1**

Data byte 1 of the message.

Bits 7:0 **DATA0[7:0]: Data Byte 0**

Data byte 0 of the message.

A message can contain from 0 to 8 data bytes and starts with byte 0.

Mailbox data high register (CAN_TDhxR) (x=0..2)

All bits of this register are write protected when the mailbox is not in empty state.

Address offsets: 0x18C, 0x19C, 0x1AC

Reset value: 0XX where X is undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA7[7:0]								DATA6[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA5[7:0]								DATA4[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **DATA7[7:0]: Data Byte 7**

Data byte 7 of the message.

Note: if TGT of this message and TTCM are active, DATA7 and DATA6 will be replaced by the TIME stamp value.

Bits 23:16 **DATA6[7:0]: Data Byte 6**

Data byte 6 of the message.

Bits 15:8 **DATA5[7:0]: Data Byte 5**

Data byte 5 of the message.

Bits 7:0 **DATA4[7:0]: Data Byte 4**

Data byte 4 of the message.

Rx FIFO mailbox identifier register (CAN_RIxR) (x=0..1)

Address offsets: 0x1B0, 0x1C0

Reset value: 0XX where X is undefined

Note: All RX registers are write protected.

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

Bits 31:21 **STID[10:0]: Standard Identifier**

The standard part of the identifier.

Bits 20:3 **EXID[17:0]: Extended Identifier**

The extended part of the identifier.

Bit 2 **IDE: Identifier Extension**

This bit defines the identifier type of message in the mailbox.

0: Standard identifier.

1: Extended identifier.

Bit 1 **RTR: Remote Transmission Request**

0: Data frame

1: Remote frame

Bit 0 Reserved

Receive FIFO mailbox data length control and time stamp register (CAN_RDTxR) (x=0..1)

Address offsets: 0x1B4, 0x1C4

Reset value: 0XX where X is undefined

Note: All RX registers are write protected.

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

Bits 31:16 **TIME[15:0]: Message Time Stamp**

This field contains the 16-bit timer value captured at the SOF detection.

Bits 15:8 **FMI[7:0]: Filter Match Index**

This register contains the index of the filter the message stored in the mailbox passed through. For more details on identifier filtering please refer to [Section 14.5.4: Identifier filtering on page 283 - Filter Match Index](#) paragraph.

Bits 7:4 Reserved, forced by hardware to 0.

Bits 3:0 **DLC[3:0]: Data Length Code**

This field defines the number of data bytes a data frame contains (0 to 8). It is 0 in the case of a remote frame request.

Receive FIFO mailbox data low register (CAN_RDLxR) (x=0..1)

All bits of this register are write protected when the mailbox is not in empty state.

Address offsets: 0x1B8, 0x1C8

Reset value: 0XX where X is undefined

Note: All RX registers are write protected.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA3[7:0]								DATA2[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA1[7:0]								DATA0[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:24 **DATA3[7:0]: Data Byte 3**

Data byte 3 of the message.

Bits 23:16 **DATA2[7:0]: Data Byte 2**

Data byte 2 of the message.

Bits 15:8 **DATA1[7:0]: Data Byte 1**
Data byte 1 of the message.

Bits 7:0 **DATA0[7:0]: Data Byte 0**
Data byte 0 of the message.
A message can contain from 0 to 8 data bytes and starts with byte 0.

Receive FIFO mailbox data high register (CAN_RDHxR) (x=0..1)

Address offsets: 0x1BC, 0x1CC
Reset value: 0XX where X is undefined

Note: All RX registers are write protected.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA7[7:0]								DATA6[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA5[7:0]								DATA4[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:24 **DATA7[7:0]: Data Byte 7**
Data byte 3 of the message.

Bits 23:16 **DATA6[7:0]: Data Byte 6**
Data byte 2 of the message.

Bits 15:8 **DATA5[7:0]: Data Byte 5**
Data byte 1 of the message.

Bits 7:0 **DATA4[7:0]: Data Byte 4**
Data byte 0 of the message.

14.8.3 CAN filter registers

CAN filter master register (CAN_FMR)

Address offset: 0x200

Reset value: 0x2A1C 0E01

Note: All bits of this register are set and cleared by software.

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

rw

Bits 31:1 Reserved, forced to reset value

Bit 0 **FINIT: Filter Init Mode**

Initialization mode for filter banks

0: Active filters mode.

1: Initialization mode for the filters.

CAN filter mode register (CAN_FM1R)

Address offset: 0x204

Reset value: 0x00

Note: This register can be written only when the filter initialization mode is set (FINIT=1) in the CAN_FMR register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	FBM13	FBM12	FBM11	FBM10	FBM9	FBM8	FBM7	FBM6	FBM5	FBM4	FBM3	FBM2	FBM1	FBM0	

rw rw

Note: Please refer to [Figure 127: Filter bank scale configuration - register organization on page 285](#)

Bits 31:14 Reserved. Forced to 0 by hardware.

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: 0x00

Note: *This register can be written only when the filter initialization mode is set (FINIT=1) in the CAN_FMR register.*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	FSC13	FSC12	FSC11	FSC10	FSC9	FSC8	FSC7	FSC6	FSC5	FSC4	FSC3	FSC2	FSC1	FSC0	

Note: *Please refer to Figure 127: Filter bank scale configuration - register organization on page 285*

Bits 31:14 Reserved, forced by hardware to 0.

Bits 13:0 **FSCx: Filter Scale Configuration**

These bits define the scale configuration of Filters 13-0.

0: Dual 16-bit scale configuration

1: Single 32-bit scale configuration

CAN filter FIFO assignment register (CAN_FFA1R)

Address offset: 0x214

Reset value: 0x00

Note: *This register can be written only when the filter initialization mode is set (FINIT=1) in the CAN_FMR register.*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	FFA13	FFA12	FFA11	FFA10	FFA9	FFA8	FFA7	FFA6	FFA5	FFA4	FFA3	FFA2	FFA1	FFA0	

Bits 31:14 Reserved, forced by hardware to 0.

Bits 13:0 **FFAx: Filter FIFO Assignment for Filter x**

The message passing through this filter will be stored in the specified FIFO.

0: Filter assigned to FIFO 0

1: Filter assigned to FIFO 1

CAN filter activation register (CAN_FA1R)

Address offset: 0x21C

Reset value: 0x00

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	FACT13	FACT12	FACT11	FACT10	FACT9	FACT8	FACT7	FACT6	FACT5	FACT4	FACT3	FACT2	FACT1	FACT0	

Bits 31:14 Reserved, forced by hardware to 0.

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..0x2AC

Reset value: 0XX where X is undefined

Note: There are 14 filter banks, $i=0..13$. Each filter bank i is composed of two 32-bit registers, CAN_FiR[2:1].

This register can only be modified when the FACTx bit of the CAN_FAxR register is cleared or when the FINIT bit of the CAN_FMR register is set.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FB31	FB30	FB29	FB28	FB27	FB26	FB25	FB24	FB23	FB22	FB21	FB20	FB19	FB18	FB17	FB16
rw															
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: Don't care, the bit is not used for the comparison

1: Must match, the bit of the incoming identifier must have the same level has specified in the corresponding identifier register of the filter.

Note: Depending on the scale and mode configuration of the filter the function of each register can differ. For the filter mapping, functions description and mask registers association, refer to [Section 14.5.4: Identifier filtering on page 283](#).

A Mask/Identifier register in **mask mode** has the same bit mapping as in **identifier list mode**.

For the register mapping/addresses of the filter banks please refer to the [Table 45 on page 314](#).

14.9 bxCAN register map

Refer to [Table 1 on page 27](#) for the register boundary addresses.

Table 45. bxCAN - register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x000	CAN_MCR Reset value																																	
0x004	CAN_MSR Reset value																																	
0x008	CAN_TSR Reset value	LOW[2:0]	TME[2:0]		CODE[1:0]		ABRQ2		Reserved	TERR2	ALST2	TXOK2	RQC2	ABRQ1	Reserved	TERR1	ALST1	TXOK1	ABRQ0	Reserved	RX	SAMP	TXM	TTCM	ABOM	0	0	0	0	0	0			
0x00C	CAN_RF0R Reset value																																	
0x010	CAN_RF1R Reset value																																	
0x014	CAN_IER Reset value																																	
0x018	CAN_ESR Reset value	REC[7:0]								TEC[7:0]																								
0x01C	CAN_BTR Reset value	SILM	LBKM	Reserved	SJW[1:0]	Reserved	TS2[2:0]	TS1[3:0]																										
0x020-0x17F																																		
0x180	CAN_TI0R Reset value						STID[10:0]																											
0x184	CAN_TDT0R Reset value							TIME[15:0]																										
0x188	CAN_TDL0R Reset value							DATA3[7:0]			DATA2[7:0]				DATA1[7:0]																			
0x18C	CAN_TDHO0R Reset value							DATA7[7:0]			DATA6[7:0]				DATA5[7:0]																			
0x190	CAN_TI1R Reset value							STID[10:0]							EXID[17:0]																			

Table 45. 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
0x194	CAN_TDT1R																																
	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			
0x198	CAN_TDL1R																																
	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			
0x19C	CAN_TD1H1R																																
	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			
0x1A0	CAN_TI2R																																
	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	0			
0x1A4	CAN_TDT2R																																
	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				
0x1A8	CAN_TDL2R																																
	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				
0x1AC	CAN_TD2H2R																																
	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				
0x1B0	CAN_RI0R																																
	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	Reserved				
0x1B4	CAN_RDT0R																																
	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					
0x1B8	CAN_RDL0R																																
	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					
0x1BC	CAN_RDH0R																																
	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					
0x1C0	CAN_RI1R																																
	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	Reserved				
0x1C4	CAN_RDT1R																																
	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					
0x1C8	CAN_RDL1R																																
	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					
0x1CC	CAN_RDH1R																																
	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					
0x1D0-0x1FF																																	

Table 45. 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					
0x200	CAN_FMR																																					
	Reset value																																					
0x204	CAN_FM1R																																					
	Reset value																																					
0x208																																						
0x20C	CAN_FS1R																																					
	Reset value																																					
0x210																																						
0x214	CAN_FFA1R																																					
	Reset value																																					
0x218																																						
0x21C	CAN_FA1R																																					
	Reset value																																					
0x220																																						
0x224-0x23F																																						
0x240	CAN_F0R1																																					
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		
0x244	CAN_F0R2																																					
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x			
0x248	CAN_F1R1																																					
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x			
0x24C	CAN_F1R2																																					
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x			
.
0x2A8	CAN_F13R1																																					
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x			
0x2AC	CAN_F13R2																																					
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x			

15 Inter-integrated circuit (I^2C) interface

15.1 Introduction

I^2C (inter-integrated circuit) bus Interface serves as an interface between the microcontroller and the serial I^2C bus. It provides multimaster capability, and controls all I^2C bus-specific sequencing, protocol, arbitration and timing. It supports standard and fast speed modes. It is also SMBus 2.0 compatible.

It may be used for a variety of purposes, including CRC generation and verification, SMBus (system management bus) and PMBus (power management bus).

Depending on specific device implementation DMA capability can be available for reduced CPU overload.

15.2 Main features

- Parallel-bus/ I^2C protocol converter
- Multimaster capability: the same interface can act as Master or Slave
- I^2C Master features:
 - Clock generation
 - Start and Stop generation
- I^2C Slave features:
 - Programmable I^2C Address detection
 - Dual Addressing Capability to acknowledge 2 slave addresses
 - Stop bit detection
- Generation and detection of 7-bit/10-bit addressing and General Call
- Supports different communication speeds:
 - Standard Speed (up to 100 kHz),
 - Fast Speed (up to 400 kHz)
- Status flags:
 - Transmitter/Receiver mode flag
 - End-of-Byte transmission flag
 - I^2C busy flag
- Error flags:
 - Arbitration lost condition for master mode
 - Acknowledgement failure after address/ data transmission
 - Detection of misplaced start or stop condition
 - Overrun/Underrun if clock stretching is disabled
- 2 Interrupt vectors:
 - 1 Interrupt for successful address/ data communication
 - 1 Interrupt for error condition
- Optional Clock Stretching
- 1-byte buffer with DMA capability

- Configurable PEC (Packet Error Checking) Generation or Verification:
 - PEC value can be transmitted as last byte in Tx mode
 - PEC error checking for last received byte
- SMBus 2.0 Compatibility:
 - 25 ms clock low timeout delay
 - 10 ms master cumulative clock low extend time
 - 25 ms slave cumulative clock low extend time
 - Hardware PEC generation/verification with ACK control
 - Address Resolution Protocol (ARP) supported
- PMBus Compatibility

Note: *Some of the above features may not be available in certain products. The user should refer to the product data sheet, to identify the specific features supported by the I^2C interface implementation.*

15.3 General description

In addition to receiving and transmitting data, this interface converts it from serial to parallel format and vice versa. The interrupts are enabled or disabled by software. The interface is connected to the I^2C bus by a data pin (SDA) and by a clock pin (SCL). It can be connected with a standard (up to 100 kHz) or fast (up to 400 kHz) I^2C bus.

Mode selection

The interface can operate in one of the four following modes:

- Slave transmitter
- Slave receiver
- Master transmitter
- Master receiver

By default, it operates in slave mode. The interface automatically switches from slave to master, after it generates a START condition and from master to slave, if an arbitration loss or a Stop generation occurs, allowing multimaster capability.

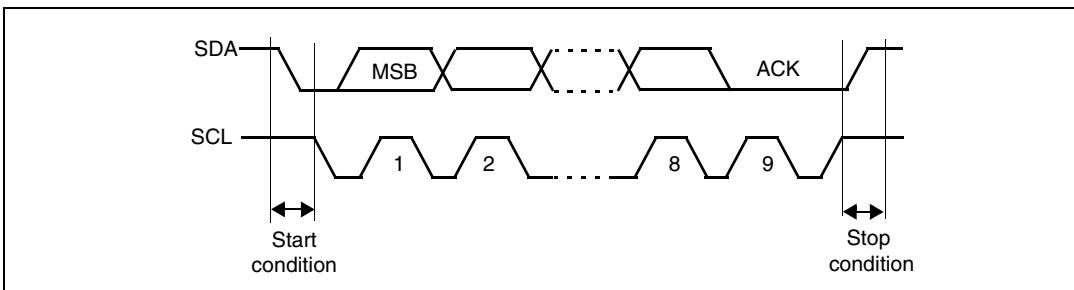
Communication flow

In Master mode, the I^2C interface initiates a data transfer and generates the clock signal. A serial data transfer always begins with a start condition and ends with a stop condition. Both start and stop conditions are generated in master mode by software.

In Slave mode, the interface is capable of recognizing its own addresses (7 or 10-bit), and the General Call address. The General Call address detection may be enabled or disabled by software.

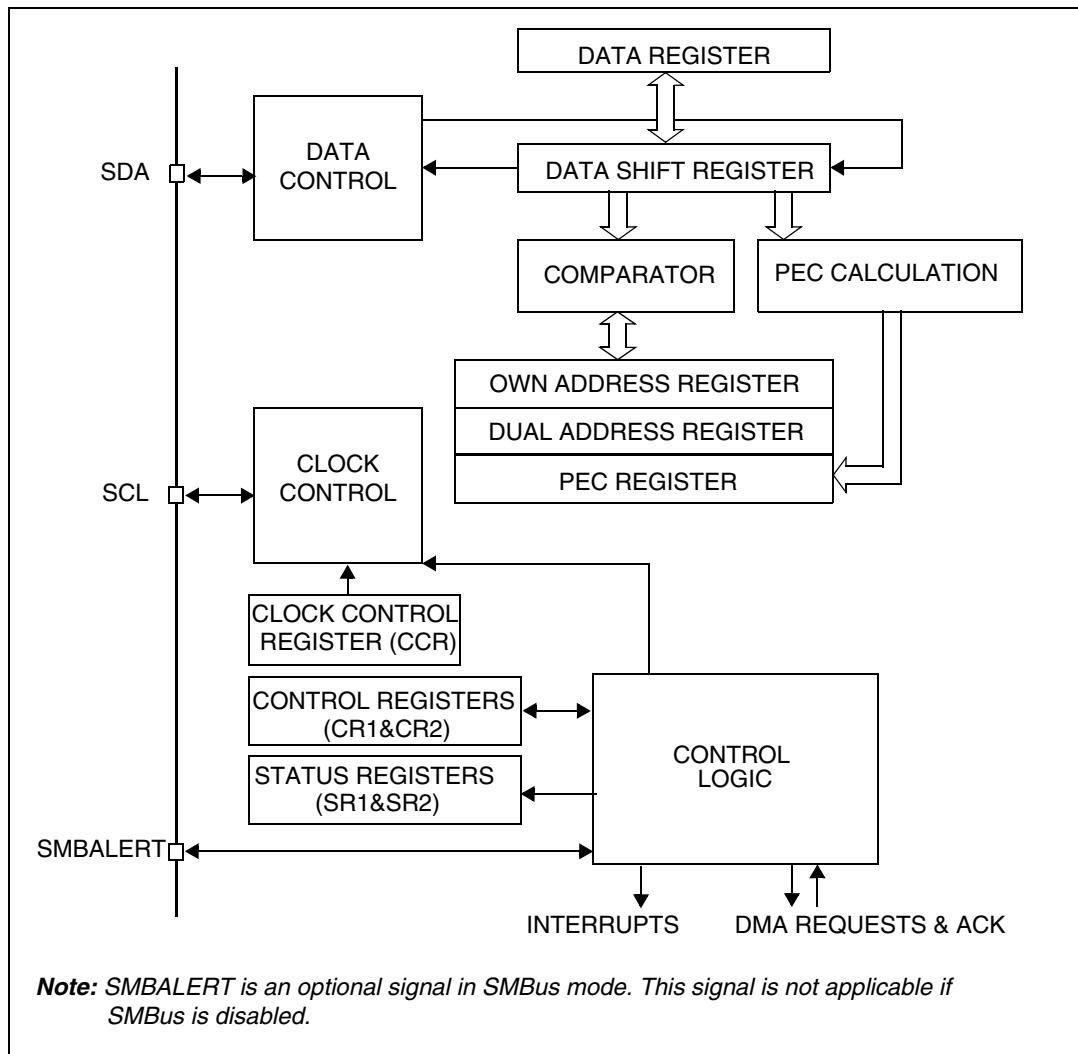
Data and addresses are transferred as 8-bit bytes, MSB first. The first byte(s) following the start condition contain the address (one in 7-bit mode, two in 10-bit mode). The address is always transmitted in Master mode.

A 9th clock pulse follows the 8 clock cycles of a byte transfer, during which the receiver must send an acknowledge bit to the transmitter. Refer to the following figure.

Figure 134. I^2C bus protocol

Acknowledge may be enabled or disabled by software. The I²C interface addresses (dual addressing 7-bit/ 10-bit and/or general call address) can be selected by software.

The block diagram of the I²C interface is shown in [Figure 135](#).

Figure 135. I^2C block diagram

15.4 Functional description

By default the I²C interface operates in Slave mode. To switch from default Slave mode to Master mode a Start condition generation is needed.

15.4.1 I²C slave mode

The peripheral input clock must be programmed in the I2C_CR2 register in order to generate correct timings. The peripheral input clock frequency must be at least:

- 2 MHz in Standard mode
- 4 MHz in Fast mode

As soon as a start condition is detected, the address is received from the SDA line and sent to the shift register. Then it is compared with the address of the interface (OAR1) and with OAR2 (if ENDUAL=1) or the General Call address (if ENGC = 1).

Note: In 10-bit addressing mode, the comparison includes the header sequence (11110xx0), where xx denotes the two most significant bits of the address.

Header or address not matched: the interface ignores it and waits for another Start condition.

Header matched (10-bit mode only): the interface generates an acknowledge pulse if the ACK bit is set and waits for the 8-bit slave address.

Address matched: the interface generates in sequence:

- An acknowledge pulse if the ACK bit is set
- The ADDR bit is set by hardware and an interrupt is generated if the ITEVFEN bit is set.
- If ENDUAL=1, the software has to read the DUALF bit to check which slave address has been acknowledged.

In 10-bit mode, after receiving the address sequence the slave is always in Receiver mode. It will enter Transmitter mode on receiving a repeated Start condition followed by the header sequence with matching address bits and the least significant bit set (11110xx1).

The TRA bit indicates whether the slave is in Receiver or Transmitter mode.

Slave transmitter

Following the address reception and after clearing ADDR, the slave sends bytes from the DR register to the SDA line via the internal shift register.

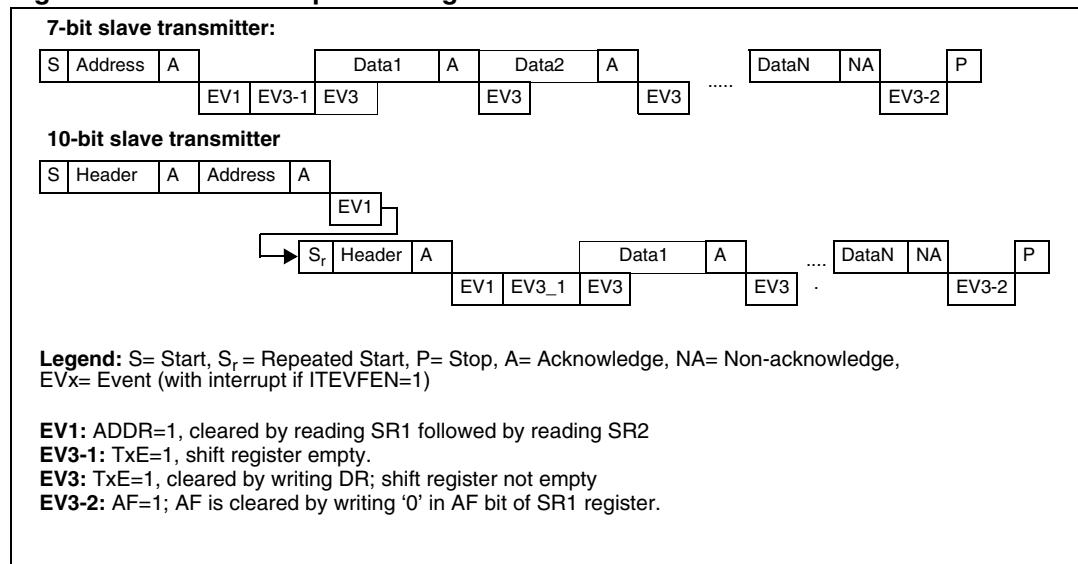
The slave stretches SCL low until ADDR is cleared and DR filled with the data to be sent (see [Figure 136 Transfer sequencing EV1 EV3](#)).

When the acknowledge pulse is received:

- The TxE bit is set by hardware with an interrupt if the ITEVFEN and the ITBUFEN bits are set.

If TxE is set and a data was not written in the DR register before the end of the last data transmission, the BTF bit is set and the interface waits for a write in the DR register, stretching SCL low.

Figure 136. Transfer sequence diagram for slave transmitter



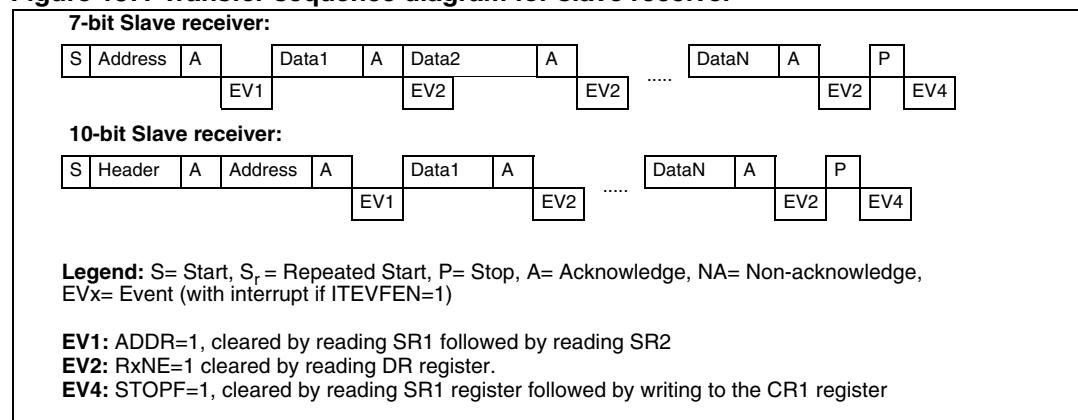
Slave receiver

Following the address reception and after clearing ADDR, the slave receives bytes from the SDA line into the DR register via the internal shift register. After each byte the interface generates in sequence:

- An acknowledge pulse if the ACK bit is set
- The RxNE bit is set by hardware and an interrupt is generated if the ITEVFEN and ITBUFEN bit is set.

If RxNE is set and the data in the DR register is not read before the end of the last data reception, the BTF bit is set and the interface waits for a read to the DR register, stretching SCL low (see [Figure 137 Transfer sequencing](#)).

Figure 137. Transfer sequence diagram for slave receiver



Closing slave communication

After the last data byte is transferred a Stop Condition is generated by the master. The interface detects this condition and sets,

- The STOPF bit and generates an interrupt if the ITEVFEN bit is set.

Then the interface waits for a read of the SR1 register followed by a write to the CR1 register (see [Figure 137 Transfer sequencing](#) EV4).

15.4.2 I²C master mode

In Master mode, the I²C interface initiates a data transfer and generates the clock signal. A serial data transfer always begins with a Start condition and ends with a Stop condition. Master mode is selected as soon as the Start condition is generated on the bus with a START bit.

The following is the required sequence in master mode.

- Program the peripheral input clock in I2C_CR2 Register in order to generate correct timings
- Configure the clock control registers
- Configure the rise time register
- Program the I2C_CR1 register to enable the peripheral
- Set the START bit in the I2C_CR1 register to generate a Start condition

The peripheral input clock frequency must be at least:

- 2 MHz in Standard mode
- 4 MHz in Fast mode

Start condition

Setting the START bit causes the interface to generate a Start condition and to switch to Master mode (M/SL bit set) when the BUSY bit is cleared.

Note: In master mode, setting the START bit causes the interface to generate a ReStart condition at the end of the current byte transfer.

Once the Start condition is sent:

- The SB bit is set by hardware and an interrupt is generated if the ITEVFEN bit is set.

Then the master waits for a read of the SR1 register followed by a write in the DR register with the Slave address (see [Figure 138](#) & [Figure 139](#) Transfer sequencing EV5).

Slave address transmission

Then the slave address is sent to the SDA line via the internal shift register.

- In 10-bit addressing mode, sending the header sequence causes the following event:
 - The ADD10 bit is set by hardware and an interrupt is generated if the ITEVFEN bit is set.

Then the master waits for a read of the SR1 register followed by a write in the DR register with the second address byte (see [Figure 138](#) & [Figure 139](#) Transfer sequencing).

- The ADDR bit is set by hardware and an interrupt is generated if the ITEVFEN bit is set.

Then the master waits for a read of the SR1 register followed by a read of the SR2 register (see [Figure 138](#) & [Figure 139](#) Transfer sequencing).

- In 7-bit addressing mode, one address byte is sent.

As soon as the address byte is sent,

- The ADDR bit is set by hardware and an interrupt is generated if the ITEVFEN bit is set.

Then the master waits for a read of the SR1 register followed by a read of the SR2 register (see [Figure 138](#) & [Figure 139](#) Transfer sequencing).

The master can decide to enter Transmitter or Receiver mode depending on the LSB of the slave address sent.

- In 7-bit addressing mode,
 - To enter Transmitter mode, a master sends the slave address with LSB reset.
 - To enter Receiver mode, a master sends the slave address with LSB set.
- In 10-bit addressing mode,
 - To enter Transmitter mode, a master sends the header (11110xx0) and then the slave address with LSB reset, (where xx denotes the two most significant bits of the address).
 - To enter Receiver mode, a master sends the header (11110xx0) and then the slave address with LSB reset. Then it should send a repeated Start condition followed by the header (11110xx1), (where xx denotes the two most significant bits of the address).

The TRA bit indicates whether the master is in Receiver or Transmitter mode.

Master transmitter

Following the address transmission and after clearing ADDR, the master sends bytes from the DR register to the SDA line via the internal shift register.

The master waits until TxE is cleared, (see [Figure 138](#) Transfer sequencing EV8).

When the acknowledge pulse is received:

- The TxE bit is set by hardware and an interrupt is generated if the ITEVFEN and ITBUFEN bits are set.

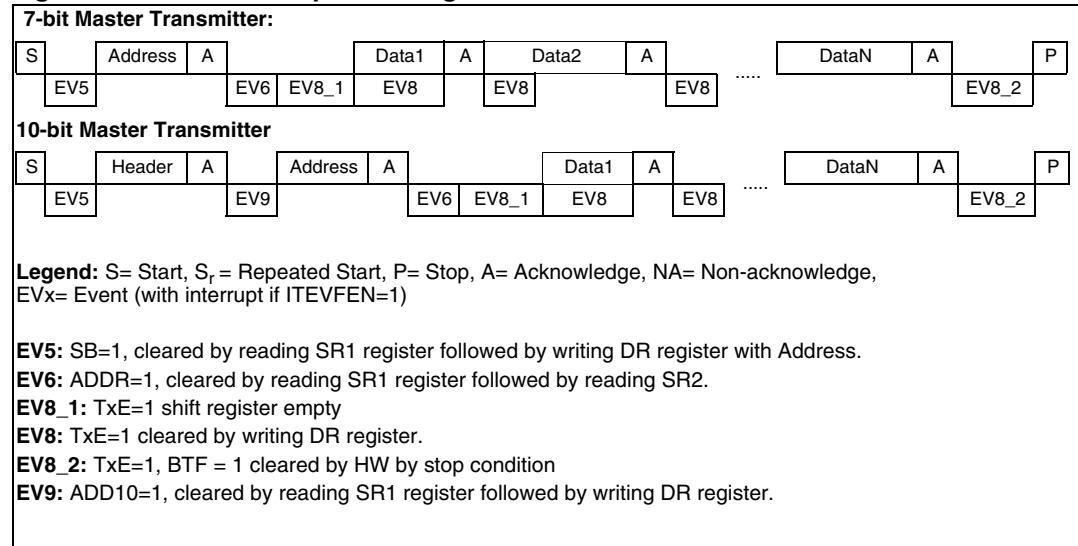
If TxE is set and a data byte was not written in the DR register before the end of the last data transmission, BTF is set and the interface waits until BTF is cleared.

Closing the communication

After writing the last byte to the DR register, the STOP bit is set by software to generate a Stop condition (see *Figure 138* Transfer sequencing EV8_2). The interface goes automatically back to slave mode (M/SL bit cleared).

Note: Stop condition should be programmed during EV8_2 event, when either TxE or BTF is set.

Figure 138. Transfer sequence diagram for master transmitter



Master receiver

Following the address transmission and after clearing ADDR, the I^2C interface enters Master Receiver mode. In this mode the interface receives bytes from the SDA line into the DR register via the internal shift register. After each byte the interface generates in sequence:

- An acknowledge pulse if the ACK bit is set
- The RxNE bit is set and an interrupt is generated if the ITEVFEN and ITBUFEN bits are set (see [Figure 139 Transfer sequencing EV7](#)).

If the RxNE bit is set and the data in the DR register is not read before the end of the last data reception, the BTF bit is set by hardware and the interface waits for a read in the DR register.

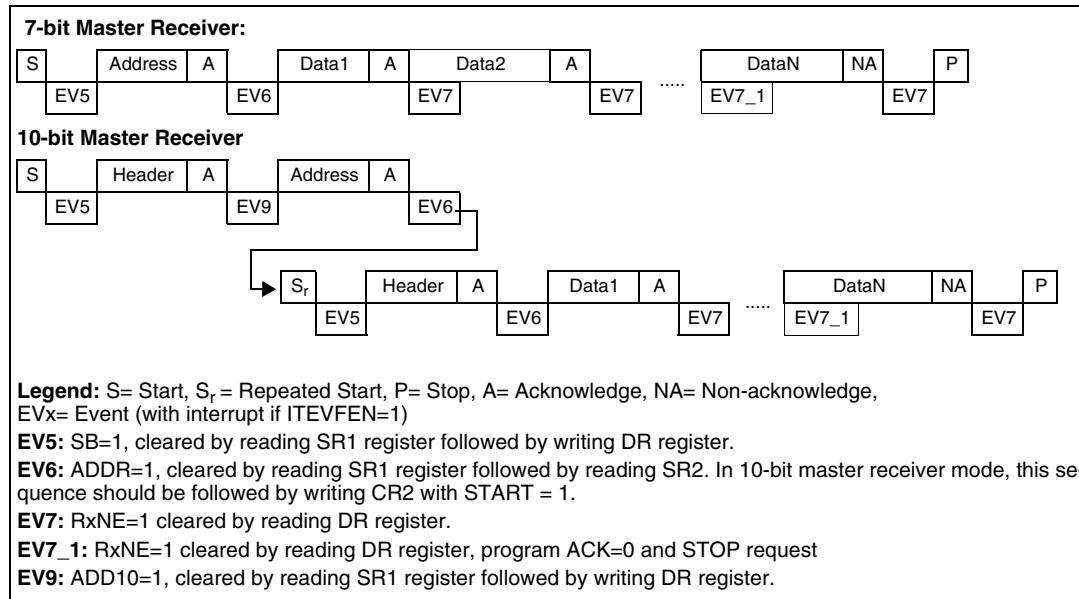
Closing the communication

The master sends a NACK for the last byte received from the slave. After receiving this NACK, the slave releases the control of the SCL and SDA lines. Then the master can send a Stop/Re-Start condition.

- In order to generate the non-acknowledge pulse after the last received data byte, the ACK bit must be cleared just after reading the second last data byte (after second last RxNE event).
- In order to generate the Stop/Re-Start condition, software must set the STOP/START bit just after reading the second last data byte (after the second last RxNE event).

After the Stop condition generation, the interface goes automatically back to slave mode (M/SL bit cleared).

Figure 139. Transfer sequence diagram for master receiver



15.4.3 Error conditions

The following are the error conditions which may cause communication to fail.

Bus error (BERR)

This error occurs when the I²C interface detects a Stop or a Start condition during a byte transfer. In this case,

- The BERR bit is set and an interrupt is generated if the ITERREN bit is set
- In case of Slave: data is discarded and the lines are released by hardware:
 - in case of misplaced start, the slave considers it is a restart and waits for address, or stop condition.
 - in case of misplaced stop, the slave reacts like for a stop condition and the lines are released by hardware.

Acknowledge failure (AF)

This error occurs when the interface detects a non-acknowledge bit. In this case,

- The AF bit is set and an interrupt is generated if the ITERREN bit is set
- A transmitter which receives a NACK must reset the communication:
 - If Slave: lines are released by hardware
 - If Master: a Stop condition must be generated by software

Arbitration lost (ARLO)

This error occurs when the I²C interface detects an arbitration lost condition. In this case,

- The ARLO bit is set by hardware (and an interrupt is generated if the ITERREN bit is set)
- The I²C Interface goes automatically back to slave mode (the M/SL bit is cleared)
- Lines are released by hardware

Overrun/underrun error (OVR)

An Overrun error can occur in slave mode when clock stretching is disabled and the I²C interface is receiving data. The interface has received a byte (RxNE=1) and the data in DR has not been read, before the next byte is received by the interface. In this case,

- The last received byte is lost.
- In case of Overrun error, software should clear the RxNE bit and the transmitter should re-transmit the last received byte.

Underrun error can occur in slave mode when clock stretching is disabled and the I²C interface is transmitting data. The interface has not updated the DR with the next byte (TxE=1), before the clock comes for the next byte. In this case,

- The same byte in the DR register will be sent again
- The user should make sure that data received on the receiver side during an underrun error is discarded and that the next bytes are written within the clock low time specified in the I²C bus standard.

15.4.4 SDA/SCL line control

- If clock stretching is enabled:
 - Transmitter mode: If $TxE=1$ and $BTF=1$: the interface holds the clock line low before transmission to wait for the microcontroller to read SR1 and then write the byte in the Data Register (both buffer and shift register are empty).
 - Receiver mode: If $RxNE=1$ and $BTF=1$: the interface holds the clock line low after reception to wait for the microcontroller to read SR1 and then read the byte in the Data Register (both buffer and shift register are full).
- If clock stretching is disabled in Slave mode:
 - Overrun Error in case of $RxNE=1$ and no read of DR has been done before the next byte is received. The last received byte is lost.
 - Underrun Error in case $TxE=1$ and no write into DR has been done before the next byte must be transmitted. The same byte will be sent again.
 - Write Collision not managed.

15.4.5 SMBus

Introduction

The System Management Bus (SMBus) is a two-wire interface through which various devices can communicate with each other and with the rest of the system. It is based on I^2C principles of operation. SMBus provides a control bus for system and power management related tasks. A system may use SMBus to pass messages to and from devices instead of toggling individual control lines.

The System Management Bus Specification refers to three types of devices. A *slave* is a device that is receiving or responding to a command. A *master* is a device that issues commands, generates the clocks, and terminates the transfer. A *host* is a specialized master that provides the main interface to the system's CPU. A host must be a master-slave and must support the SMBus host notify protocol. Only one host is allowed in a system.

Similarities between SMBus and I^2C

- 2 wire bus protocol (1 Clk, 1 Data) + SMBus Alert line optional
- Master-slave communication, Master provides clock
- Multi master capability
- SMBus data format similar to I^2C 7-bit addressing format ([Figure 134](#)).

Differences between SMBus and I²C

The following table describes the differences between SMBus and I²C.

Table 46. SMBus vs. I²C

SMBus	I ² C
Max. speed 100 kHz	Max. speed 400 kHz
Min. clock speed 10 kHz	No minimum clock speed
35 ms clock low timeout	No timeout
Logic levels are fixed	Logic levels are VDD dependent
Different address types (reserved, dynamic etc.)	7-bit, 10-bit and general call slave address types
Different bus protocols (quick command, process call etc.)	No bus protocols

SMBus application usage

With System Management Bus, a device can provide manufacturer information, tell the system what its model/part number is, save its state for a suspend event, report different types of errors, accept control parameters, and return its status. SMBus provides a control bus for system and power management related tasks.

Device identification

Any device that exists on the System Management Bus as a slave has a unique address called the Slave Address. For the list of reserved slave addresses, refer to the SMBus specification ver. 2.0 (<http://smbus.org/specs/>).

Bus protocols

The SMBus specification supports up to 9 bus protocols. For more details of these protocols and SMBus address types, refer to SMBus specification ver. 2.0 (<http://smbus.org/specs/>). These protocols should be implemented by the user software.

Address resolution protocol (ARP)

SMBus slave address conflicts can be resolved by dynamically assigning a new unique address to each slave device. The Address Resolution Protocol (ARP) has the following attributes:

- Address assignment uses the standard SMBus physical layer arbitration mechanism
- Assigned addresses remain constant while device power is applied; address retention through device power loss is also allowed
- No additional SMBus packet overhead is incurred after address assignment. (i.e. subsequent accesses to assigned slave addresses have the same overhead as accesses to fixed address devices.)
- Any SMBus master can enumerate the bus

Unique device identifier (UDID)

In order to provide a mechanism to isolate each device for the purpose of address assignment, each device must implement a unique device identifier (UDID).

For the details on 128 bit UDID and more information on ARP, refer to SMBus specification ver. 2.0 (<http://smbus.org/specs/>).

SMBus alert mode

SMBus Alert is an optional signal with an interrupt line for devices that want to trade their ability to master for a pin. SMBALERT is a wired-AND signal just as the SCL and SDA signals are. SMBALERT is used in conjunction with the SMBus General Call Address. Messages invoked with the SMBus are 2 bytes long.

A slave-only device can signal the host through SMBALERT that it wants to talk by setting ALERT bit in I2C_CR1 register. The host processes the interrupt and simultaneously accesses all SMBALERT devices through the *Alert Response Address* (known as ARA having a value 0001 100X). Only the device(s) which pulled SMBALERT low will acknowledge the Alert Response Address. This status is identified using SMBALERT Status flag in I2C_SR1 register. The host performs a modified Receive Byte operation. The 7 bit device address provided by the slave transmit device is placed in the 7 most significant bits of the byte. The eighth bit can be a zero or one.

If more than one device pulls SMBALERT low, the highest priority (lowest address) device will win communication rights via standard arbitration during the slave address transfer. After acknowledging the slave address the device must disengage its SMBALERT pull-down. If the host still sees SMBALERT low when the message transfer is complete, it knows to read the ARA again.

A host which does not implement the SMBALERT signal may periodically access the ARA.

For more details on SMBus Alert mode, refer to SMBus specification ver. 2.0 (<http://smbus.org/specs/>).

Timeout error

There are differences in the timing specifications between I^2C and SMBus.

SMBus defines a clock low timeout, TIMEOUT of 35 ms. Also SMBus specifies TLOW: SEXT as the cumulative clock low extend time for a slave device. SMBus specifies TLOW: MEXT as the cumulative clock low extend time for a master device. For more details on these timeouts, refer to SMBus specification ver. 2.0 (<http://smbus.org/specs/>).

The status flag Timeout or Tlow Error in I2C_SR1 shows the status of this feature.

How to use the interface in SMBus mode

To switch from I^2C mode to SMBus mode, the following sequence should be performed.

- Set the SMBus bit in the I2C_CR1 register
- Configure the SMBTYPE and ENARP bits in the I2C_CR1 register as required for the application

If you want to configure the device as a master, follow the Start condition generation procedure in [Section 15.4.2: I²C master mode](#). Otherwise, follow the sequence in [Section 15.4.1: I²C slave mode](#).

The application has to control the various SMBus protocols by software.

- SMB Device Default Address acknowledged if ENARP=1 and SMBTYPE=0
- SMB Host Header acknowledged if ENARP=1 and SMBTYPE=1
- SMB Alert Response Address acknowledged if SMBALERT=1

15.4.6 DMA requests

DMA requests (when enabled) are generated only for data transfer. DMA requests are generated by Data Register becoming empty in transmission and Data Register becoming full in reception. When the number of data transfers which has been programmed for the corresponding DMA channel is reached, the DMA controller sends an End of Transfer EOT signal to the I²C interface and generates a Transfer Complete interrupt if enabled:

- Master Transmitter: In the interrupt routine after the EOT interrupt, disable DMA requests then wait for a BTF event before programming the Stop condition.
- Master Receiver: The DMA controller sends a hardware signal EOT_1 corresponding to the (number of bytes -1). If, in the I2C_CR2 register, the LAST bit is set, I²C automatically sends a NACK after the next byte following EOT_1. The user can generate a Stop condition in the DMA Transfer Complete interrupt routine if enabled.

Note: Please refer to the product specs for availability DMA controller. If DMA is not available in the product, the user should use I²C as explained in section 1.4. In the I²C ISR, the user can clear TxE/ RxNE flags to achieve continuous communication.

Transmission using DMA

DMA mode can be enabled for transmission by setting the DMAEN bit in the I2C_CR2 register. The DMAEN bit must be set only after receiving the address sequence, when ADDR is cleared. Data will be loaded from a Memory area configured using the DMA peripheral (refer to the DMA specification) to the I2C_DR register whenever the TxE bit is set. To map a DMA channel for I²C transmission, perform the following sequence. Here x is the channel number.

1. Set the I2C_DR register address in the DMA_CPARx register. The data will be moved to this address from the memory after each TxE event.
2. Set the memory address in the DMA_CMARx register. The data will be loaded into I2C_DR from this memory after each TxE event.
3. Configure the total number of bytes to be transferred in the DMA_CNDTRx register. After each TxE event, this value will be decremented.
4. Configure the channel priority using the PL[0:1] bits in the DMA_CCRx register
5. Set the DIR bit and, in the DMA_CCRx register, configure interrupts after half transfer or full transfer depending on application requirements.
6. Activate the channel by setting the EN bit in the DMA_CCRx register.

When the number of data transfers which has been programmed in the DMA Controller registers is reached, the DMA controller sends an End of Transfer EOT/ EOT_1 signal to the I²C interface and the DMA generates an interrupt, if enabled, on the DMA channel interrupt vector.

Note: Do not enable the ITEVTEN bit in the I2C_CR2 register if DMA is used for transmission.

Reception using DMA

DMA mode can be enabled for reception by setting the DMAEN bit in the I2C_CR2 register. The DMAEN bit must be set only after receiving the address sequence, when ADDR is cleared. Data will be loaded from the I2C_DR register to a Memory area configured using the DMA peripheral (refer to the DMA specification) whenever a data byte is received. To map a DMA channel for I^2C reception, perform the following sequence. Here x is the channel number.

1. Set the I2C_DR register address in DMA_CPARx register. The data will be moved from this address to the memory after each RxNE event.
2. Set the memory address in the DMA_CMARx register. The data will be loaded from the I2C_DR register to this memory area after each RxNE event.
3. Configure the total number of bytes to be transferred in the DMA_CNDTRx register. After each RxNE event, this value will be decremented.
4. Configure the channel priority using the PL[0:1] bits in the DMA_CCRx register
5. Reset the DIR bit and configure interrupts in the DMA_CCRx register after half transfer or full transfer depending on application requirements.
6. Activate the channel by setting the EN bit in the DMA_CCRx register.

When the number of data transfers which has been programmed in the DMA Controller registers is reached, the DMA controller sends an End of Transfer EOT/ EOT_1 signal to the I^2C interface and DMA generates an interrupt, if enabled, on the DMA channel interrupt vector.

Note: *Do not enable the ITEVTEN bit in the I2C_CR2 register if DMA is used for reception.*

15.4.7 Packet error checking

A PEC calculator has been implemented to improve the reliability of communication. The PEC is calculated by using a programmable polynomial serially on each bit.

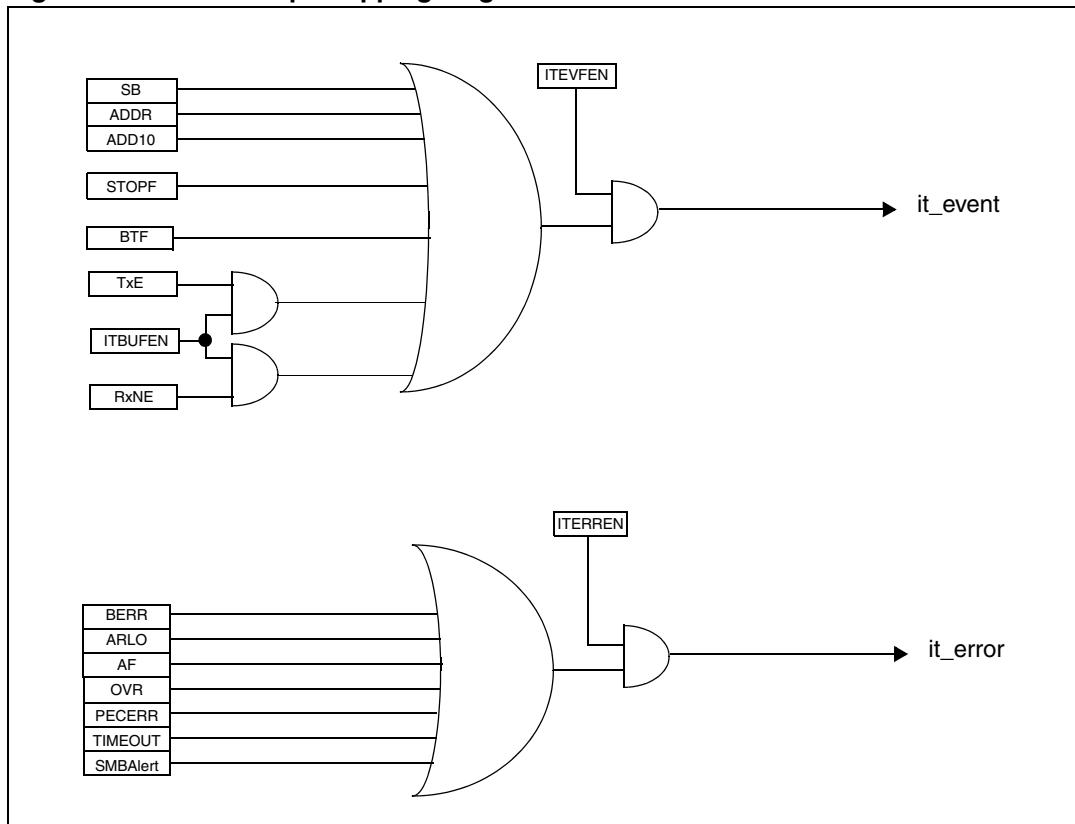
- PEC calculation is enabled by setting the ENPEC bit in the I2C_CR1 register. PEC is a CRC-8 calculated on all message bytes including addresses and R/W bits.
 - In transmission: in the last TxE event: set the PEC transfer bit in the I2C_CR1 register. The PEC will be transferred after the current byte.
 - In reception: in the last RxNE event: set the PEC bit in the I2C_CR1 register so that receiver sends a NACK if the next received byte is not equal to the internally calculated PEC. In case of Master-Receiver, a NACK must follow the PEC whatever the check result.
- A PECERR error flag/interrupt is also available in the I2C_SR1 register.
- If DMA and PEC calculation are both enabled:
 - In transmission: when the I^2C interface receives an EOT signal from the DMA controller, it automatically sends a PEC after the last byte.
 - In reception: when the I^2C interface receives an EOT_1 signal from the DMA controller, it will automatically consider the next byte as a PEC and will check it. A DMA request is generated after PEC reception.
- To allow intermediate PEC transfers, a control bit is available in the I2C_CR2 register (LAST bit) to determine if it is really the last DMA transfer or not. If it is the last DMA request for a master receiver, a NACK is automatically sent after the last received byte.
- PEC calculation is corrupted by an arbitration loss.

15.5 Interrupt requests

Table 47. I²C Interrupt requests

Interrupt event	Event flag	Enable Control bit
Start bit sent (Master)	SB	ITEVFEN
Address sent (Master) or Address matched (Slave)	ADDR	
10-bit header sent (Master)	ADD10	
Stop received (Slave)	STOPF	
Data Byte Transfer Finished	BTF	
Receive buffer not empty	RxNE	ITEVFEN and ITBUFEN
Transmit buffer empty	TxE	
Bus error	BERR	ITERREN
Arbitration loss (Master)	ARLO	
Acknowledge failure	AF	
Overrun/Underrun	OVR	
PEC error	PECERR	
Timeout/Tlow error	TIMEOUT	
SMBus Alert	SMBALERT	

- Note:*
- 1 *SB, ADDR, ADD10, STOPF, BTF, RxNE and TxE are logically ORed on the same interrupt channel.*
 - 2 *BERR, ARLO, AF, OVR, PECERR, TIMEOUT and SMBALERT are logically ORed on the same interrupt channel.*

Figure 140. I²C interrupt mapping diagram

15.6 I²C debug mode

When the microcontroller enters the debug mode (Cortex-M3 core halted), the SMBUS timeout either continues to work normally or stops, depending on the DBG_I2Cx_SMBUS_TIMEOUT configuration bits in the DBG module. For more details, refer to [Section 20.15.2: Debug support for timers, watchdog, bxCAN and I²C on page 487](#).

15.7 I²C register description

Refer to [Section 1.1 on page 23](#) for a list of abbreviations used in register descriptions.

15.7.1 Control register 1(I2C_CR1)

Address offset: 0x000

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SW RST	Res.	ALERT	PEC	POS	ACK	STOP	START	NO STRETCH	ENG C	EN PEC	EN ARP	SMB TYPE	Res.	SM BUS	PE

rw rw

Bit 15 SWRST: Software Reset

When set, the I²C is under reset state. Before resetting this bit, make sure the I²C lines are released and the bus is free.

- 0: I²C Peripheral not under reset
- 1: I²C Peripheral under reset state

Note:

This bit can be used in case the BUSY bit is set to '1' when no stop condition has been detected on the bus.

Bit 14 Reserved, forced by hardware to 0.**Bit 13 ALERT: SMBus Alert**

This bit is set and cleared by software, and cleared by hardware when PE=0.

- 0: Releases SMBAlert pin high. Alert Response Address Header followed by NACK.
- 1: Drives SMBAlert pin low. Alert Response Address Header followed by ACK.

Bit 12 PEC: Packet Error Checking.

This bit is set and cleared by software, and cleared by hardware when PEC is transferred or by a START or Stop condition or when PE=0.

- 0: No PEC transfer
- 1: PEC transfer (in Tx or Rx mode)

Note: PEC calculation is corrupted by an arbitration loss.

Bit 11 POS: Acknowledge/PEC Position (for data reception).

This bit is set and cleared by software and cleared by hardware when PE=0.

- 0: ACK bit controls the (N)ACK of the current byte being received in the shift register. The PEC bit indicates that current byte in shift register is a PEC.
- 1: ACK bit controls the (N)ACK of the next byte which will be received in the shift register. The PEC bit indicates that the next byte in the shift register is a PEC

Note:

This bit must be configured before data reception starts.

This configuration must be used only in ADDR stretch event in case there are only 2 data bytes

Bit 10 ACK: Acknowledge Enable

This bit is set and cleared by software and cleared by hardware when PE=0.

- 0: No acknowledge returned
- 1: Acknowledge returned after a byte is received (matched address or data)

Bit 9 STOP: Stop Generation

The bit is set and cleared by software, cleared by hardware when a Stop condition is detected, set by hardware when a timeout error is detected.

In Master Mode:

- 0: No Stop generation.

1: Stop generation after the current byte transfer or after the current Start condition is sent.

In Slave mode:

- 0: No Stop generation.

1: Release the SCL and SDA lines after the current byte transfer.

Note:

In Master mode, the BTF bit of the I²C_SR1 register must be cleared when Stop is requested.

Bit 8 START: Start Generation

This bit is set and cleared by software and cleared by hardware when start is sent or PE=0.

In Master Mode:

- 0: No Start generation
- 1: Repeated start generation

In Slave mode:

- 0: No Start generation
- 1: Start generation when the bus is free

Bit 7 NOSTRETCH: Clock Stretching Disable (Slave mode)

This bit is used to disable clock stretching in slave mode when ADDR or BTF flag is set, until it is reset by software.

- 0: Clock stretching enabled
- 1: Clock stretching disabled

Bit 6 ENGC: General Call Enable

0: General call disabled. Address 00h is NACKed.

1: General call enabled. Address 00h is ACKed.

Bit 5 ENPEC: PEC Enable

- 0: PEC calculation disabled
- 1: PEC calculation enabled

Bit 4 ENARP: ARP Enable

- 0: ARP disable
- 1: ARP enable

SMBus Device default address recognized if SMBTYPE=0

SMBus Host address recognized if SMBTYPE=1

Bit 3 SMBTYPE: SMBus Type

- 0: SMBus Device
- 1: SMBus Host

Bit 2 Reserved, forced by hardware to 0.

Bit 1 SMBUS: SMBus Mode

- 0: I²C mode
- 1: SMBus mode

Bit 0 PE: Peripheral Enable

- 0: Peripheral disable
- 1: Peripheral enable: the corresponding I/Os are selected as alternate functions depending on SMBus bit.

Note:

If this bit is reset while a communication is on going, the peripheral is disabled at the end of the current communication, when back to IDLE state.

All bit resets due to PE=0 occur at the end of the communication.

In master mode, this bit must not be reset before the end of the communication.

15.7.2 Control register 2 (I2C_CR2)

Address offset: 0x04

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		LAST	DMA EN	ITBUF EN	ITEVT EN	ITER REN	Reserved					FREQ[5:0]			

rw rw

Bits 15:13 Reserved, forced by hardware to 0.

Bit 12 **LAST**: DMA Last Transfer

0: Next DMA EOT is not the last transfer

1: Next DMA EOT is the last transfer

Note:

This bit is used in master receiver mode to permit the generation of a NACK on the last received data.

Bit 11 **DMAEN**: DMA Requests Enable

0: DMA requests disabled

1: DMA request enabled when TxE=1 or RxNE =1

Note: The DMAEN bit must be set only after receiving the address sequence, when ADDR is cleared.

Bit 10 **ITBUFEN**: Buffer Interrupt Enable

0: TxE = 1 or RxNE = 1 does not generate any interrupt.

1:TxE = 1 or RxNE = 1 generates Event Interrupt (whatever the state of DMAEN)

Bit 9 **ITEVTEN**: Event Interrupt Enable

0: Event interrupt disabled

1: Event interrupt enabled

This interrupt is generated when:

- SB = 1 (Master)
- ADDR = 1 (Master/Slave)
- ADD10= 1 (Master)
- STOPF = 1 (Slave)
- BTF = 1 with no TxE or RxNE event
- TxE event to 1 if ITBUFEN = 1
- RxNE event to 1if ITBUFEN = 1

Bit 8 **ITERREN**: Error Interrupt Enable

0: Error interrupt disabled

1: Error interrupt enabled

This interrupt is generated when:

- BERR = 1
- ARLO = 1
- AF = 1
- OVR = 1
- PECERR = 1
- TIMEOUT = 1
- SMBAlert = 1

Bits 7:6 Reserved, forced by hardware to 0.

Bits 5:0 **FREQ[5:0]: Peripheral Clock Frequency**

Input clock frequency must be programmed to generate correct timings

The allowed range is between 2 MHz and 36 MHz

000000: Not allowed

000001: Not allowed

000010: 2 MHz

...

100100: 36 MHz

Higher than 100100: Not allowed

15.7.3 Own address register 1 (I2C_OAR1)

Reset Address offset: 0x08

Value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD MODE	Res.	Reserved			ADD[9:8]	ADD[7:1]								ADD0	
rw	rw				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 **ADDMODE: Addressing Mode (Slave mode)**

0: 7-bit slave address (10-bit address not acknowledged)

1: 10-bit slave address (7-bit address not acknowledged)

Bit 14 Must be configured and kept at 1.

Bits 13:10 Reserved, forced by hardware to 0.

Bits 9:8 **ADD[9:8]: Interface Address**

7-bit addressing mode: don't care

10-bit addressing mode: bits9:8 of address

Bits 7:1 **ADD[7:1]: Interface Address**

bits 7:1 of address

Bit 0 **ADD0: Interface Address**

7-bit addressing mode: don't care

10-bit addressing mode: bit 0 of address

15.7.4 Own address register 2 (I2C_OAR2)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								ADD2[7:1]							
								rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:8 Reserved, forced by hardware to 0.

Bits 7:1 **ADD2[7:1]: Interface address**

bits 7:1 of address in dual addressing mode

Bit 0 **ENDUAL**: *Dual addressing mode enable*

0: Only OAR1 is recognized in 7-bit addressing mode

1: Both OAR1 and OAR2 are recognized in 7-bit addressing mode

15.7.5 Data register (I2C_DR)

Address offset: 0x10

Reset value: 0x0000

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

Bits 15:8 Reserved, forced by hardware to 0.

Bits 7:0 **DR[7:0]** 8-bit *Data Register*⁽¹⁾⁽²⁾⁽³⁾

Byte received or to be transmitted to the bus.

- Transmitter mode: Byte transmission starts automatically when a byte is written in the DR register. A continuous transmit stream can be maintained if the next data to be transmitted is put in DR once the transmission is started (TxE=1)
- Receiver mode: Received byte is copied into DR (RxNE=1). The received data in the DR register must be read before the next data reception, otherwise an overrun occurs and the last byte will be lost.

1. In slave mode, the address is not copied into DR.
2. Write collision is not managed (DR can be written if TxE=0).
3. If an ARLO event occurs on ACK pulse, the received byte is not copied into DR and so cannot be read.

15.7.6 Status register 1 (I²C_SR1)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIME OUT	Res.	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res.	STOP F	ADD10	BTF	ADDR	SB

Bit 15 **SMBALERT**: SMBus Alert

In SMBus host mode:

0: no SMBAlert

1: SMBAlert event occurred on pin

In SMBus slave mode:

0: no SMBAlert response address header

1: SMBAlert response address header to SMBAlert LOW received

- Cleared by software writing 0, or by hardware when PE=0.

Bit 14 **TIMEOUT**: Timeout or Tlow Error

0: No timeout error

1: SCL remained LOW for 25 ms (Timeout)

or

Master cumulative clock low extend time more than 10 ms (Tlow:mext)

or

Slave cumulative clock low extend time more than 25 ms (Tlow:sext)

- When set in slave mode: slave resets the communication and lines are released by hardware
- When set in master mode: Stop condition sent by hardware
- Cleared by software writing 0, or by hardware when PE=0.

Bit 13 Reserved, forced by hardware to 0.

Bit 12 **PECERR**: PEC Error in reception

0: no PEC error: receiver returns ACK after PEC reception (if ACK=1)

1: PEC error: receiver returns NACK after PEC reception (whatever ACK)

- Cleared by software writing 0, or by hardware when PE=0.

Bit 11 **OVR**: Overrun/Underrun

0: No overrun/underrun

1: Overrun or underrun

– Set by hardware in slave mode when NOSTRETCH=1 and:

- In reception when a new byte is received (including ACK pulse) and the DR register has not been read yet. New received byte is lost.
- In transmission when a new byte should be sent and the DR register has not been written yet. The same byte is sent twice.
- Cleared by software writing 0, or by hardware when PE=0.

Note:

If the DR write occurs very close to SCL rising edge, the sent data is unspecified and a hold timing error occurs

Bit 10 **AF**: *Acknowledge Failure*.

- 0: No acknowledge failure
- 1: Acknowledge failure
 - Set by hardware when no acknowledge is returned.
 - Cleared by software writing 0, or by hardware when PE=0.

Bit 9 **ARLO**: *Arbitration Lost (master mode)*

- 0: No Arbitration Lost detected
 - 1: Arbitration Lost detected
 - Set by hardware when the interface loses the arbitration of the bus to another master
 - Cleared by software writing 0, or by hardware when PE=0.
- After an ARLO event the interface switches back automatically to Slave mode (M/SL=0).

Note:

In SMBUS, the arbitration on the data in slave mode occurs only during the data phase, or the acknowledge transmission (not on the address acknowledge).

Bit 8 **BERR**: *Bus Error*

- 0: No misplaced Start or Stop condition
- 1: Misplaced Start or Stop condition
 - Set by hardware when the interface detects a misplaced Start or Stop condition
 - Cleared by software writing 0, or by hardware when PE=0.

Bit 7 **TxE**: *Data Register Empty (transmitters)*

- 0: Data register not empty
- 1: Data register empty
 - Set when DR is empty in transmission. TxE is not set during address phase.
 - Cleared by software writing to the DR register or by hardware after a start or a stop condition or when PE=0.

TxE is not set if either a NACK is received, or if next byte to be transmitted is PEC (PEC=1)

Bit 6 **RxNE**: *Data Register not Empty (receivers)*.

- 0: Data register empty
- 1: Data register not empty
 - Set when data register is not empty in receiver mode. RxNE is not set during address phase.
 - Cleared by software reading or writing the DR register or by hardware when PE=0.

RxNE is not set in case of ARLO event.

Bit 5 Reserved, forced by hardware to 0.

Bit 4 **STOPF**: *Stop detection (Slave mode)*

- 0: No Stop condition detected
- 1: Stop condition detected
 - Set by hardware when a Stop condition is detected on the bus by the slave after an acknowledge (if ACK=1).
 - Cleared by software reading the SR1 register followed by a write in the CR1 register, or by hardware when PE=0

Note:

The STOPF bit is not set after a NACK reception

Bit 3 ADD10: 10-bit header sent (Master mode)

- 0: No ADD10 event occurred.
- 1: Master has sent first address byte (header).
 - Set by hardware when the master has sent the first byte in 10-bit address mode.
 - Cleared by software reading the SR1 register followed by a write in the DR register of the second address byte, or by hardware when PE=0.

Note:

ADD10 bit is not set after a NACK reception

Bit 2 BTF: Byte Transfer Finished.

- 0: Data Byte transfer not done
- 1: Data Byte transfer succeeded
 - Set by hardware when NOSTRETCH=0 and:
 - In reception when a new byte is received (including ACK pulse) and DR has not been read yet (RxNE=1).
 - In transmission when a new byte should be sent and DR has not been written yet (TxEN=1).
 - Cleared by software reading SR1 followed by either a read or write in the DR register or by hardware after a start or a stop condition in transmission or when PE=0.

Note:

The BTF bit is not set after a NACK reception

The BTF bit is not set if next byte to be transmitted is the PEC (TRA=1 in I2C_SR2 register and PEC=1 in I2C_CR1 register)

Bit 1 ADDR: Address sent (master mode)/matched (slave mode)

This bit is cleared by software reading SR1 register followed reading SR2, or by hardware when PE=0.

Address Matched (Slave)

- 0: Address mismatched or not received.
- 1: Received address matched.
 - Set by hardware as soon as the received slave address matched with the OAR registers content or a general call or a SMBus Device Default Address or SMBus Host or SMBus Alert is recognized. (when enabled depending on configuration).

Address Sent (Master)

- 0: No end of address transmission
- 1: End of address transmission
 - For 10-bit addressing, the bit is set after the ACK of the 2nd byte.
 - For 7-bit addressing, the bit is set after the ACK of the byte.

Note:

ADDR is not set after a NACK reception

Bit 0 SB: Start Bit (Master mode).

- 0: No Start condition
- 1: Start condition generated.
 - Set when a Start condition generated.
 - Cleared by software by reading the SR1 register followed by writing the DR register, or by hardware when PE=0

15.7.7 Status register 2 (I²C_SR2)

Address offset: 0x18

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
							PEC[7:0]		DUALF	SMB HOST	SMB DEF AULT	GEN CALL	Res.	TRA	BUSY	MSL

r r r r r r r r r r r r r r r r

Bits 15:8 **PEC[7:0] Packet Error Checking Register**

This register contains the internal PEC when ENPEC=1.

Bit 7 **DUALF: Dual Flag (Slave mode)**

0: Received address matched with OAR1

1: Received address matched with OAR2

– Cleared by hardware after a Stop condition or repeated Start condition, or when PE=0.

Bit 6 **SMBHOST: SMBus Host Header (Slave mode)**

0: No SMBus Host address

1: SMBus Host address received when SMBTYPE=1 and ENARP=1.

– Cleared by hardware after a Stop condition or repeated Start condition, or when PE=0.

Bit 5 **SMBDEFAULT: SMBus Device Default Address (Slave mode)**

0: No SMBus Device Default address

1: SMBus Device Default address received when ENARP=1

– Cleared by hardware after a Stop condition or repeated Start condition, or when PE=0.

Bit 4 **GENDCALL: General Call Address (Slave mode)**

0: No General Call

1: General Call Address received when ENGC=1

– Cleared by hardware after a Stop condition or repeated Start condition, or when PE=0.

Bit 3 Reserved, forced by hardware to 0.

Bit 2 **TRA: Transmitter/Receiver**

0: Data bytes received

1: Data bytes transmitted

This bit is set depending on R/W bit of address byte, at the end of total address phase.

It is also cleared by hardware after detection of Stop condition (STOPF=1), repeated Start condition, loss of bus arbitration (ARLO=1), or when PE=0.

Bit 1 **BUSY: Bus Busy**

0: No communication on the bus

1: Communication ongoing on the bus

– Set by hardware on detection of SDA or SCL low

– cleared by hardware on detection of a Stop condition.

It indicates a communication in progress on the bus. This information is still updated when the interface is disabled (PE=0).

Bit 0 **MSL: Master/Slave**

- 0: Slave Mode
- 1: Master Mode
 - Set by hardware as soon as the interface is in Master mode (SB=1).
 - Cleared by hardware after detecting a Stop condition on the bus or a loss of arbitration (ARLO=1), or by hardware when PE=0.

15.7.8 Clock control register (I2C_CCR)

Address offset: 0x1C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F/S	DUTY	Reserved													CCR[11:0]

rw rw rw rw rw rw rw rw rw rw rw rw rw rw rw

Bit 15 **F/S I²C Master Mode Selection**

- 0: Standard Mode I²C
- 1: Fast Mode I²C

Bit 14 **DUTY Fast Mode Duty Cycle**

- 0: Fast Mode $t_{low}/t_{high} = 2$
- 1: Fast Mode $t_{low}/t_{high} = 16/9$ (see CCR)

Bits 13:12 Reserved, forced by hardware to 0.

Bits 11:0 **CCR[11:0] Clock Control Register in Fast/Standard mode (Master mode)**

Controls the SCL clock in master mode.

Standard Mode or SMBus:

$$T_{high} = CCR * T_{CK}$$

$$T_{ow} = CCR * T_{CK}$$

Fast Mode:

If DUTY = 0:

$$T_{high} = CCR * T_{CK}$$

$$T_{ow} = 2 * CCR * T_{CK}$$

If DUTY = 1: (to reach 400 kHz)

$$T_{high} = 9 * CCR * T_{CK}$$

$$T_{ow} = 16 * CCR * T_{CK}$$

For instance: in standard mode, to generate a 100kHz SCL frequency:

If FREQR = 08, $T_{ck} = 125\text{ns}$ so CCR must be programmed with 28h(28h \leftrightarrow 40d \times 125ns = 5000 ns.)**Notes:**

1. The minimum allowed value is 04h, except in FAST DUTY mode where the minimum allowed value is 01h
2. t_{high} includes the SCLH rising edge
3. t_{low} includes the SCLH falling edge
4. These timings are without filters.

15.7.9 TRISE Register (I2C_TRISE)

Address offset: 0x20

Reset value: 0x0002

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

Bits 15:6 Reserved, forced by hardware to 0.

Bits 5:0 **TRISE[5:0]**: Maximum Rise Time in Fast/Standard mode (Master mode)

These bits must be programmed with the maximum SCL rise time given in the I²C bus specification, incremented by 1.

For instance: in standard mode, the maximum allowed SCL rise time is 1000 ns.

If, in the I2C_CR2 register, the value of FREQ[5:0] bits is equal to 08h and t_{CK} = 125 ns therefore the TRISE[5:0] bits must be programmed with 09h.

(1000 ns / 125 ns = 8 + 1)

The filter value can also be added to TRISE[5:0].

If the result is not an integer, TRISE[5:0] must be programmed with the integer part, in order to respect the t_{HIGH} parameter.

Note:

TRISE[5:0] must be configured only when the I2C is disabled (PE = 0).

15.8 I²C register map

Table 48. I²C register map and reset values

Refer to [Table 1 on page 27](#) for the register boundary addresses.

16 Analog-to-digital converter (ADC)

16.1 Introduction

The 12-bit ADC is a successive approximation analog-to-digital converter. It has up to 18 multiplexed channels allowing it measure signals from 16 external and two internal sources. A/D conversion of the various channels can be performed in single, continuous, scan or discontinuous mode. The result of the ADC is stored in a left-aligned or right-aligned 16-bit data register.

The analog watchdog feature allows the application to detect if the input voltage goes outside the user-defined high or low thresholds.

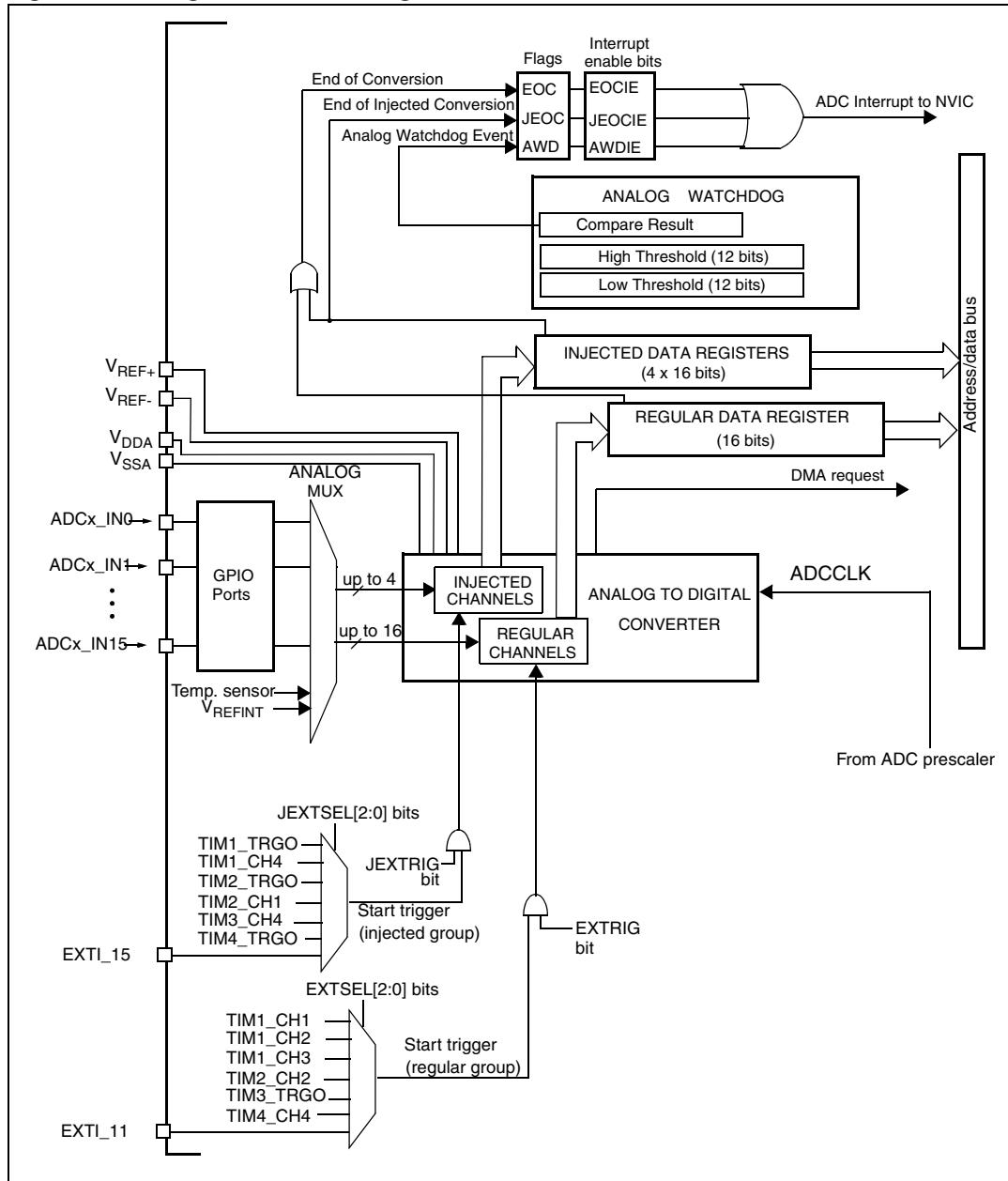
16.2 Main features

- 12-bit resolution
- Interrupt generation at End of Conversion, End of Injected conversion and Analog Watchdog event
- Single and continuous conversion modes
- Scan mode for automatic conversion of channel 0 to channel ‘n’
- Self-calibration
- Data alignment with in-built data coherency
- Channel by channel programmable sampling time
- External trigger option for both regular and injected conversion
- Discontinuous mode
- Dual mode (on devices with 2 ADCs)
- ADC conversion time:
 - STM32F103xx performance line devices: 1 μ s at 56 MHz (1.17 μ s at 72 MHz)
 - STM32F101xx access line devices: 1 μ s at 28 MHz (1.55 μ s at 36 MHz)
- ADC supply requirement: 2.4 V to 3.6 V
- ADC input range: $V_{REF-} \leq V_{IN} \leq V_{REF+}$
- DMA request generation during regular channel conversion

The block diagram of the ADC is shown in [Figure 141](#).

Note: V_{REF-} , if available (depending on package), must be tied to V_{SSA} .

Figure 141. Single ADC block diagram



16.3 Pin description

Table 49. ADC pins

Name	Signal type	Remarks
V_{REF+}	Input, analog reference positive	The higher/positive reference voltage for the ADC, $2.4 \text{ V} \leq V_{REF+} \leq V_{DDA}$
V_{DDA}	Input, analog supply	Analog power supply equal to V_{DD} and $2.4 \text{ V} \leq V_{DDA} \leq V_{DD}$ (3.6 V)
V_{REF-}	Input, analog reference negative	The lower/negative reference voltage for the ADC, $V_{REF-} = V_{SSA}$
V_{SSA}	Input, analog supply ground	Ground for analog power supply equal to V_{SS}
ADCx_IN[15:0]	Analog input signals	16 analog input channels

16.4 Functional description

16.4.1 ADC on-off control

The ADC can be powered-on by setting the ADON bit in the ADC_CR1 register. When the ADON bit is set for the first time, it wakes up the ADC from Power Down mode.

Conversion starts when ADON bit is set for a second time by software after ADC power-up time (t_{STAB}).

You can stop conversion and put the ADC in power down mode by resetting the ADON bit. In this mode the ADC consumes almost no power (only a few μA).

16.4.2 ADC clock

The ADCCLK clock provided by the Clock Controller is synchronous with the PCLK2 (APB2 clock). The CLK controller provides has a dedicated programmable prescaler for the ADC clock, refer to the CLK chapter for more details.

16.4.3 Channel selection

There are 16 multiplexed channels. It is possible to organize the conversions in two groups: regular and injected. A group consists of a sequence of conversions which can be done on any channel and in any order. For instance, it is possible to do the conversion in the following order: Ch3, Ch8, Ch2, Ch2, Ch0, Ch2, Ch2, Ch15.

- The **regular group** is composed of up to 16 conversions. The regular channels and their order in the conversion sequence must be selected in the ADC_SQRx registers. The total number of conversions in the regular group must be written in the L[3:0] bits in the ADC_SQR1 register.
- The **injected group** is composed of up to 4 conversions. The injected channels and their order in the conversion sequence must be selected in the ADC_JSQR register. The total number of conversions in the injected group must be written in the L[1:0] bits in the ADC_JSQR register.

If the ADC_SQRx or ADC_JSQR registers are modified during a conversion, the current conversion is reset and a new start pulse is sent to the ADC to convert the new chosen group.

Temperature sensor/V_{REFINT} internal channels

The Temperature sensor is connected to channel ADCx_IN16 and the internal reference voltage V_{REFINT} is connected to ADCx_IN17. These two internal channels can be selected and converted as injected or regular channels.

Note: The sensor and V_{REFINT} are only available on the master ADC1 peripheral.

16.4.4 Single conversion mode

In Single conversion mode the ADC does one conversion. This mode is started either by setting the ADON bit in the ADC_CR2 register (for a regular channel only) or by external trigger (for a regular or injected channel), while the CONT bit is 0.

Once the conversion of the selected channel is complete:

- If a regular channel was converted:
 - The converted data is stored in the 16-bit ADC_DR register
 - The EOC (End Of Conversion) flag is set
 - and an interrupt is generated if the EOCIE is set.
- If an injected channel was converted:
 - The converted data is stored in the 16-bit ADC_DRJ1 register
 - The JEOC (End Of Conversion Injected) flag is set
 - and an interrupt is generated if the JEOCIE bit is set.

The ADC is then stopped.

16.4.5 Continuous conversion mode

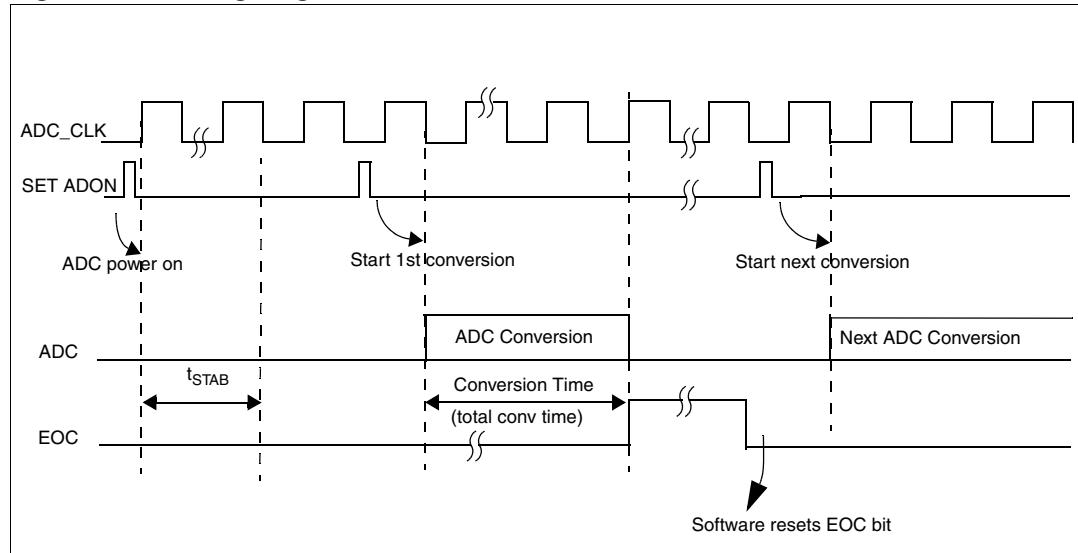
In continuous conversion mode ADC starts another conversion as soon as it finishes one. This mode is started either by external trigger or by setting the ADON bit in the ADC_CR2 register, while the CONT bit is 1.

After each conversion:

- If a regular channel was converted:
 - The converted data is stored in the 16-bit ADC_DR register
 - The EOC (End Of Conversion) flag is set
 - An interrupt is generated if the EOCIE is set.
- If an injected channel was converted:
 - The converted data is stored in the 16-bit ADC_DRJ1 register
 - The JEOC (End Of Conversion Injected) flag is set
 - An interrupt is generated if the JEOCIE bit is set.

16.4.6 Timing diagram

As shown in [Figure 142](#), the ADC needs a stabilization time of t_{STAB} before it starts converting accurately. After the start of ADC conversion and after 14 clock cycles, the EOC flag is set and the 16-bit ADC Data register contains the result of the conversion.

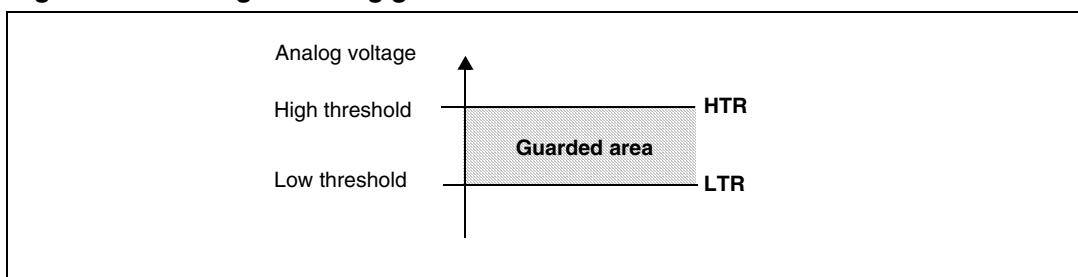
Figure 142. Timing diagram

16.4.7 Analog watchdog

The AWD analog watchdog status bit is set if the analog voltage converted by the ADC is below a low threshold or above a high threshold. These thresholds are programmed in the 12 least significant bits of the ADC_HTR and ADC_LTR 16-bit registers. An interrupt can be enabled by using the AWDIE bit in the ADC_CR1 register.

The threshold value is independent of the alignment selected by the ALIGN bit in the ADC_CR2 register. The comparison is done before the alignment (see [Section 16.6](#)).

The analog watchdog can be enabled on one or more channels by configuring the ADC_CR1 register as shown in [Table 50](#).

Figure 143. Analog watchdog guarded area**Table 50. Analog watchdog channel selection**

Channels to be guarded by Analog Watchdog	ADC_CR1 register control bits (x = don't care)		
	AWDSGL bit	AWDEN bit	JAWDEN bit
None	x	0	0
All injected channels	0	0	1
All regular channels	0	1	0
All regular and injected channels	0	1	1

Table 50. Analog watchdog channel selection (continued)

Channels to be guarded by Analog Watchdog	ADC_CR1 register control bits (x = don't care)		
	AWDSGL bit	AWDEN bit	JAWDEN bit
Single ⁽¹⁾ injected channel	1	0	1
Single ⁽¹⁾ regular channel	1	1	0
Single ⁽¹⁾ regular or injected channel	1	1	1

1. Selected by AWDCH[4:0] bits

16.4.8 Scan mode

This mode is used to scan a group of analog channels.

Scan mode can be selected by setting the SCAN bit in the ADC_CR1 register. Once this bit is set, ADC scans all the channels selected in the ADC_SQRx registers (for regular channels) or in the ADC_JSQR (for injected channels). A single conversion is performed for each channel of the group. After each end of conversion the next channel of the group is converted automatically. If the CONT bit is set, conversion does not stop at the last selected group channel but continues again from the first selected group channel.

If the DMA bit is set, the direct memory access controller is used to transfer the converted data of regular group channels to SRAM after each EOC.

The injected channel converted data is always stored in the ADC_JDRx registers.

16.4.9 Injected channel management

Triggered injection

To use triggered injection, the JAUTO bit must be cleared and SCAN bit must be set in the ADC_CR1 register.

1. Start conversion of a group of regular channels either by external trigger or by setting the ADON bit in the ADC_CR2 register.
2. If an external injected trigger occurs during the regular group channel conversion, the current conversion is reset and the injected channel sequence is converted in Scan once mode.
3. Then, the regular group channel conversion is resumed from the last interrupted regular conversion. If a regular event occurs during an injected conversion, it doesn't interrupt it but the regular sequence is executed at the end of the injected sequence.

Figure 144 shows the timing diagram.

Note: When using triggered injection, one must ensure that the interval between trigger events is longer than the injection sequence. For instance, if the sequence length is 28 ADC clock cycles (that is two conversions with a 1.5 clock-period sampling time), the minimum interval between triggers must be 29 ADC clock cycles.

Auto-injection

If the JAUTO bit is set, then the injected group channels are automatically converted after the regular group channels. This can be used to convert a sequence of up to 20 conversions programmed in the ADC_SQRx and ADC_JSQR registers.

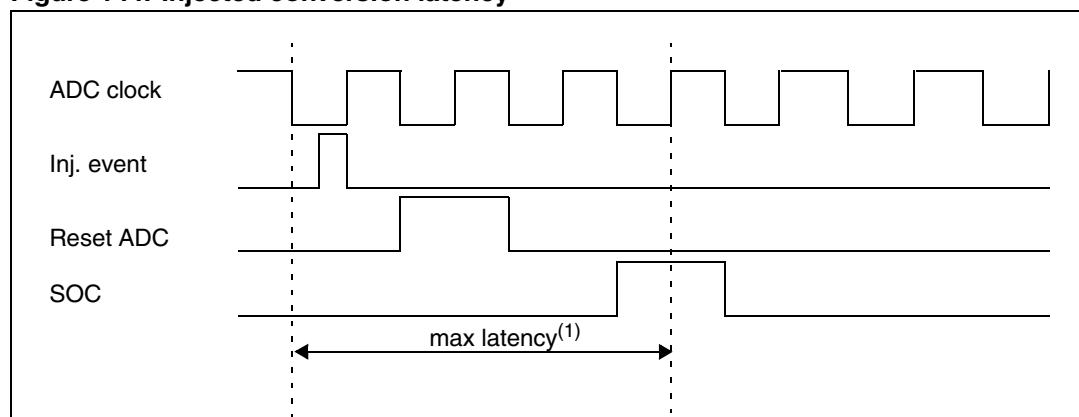
In this mode, external trigger on injected channels must be disabled.

If the CONT bit is also set in addition to the JAUTO bit, regular channels followed by injected channels are continuously converted.

For ADC clock prescalers ranging from 4 to 8, a delay of 1 ADC clock period is automatically inserted when switching from regular to injected sequence (respectively injected to regular). When the ADC clock prescaler is set to 2, the delay is 2 ADC clock periods.

Note: It is not possible to use both auto-injected and discontinuous modes simultaneously.

Figure 144. Injected conversion latency



1. The maximum latency value can be found in the electrical characteristics of the STM32F101xx and STM32F103xx datasheets.

16.4.10 Discontinuous mode

Regular group

This mode is enabled by setting the DISCEN bit in the ADC_CR1 register. It can be used to convert a short sequence of n conversions ($n \leq 8$) which is a part of the sequence of conversions selected in the ADC_SQRx registers. The value of n is specified by writing to the DISCNUM[2:0] bits in the ADC_CR1 register.

When an external trigger occurs, it starts the next n conversions selected in the ADC_SQRx registers until all the conversions in the sequence are done. The total sequence length is defined by the L[3:0] bits in the ADC_SQR1 register.

Example:

n = 3, channels to be converted = 0, 1, 2, 3, 6, 7, 9, 10
 1st trigger: sequence converted 0, 1, 2
 2nd trigger: sequence converted 3, 6, 7
 3rd trigger: sequence converted 9, 10 and an EOC event generated
 4th trigger: sequence converted 0, 1, 2

Note: When a regular group is converted in discontinuous mode, no rollover will occur.

When all sub groups are converted, the next trigger starts conversion of the first sub-group. In the example above, the 4th trigger reconverts the 1st sub-group channels 0, 1 and 2.

Injected group

This mode is enabled by setting the JDISCEN bit in the ADC_CR1 register. It can be used to convert the sequence selected in the ADC_JSQR register, channel by channel, after an external trigger event.

When an external trigger occurs, it starts the next channel conversions selected in the ADC_JSQR registers until all the conversions in the sequence are done. The total sequence length is defined by the JL[1:0] bits in the ADC_JSQR register.

Example:

n = 1, channels to be converted = 1, 2, 3
 1st trigger: channel 1 converted
 2nd trigger: channel 2 converted
 3rd trigger: channel 3 converted and EOC and JEOC events generated
 4th trigger: channel 1

- Note:*
- 1 When all injected channels are converted, the next trigger starts the conversion of the first injected channel. In the example above, the 4th trigger reconverts the 1st injected channel 1.
 - 2 It is not possible to use both auto-injected and discontinuous modes simultaneously.
 - 3 The user must avoid setting discontinuous mode for both regular and injected groups together. Discontinuous mode must be enabled only for one group conversion.

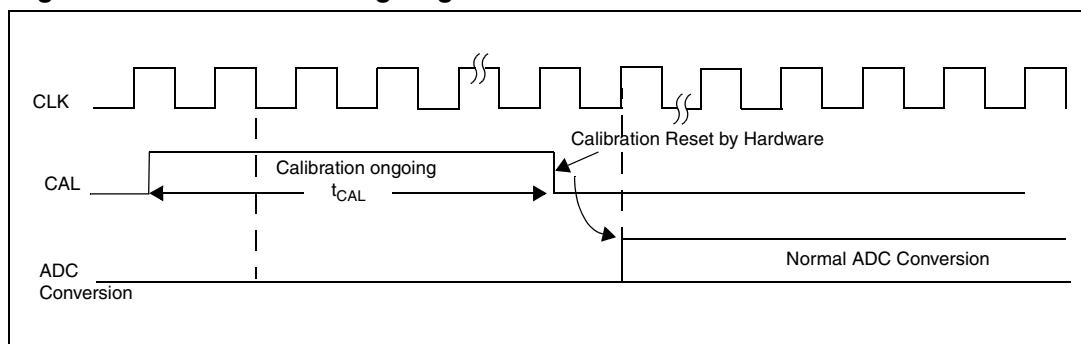
16.5 Calibration

The ADC has an built-in self calibration mode. Calibration significantly reduces accuracy errors due to internal capacitor bank variations. During calibration, an error-correction code (digital word) is calculated for each capacitor, and during all subsequent conversions, the error contribution of each capacitor is removed using this code.

Calibration is started by setting the CAL bit in the ADC_CR2 register. Once calibration is over, the CAL bit is reset by hardware and normal conversion can be performed. It is recommended to calibrate the ADC once at power-on. The calibration codes are stored in the ADC_DR as soon as the calibration phase ends.

- Note:*
- 1 *It is recommended to perform a calibration after each power-up.*
 - 2 *Before starting a calibration the ADC must have been in power-off state (ADON bit = '0') for at least two ADC clock cycles.*

Figure 145. Calibration timing diagram



16.6 Data alignment

ALIGN bit in the ADC_CR2 register selects the alignment of data stored after conversion. Data can be left or right aligned as shown in [Figure 146](#). and [Figure 147](#).

The injected group channels converted data value is decreased by the user-defined offset written in the ADC_JOFRx registers so the result can be a negative value. The SEXT bit is the extended sign value.

For regular group channels no offset is subtracted so only twelve bits are significant.

Figure 146. Right alignment of data

Injected group															
SEXT	SEXT	SEXT	SEXT	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Regular group															
0	0	0	0	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

Figure 147. Left alignment of data

Injected group															
SEXT	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0
Regular group															
D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0	0

16.7 Channel-by-channel programmable sample time

ADC samples the input voltage for a number of ADC_CLK cycles which can be modified using the SMP[2:0] bits in the ADC_SMPR1 and ADC_SMPR2 registers. Each channel can be sampled with a different sample time.

The total conversion time is calculated as follows:

$$T_{conv} = \text{Sampling time} + 12.5 \text{ cycles}$$

Example:

With an ADCCLK = 14 MHz and a sampling time of 1.5 cycles:

$$T_{conv} = 1.5 + 12.5 = 14 \text{ cycles} = 1\mu\text{s}$$

16.8 Conversion on external trigger

Conversion can be triggered by an external event (e.g. timer capture, EXTI line). If the EXT-TRIG control bit is set then external events are able to trigger a conversion. The EXT-SEL[2:0] and JEXTSEL[2:0] control bits allow the application to select decide which out of 8 possible events can trigger conversion for the regular and injected groups.

Note:

When an external trigger is selected for ADC regular or injected conversion, only the rising edge of the signal can start the conversion.

Table 51. External trigger for regular channels

Source	Type	EXTSEL[2:0]
TIM1_CC1 event	Internal signal from on-chip timers	000
TIM1_CC2 event		001
TIM1_CC3 event		010
TIM2_CC2 event		011
TIM3_TRGO event		100
TIM4_CC4 event		101
EXTI line11	External pin	110
SWSTART	Software control bit	111

Table 52. External trigger for injected channels

Source	Connection type	JEXTSEL[2:0]
TIM1_TRGO event	Internal signal from on-chip timers	000
TIM1_CC4 event		001
TIM2_TRGO event		010
TIM2_CC1 event		011
TIM3_CC4 event		100
TIM4_TRGO event		101
EXTI line15	External pin	110
JSWSTART	Software control bit	111

The software source trigger events can be generated by setting a bit in a register (SWSTART and JSWSTART in ADC_CR2).

A regular group conversion can be interrupted by an injected trigger.

16.9 DMA request

Since converted regular channels value are stored in a unique data register, it is necessary to use DMA for conversion of more than one regular channel. This avoids the loss of data already stored in the ADC_DR register.

Only the end of conversion of a regular channel generates a DMA request, which allows the transfer of its converted data from the ADC_DR register to the destination location selected by the user.

Note: Only ADC1 has a DMA capability.

16.10 Dual ADC mode

In devices with two ADCs, dual ADC mode can be used (see [Figure 148](#)).

In dual ADC mode the start of conversion is triggered alternately or simultaneously by the ADC1 master to the ADC2 slave, depending on the mode selected by the DUALMOD[2:0] bits in the ADC1_CR1 register.

Note: *In dual mode, when configuring conversion to be triggered by an external event, the user must set the trigger for the master only and set a software trigger for the slave to prevent spurious triggers to start unwanted slave conversion. However, external triggers must be enabled on both master and slave ADCs.*

The following six possible modes are implemented:

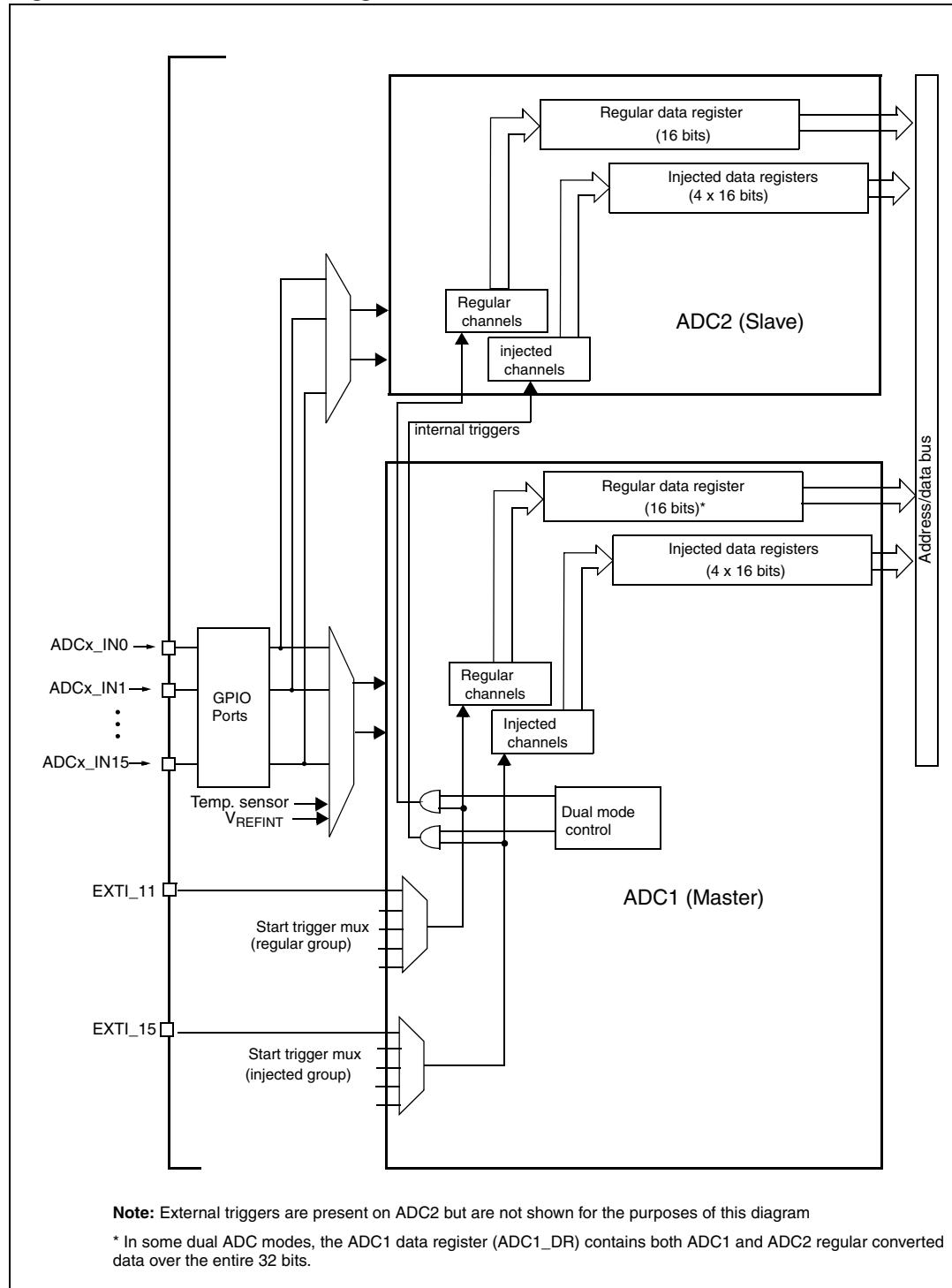
- Injected simultaneous mode
- Regular simultaneous mode
- Fast interleaved mode
- Slow interleaved mode
- Alternate trigger mode
- Independent mode

It is also possible to use the previous modes combined in the following ways:

- Injected simultaneous mode + Regular simultaneous mode
- Regular simultaneous mode + Alternate trigger mode
- Injected simultaneous mode + Interleaved mode

Note: *In dual ADC mode, to read the slave converted data on the master data register, the DMA bit must be enabled even if it is not used to transfer converted regular channel data.*

Figure 148. Dual ADC block diagram



16.10.1 Injected simultaneous mode

This mode converts an injected channel group. The source of external trigger comes from the injected group mux of ADC1 (selected by the JEXTSEL[2:0] bits in the ADC1_CR2 register). A simultaneous trigger is provided to ADC2.

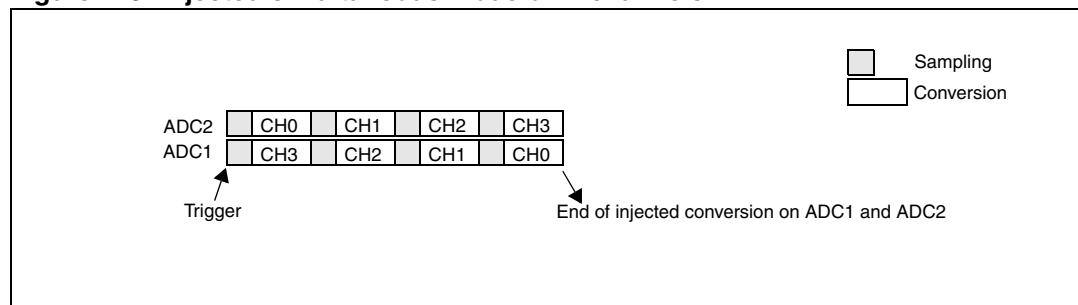
Note: *Do not convert the same channel on the two ADCs (no overlapping sampling times for the two ADCs when converting the same channel).*

At the end of conversion event on ADC1 or ADC2:

- The converted data is stored in the ADC_JDRx registers of each ADC interface.
- An JEOC interrupt is generated (if enabled on one of the two ADC interfaces) when the ADC1/ADC2 injected channels are all converted.

Note: *In simultaneous mode, one must convert sequences with the same length or ensure that the interval between triggers is longer than the longest of the 2 sequences. Otherwise, the ADC with the shortest sequence may restart while the ADC with the longest sequence is completing the previous conversions.*

Figure 149. Injected simultaneous mode on 4 channels



16.10.2 Regular simultaneous mode

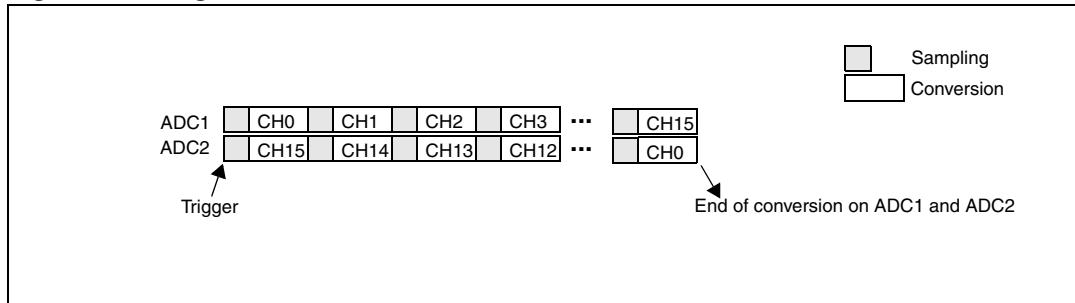
This mode is performed on a regular channel group. The source of the external trigger comes from the regular group mux of ADC1 (selected by the EXTSEL[2:0] bits in the ADC1_CR2 register). A simultaneous trigger is provided to the ADC2.

Note: *Do not convert the same channel on the two ADCs (no overlapping sampling times for the two ADCs when converting the same channel).*

At the end of conversion event on ADC1 or ADC2:

- A 32-bit DMA transfer request is generated (if DMA bit is set) which transfers to SRAM the ADC1_DR 32-bit register containing the ADC2 converted data in the upper halfword and the ADC1 converted data in the lower halfword.
- An EOC interrupt is generated (if enabled on one of the two ADC interfaces) when ADC1/ADC2 regular channels are all converted.

Note: *In regular simultaneous mode, one must convert sequences with the same length or ensure that the interval between triggers is longer than the longest of the 2 sequences. Otherwise, the ADC with the shortest sequence may restart while the ADC with the longest sequence is completing the previous conversions.*

Figure 150. Regular simultaneous mode on 16 channels

16.10.3 Fast interleaved mode

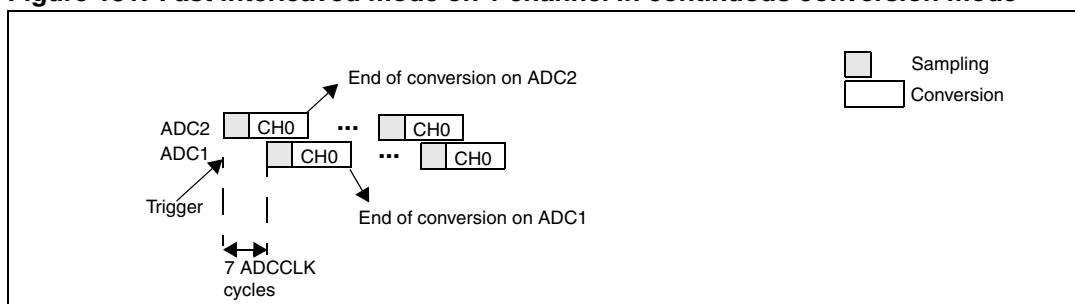
This mode can be started only on a regular channel group (usually one channel). The source of external trigger comes from the regular channel mux of ADC1. After an external trigger occurs:

- ADC2 starts immediately and
- ADC1 starts after a delay of 7 ADC clock cycles.

If CONT bit is set on both ADC1 and ADC2 the selected regular channels of both ADCs are continuously converted.

After an EOC interrupt is generated by ADC1 (if enabled through the EOCIE bit) a 32-bit DMA transfer request is generated (if the DMA bit is set) which transfers to SRAM the ADC1_DR 32-bit register containing the ADC2 converted data in the upper halfword and the ADC1 converted data in the lower halfword.

Note: *The maximum sampling time allowed is <7 ADCCLK cycles to avoid the overlap between ADC1 and ADC2 sampling phases in the event that they convert the same channel.*

Figure 151. Fast interleaved mode on 1 channel in continuous conversion mode

16.10.4 Slow interleaved mode

This mode can be started only on a regular channel group (only one channel). The source of external trigger comes from regular channel mux of ADC1. After external trigger occurs:

- ADC2 starts immediately and
- ADC1 starts after a delay of 14 ADC clock cycles.
- ADC2 starts after a second delay of 14 ADC cycles, and so on.

Note: *The maximum sampling time allowed is <14 ADCCLK cycles to avoid an overlap with the next conversion.*

After an EOC interrupt is generated by ADC1 (if enabled through the EOCIE bit) a 32-bit DMA transfer request is generated (if the DMA bit is set) which transfers to SRAM the ADC1_DR 32-bit register containing the ADC2 converted data in the upper halfword and the ADC1 converted data in the lower halfword.

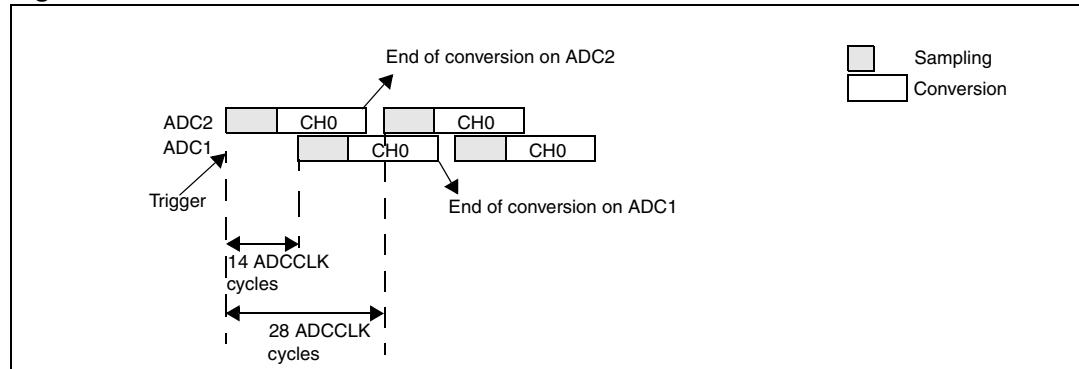
A new ADC2 start is automatically generated after 28 ADC clock cycles

CONT bit can not be set in the mode since it continuously converts the selected regular channel.

Note:

The application must ensure that no external trigger for injected channel occurs when interleaved mode is enabled.

Figure 152. Slow interleaved mode on 1 channel



16.10.5 Alternate trigger mode

This mode can be started only on an injected channel group. The source of external trigger comes from the injected group mux of ADC1.

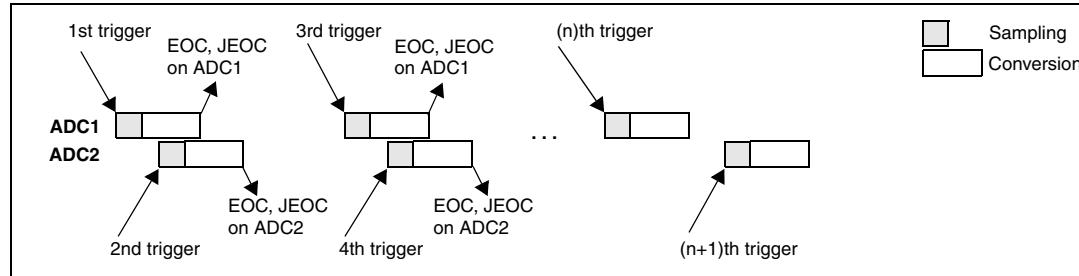
- When the 1st trigger occurs, all injected group channels in ADC1 are converted.
- When the 2nd trigger arrives, all injected group channels in ADC2 are converted
- and so on....

A JEOC interrupt, if enabled, is generated after all injected group channels of ADC1 are converted.

A JEOC interrupt, if enabled, is generated after all injected group channels of ADC2 are converted.

If another external trigger occurs after all injected group channels have been converted then the alternate trigger process restarts by converting ADC1 injected group channels.

Figure 153. Alternate trigger: injected channel group of each ADC



If the injected discontinuous mode is enabled for both ADC1 and ADC2:

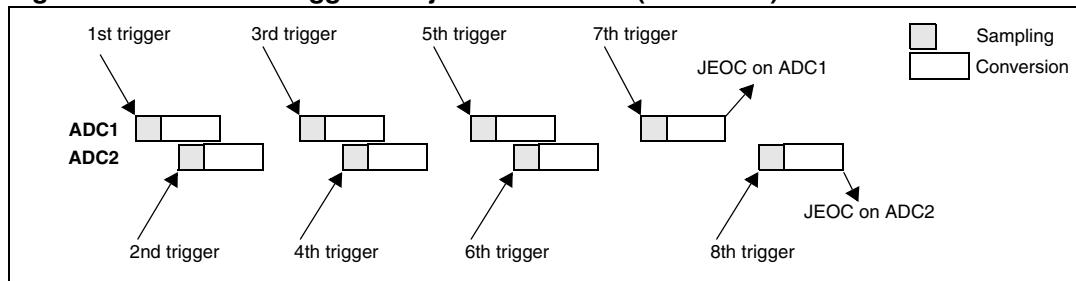
- When the 1st trigger occurs, the first injected channel in ADC1 is converted.
- When the 2nd trigger arrives, the first injected channel in ADC2 are converted
- and so on....

A JEOC interrupt, if enabled, is generated after all injected group channels of ADC1 are converted.

A JEOC interrupt, if enabled, is generated after all injected group channels of ADC2 are converted.

If another external trigger occurs after all injected group channels have been converted then the alternate trigger process restarts.

Figure 154. Alternate trigger: 4 injected channels (each ADC) in discontinuous mode



16.10.6 Independent mode

In this mode the dual ADC synchronization is bypassed and each ADC interfaces works independently.

16.10.7 Combined regular/injected simultaneous mode

It is possible to interrupt simultaneous conversion of a regular group to start simultaneous conversion of an injected group.

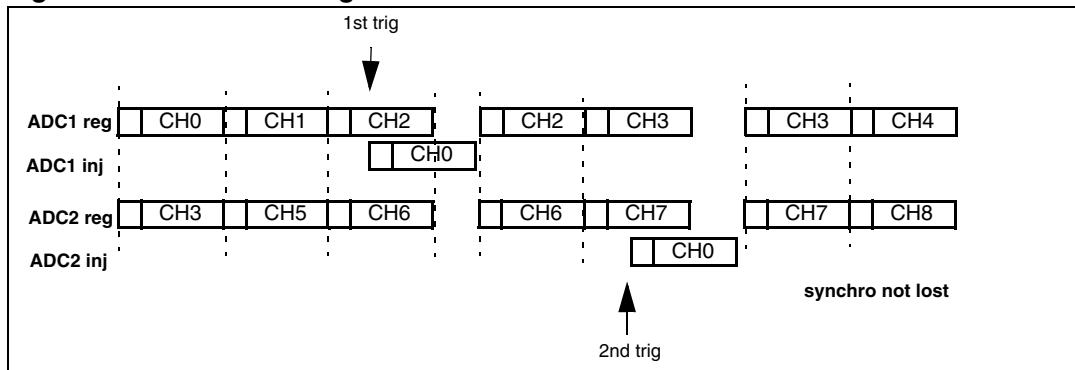
Note: *In combined regular/injected simultaneous mode, one must convert sequences with the same length or ensure that the interval between triggers is longer than the longest of the 2 sequences. Otherwise, the ADC with the shortest sequence may restart while the ADC with the longest sequence is completing the previous conversions.*

16.10.8 Combined regular simultaneous + alternate trigger mode

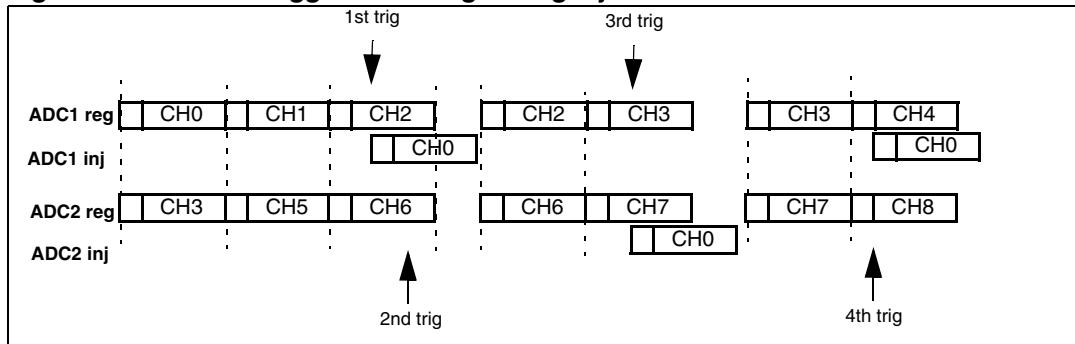
It is possible to interrupt regular group simultaneous conversion to start alternate trigger conversion of an injected group. [Figure 155](#) shows the behavior of an alternate trigger interrupting a regular simultaneous conversion.

The injected alternate conversion is immediately started after the injected event arrives. If regular conversion is already running, in order to ensure synchronization after the injected conversion, the regular conversion of both (master/slave) ADCs is stopped and resumed synchronously at the end of the injected conversion.

Note: *In combined regular simultaneous + alternate trigger mode, one must convert sequences with the same length or ensure that the interval between triggers is longer than the longest of the 2 sequences. Otherwise, the ADC with the shortest sequence may restart while the ADC with the longest sequence is completing the previous conversions.*

Figure 155. Alternate + Regular simultaneous

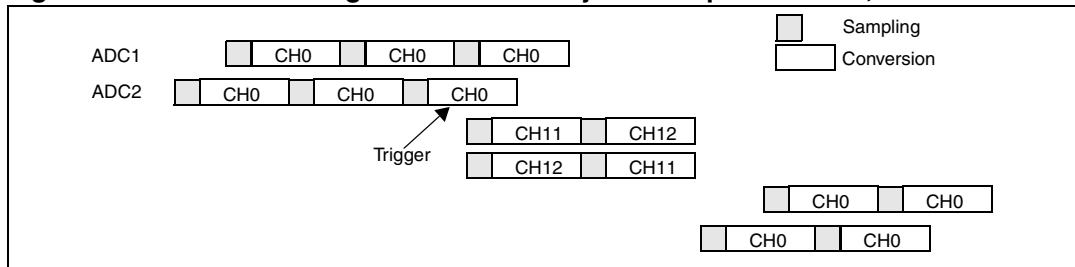
If a trigger occurs during an injected conversion that has interrupted a regular conversion, it will be ignored. [Figure 156](#) shows the behavior in this case (2nd trig is ignored).

Figure 156. Case of trigger occurring during injected conversion

16.10.9 Combined injected simultaneous + interleaved

It is possible to interrupt an interleaved conversion with an injected event. In this case the interleaved conversion is interrupted and the injected conversion starts, at the end of the injected sequence the interleaved conversion is resumed. [Figure 157](#) shows the behavior using an example.

Note: When the ADC clock prescaler is set to 4, the interleaved mode does not recover with evenly spaced sampling periods: the sampling interval is 8 ADC clock periods followed by 6 ADC clock periods, instead of 7 clock periods followed by 7 clock periods.

Figure 157. Interleaved single channel with injected sequence CH11, CH12

16.11 Temperature sensor

The temperature sensor can be used to measure the ambient temperature (T_A) of the device.

The temperature sensor is internally connected to the ADCx_IN16 input channel which is used to convert the sensor output voltage into a digital value. The sampling time for temperature sensor analog input must be greater than 2.2 μ s.

The block diagram of the temperature sensor is shown in [Figure 158](#).

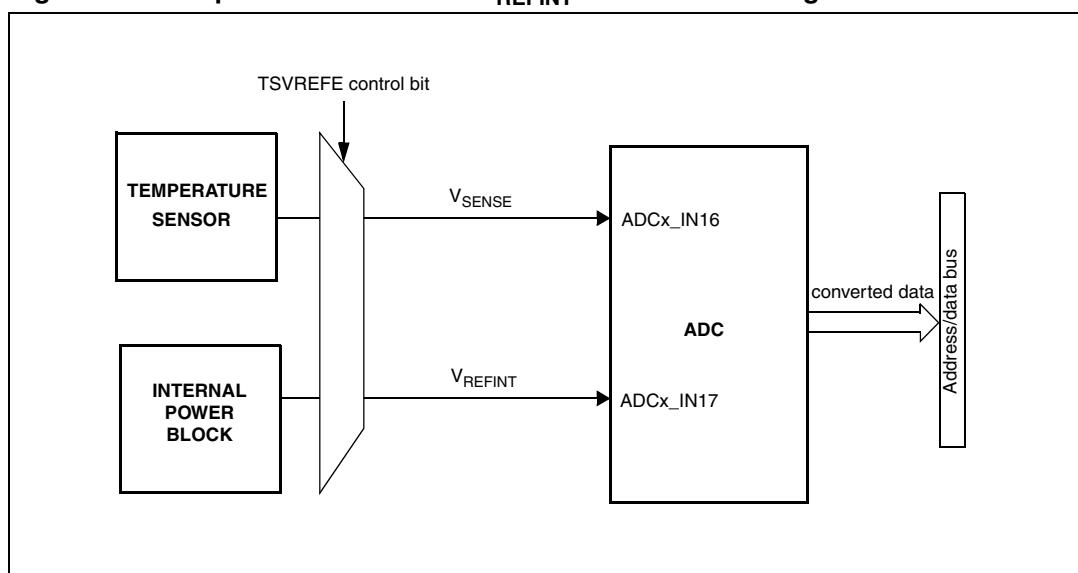
When not in use, this sensor can be put in power down mode.

Note: The TSVREFE bit must be set to enable both internal channels: ADCx_IN16 (temperature sensor) and ADCx_IN17 (V_{REFINT}) conversion.

Main features

- Supported Temperature Range: -40 to 125 degrees
- Precision: $\pm 1.5^\circ\text{C}$

Figure 158. Temperature sensor and V_{REFINT} channel block diagram



Reading the temperature

To use the sensor:

4. Select the ADCx_IN16 input channel.
5. Select a sample time greater than 2.2 μ s
6. Set the TSVREFE bit in the *ADC control register 2 (ADC_CR2)* to wake up the temperature sensor from power down mode.
7. Start the ADC conversion by setting the ADON bit (or by external trigger).
8. Read the resulting V_{SENSE} data in the ADC data register
9. Obtain the temperature using the following formula:

$$\text{Temperature (in } ^\circ\text{C)} = \{(V_{25} - V_{SENSE}) / \text{Avg_Slope}\} + 25.$$

Where,

V_{25} = V_{SENSE} value for 25° C and

Avg_Slope = Average Slope for curve between Temperature vs. V_{SENSE} (given in mV/ $^\circ$ C or μ V/ $^\circ$ C).

Refer to the Electrical characteristics section for the actual values of V_{25} and Avg_Slope.

Note:

The sensor has a startup time after waking from power down mode before it can output V_{SENSE} at the correct level. The ADC also has a startup time after power-on, so to minimize the delay, the ADON and TSVREFE bits should be set at the same time.

16.12 Interrupts

An interrupt can be produced on end of conversion for regular and injected groups and when the Analog Watchdog status bit is set. Separate interrupt enable bits are available for flexibility.

Two other flags are present in the ADC_SR register, but there is no interrupt associated with them:

- JSTART (Start of conversion for injected group channels)
- START (Start of conversion for regular group channels)

Table 53. ADC interrupts

Interrupt event	Event flag	Enable Control bit
End of Conversion regular group	EOC	EOCIE
End of Conversion injected group	JEOC	JEOCIE
Analog Watchdog Status bit is set	AWD	AWDIE

16.13 ADC register description

Refer to [Section 1.1 on page 23](#) for a list of abbreviations used in register descriptions.

16.13.1 ADC status register (ADC_SR)

Address offset: 0x00

Reset value: 0x0000 0000

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

Bits 31:5 Reserved, must be kept cleared.

Bit 4 STRT: Regular channel Start flag

This bit is set by hardware when regular channel conversion starts. It is cleared by software.

0: No regular channel conversion started

1: Regular channel conversion has started

Bit 3 JSTRT: Injected channel Start flag

This bit is set by hardware when injected channel group conversion starts. It is cleared by software.

0: No injected group conversion started

1: Injected group conversion has started

Bit 2 JEOC: Injected channel end of conversion

This bit is set by hardware at the end of all injected group channel conversion. It is cleared by software.

0: Conversion is not complete

1: Conversion complete

Bit 1 EOC: End of conversion

This bit is set by hardware at the end of a group channel conversion (regular or injected). It is cleared by software or by reading the ADC_DR.

0: Conversion is not complete

1: Conversion complete

Bit 0 AWD: Analog watchdog flag

This bit is set by hardware when the converted voltage crosses the values programmed in the ADC_LTR and ADC_HTR registers. It is cleared by software.

0: No Analog Watchdog event occurred

1: Analog Watchdog event occurred

16.13.2 ADC control register 1 (ADC_CR1)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved							AWDEN	JAWDEN	Reserved		DUALMOD[3:0]				
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
DISCNUM[2:0]		JDISCEN	DISCEN	JAUTO	AWDSGL	SCAN	JEOCIE	AWDIE	EOCIE	AWDCH[4:0]					
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 Reserved, must be kept cleared.

Bit 23 **AWDEN: Analog watchdog enable on regular channels**

This bit is set/reset by software.

0: Analog watchdog disabled on regular channels

1: Analog watchdog enabled on regular channels

Bit 22 **JAWDEN: Analog watchdog enable on injected channels**

This bit is set/reset by software.

0: Analog watchdog disabled on injected channels

1: Analog watchdog enabled on injected channels

Bits 21:20 Reserved, must be kept cleared.

Bits 19:16 **DUALMOD[3:0]: Dual mode selection**

These bits are written by software to select the operating mode.

0000: Independent mode.

0001: Combined regular simultaneous + injected simultaneous mode

0010: Combined regular simultaneous + alternate trigger mode

0011: Combined injected simultaneous + fast interleaved mode

0100: Combined injected simultaneous + slow Interleaved mode

0101: Injected simultaneous mode only

0110: Regular simultaneous mode only

0111: Fast interleaved mode only

1000: Slow interleaved mode only

1001: Alternate trigger mode only

Notes:

- These bits are reserved in ADC2.

- In dual mode, a change of channel configuration generates a restart that can produce a loss of synchronization. It is recommended to disable dual mode before any configuration change.

Bits 15:13 **DISCNUM[2:0]: Discontinuous mode channel count**

These bits are written by software to define the number of regular channels to be converted in discontinuous mode, after receiving an external trigger.

000: 1 channel

001: 2 channels

.....

111: 8 channels

Bit 12 JDISCEN: *Discontinuous mode on injected channels*

This bit set and cleared by software to enable/disable discontinuous mode on injected group channels

- 0: Discontinuous mode on injected channels disabled
- 1: Discontinuous mode on injected channels enabled

Bit 11 DISCEN: *Discontinuous mode on regular channels*

This bit set and cleared by software to enable/disable Discontinuous mode on regular channels.

- 0: Discontinuous mode on regular channels disabled
- 1: Discontinuous mode on regular channels enabled

Bit 10 JAUTO: *Automatic Injected Group conversion*

This bit set and cleared by software to enable/disable automatic injected group conversion after regular group conversion.

- 0: Automatic injected group conversion disabled
- 1: Automatic injected group conversion enabled

Bit 9 AWDSGL: *Enable the watchdog on a single channel in scan mode*

This bit set and cleared by software to enable/disable the analog watchdog on the channel identified by the AWDCH[4:0] bits.

- 0: Analog watchdog enabled on all channels
- 1: Analog watchdog enabled on a single channel

Bit 8 SCAN: *Scan mode*

This bit is set and cleared by software to enable/disable Scan mode. In Scan mode, the inputs selected through the ADC_SQRx or ADC_JSQRx registers are converted.

- 0: Scan mode disabled
- 1: Scan mode enabled

Note: An EOC or JEOC interrupt is generated only on the end of conversion of the last channel if the corresponding EOcie or JEOCIE bit is set

Bit 7 JEOCIE: *Interrupt enable for injected channels*

This bit is set and cleared by software to enable/disable the end of conversion interrupt for injected channels.

- 0: JEOC interrupt disabled
- 1: JEOC interrupt enabled. An interrupt is generated when the JEOC bit is set.

Bit 6 AWDIE: *Analog Watchdog interrupt enable*

This bit is set and cleared by software to enable/disable the analog watchdog interrupt. In Scan mode if the watchdog thresholds are crossed, scan is aborted only if this bit is enabled.

- 0: Analog Watchdog interrupt disabled
- 1: Analog Watchdog interrupt enabled

Bit 5 EOcie: *Interrupt enable for EOC*

This bit is set and cleared by software to enable/disable the End of Conversion interrupt.

- 0: EOC interrupt disabled
- 1: EOC interrupt enabled. An interrupt is generated when the EOC bit is set.

Bits 4:0 **AWDCH[4:0]: Analog watchdog channel select bits**

These bits are set and cleared by software. They select the input channel to be guarded by the Analog Watchdog.

00000: ADC analog input Channel0
00001: ADC analog input Channel1

....

01111: ADC analog input Channel15
10000: ADC analog input Channel16
10001: ADC analog input Channel17
Other values reserved.

Notes:

- ADC1 analog inputs Channel16 and Channel17 are internally connected to the temperature sensor and to V_{REFINT} , respectively.
- ADC2 analog inputs Channel16 and Channel17 are internally connected to V_{SS} .

16.13.3 ADC control register 2 (ADC_CR2)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								TSVRE FE	SWST ART	JSWST ART	EXTT RIG	EXTSEL[2:0]			Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JEXTT RIG	JEXTSEL[2:0]		ALIGN	Reserved		DMA	Reserved				RST CAL	CAL	CONT	ADON	
rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 Reserved, must be kept cleared.

Bit 23 TSVREFE: Temperature Sensor and V_{REFINT} Enable

This bit is set and cleared by software to enable/disable the temperature sensor and V_{REFINT} channel. In devices with dual ADCs this bit is present only in ADC1.

0: Temperature sensor and V_{REFINT} channel disabled
1: Temperature sensor and V_{REFINT} channel enabled

Bit 22 SWSTART: Start Conversion of regular channels

This bit is set by software to start conversion and cleared by hardware as soon as conversion starts. It starts a conversion of a group of regular channels if SWSTART is selected as trigger event by the EXTSEL[2:0] bits.

0: Reset state
1: Starts conversion of regular channels

Bit 21 JSWSTART: Start Conversion of injected channels

This bit is set by software and cleared by software or by hardware as soon as the conversion starts. It starts a conversion of a group of injected channels (if JSWSTART is selected as trigger event by the JEXTSEL[2:0] bits).

0: Reset state
1: Starts conversion of injected channels

Bit 20 EXTTRIG: External Trigger Conversion mode for regular channels

This bit is set and cleared by software to enable/disable the external trigger used to start conversion of a regular channel group.

- 0: Conversion on external event disabled
- 1: Conversion on external event enabled

Bits 19:17 EXTSEL[2:0]: External Event Select for regular group

These bits select the external event used to trigger the start of conversion of a regular group:

- 000: Timer 1 CC1 event
- 001: Timer 1 CC2 event
- 010: Timer 1 CC3 event
- 011: Timer 2 CC2 event
- 100: Timer 3 TRGO event
- 101: Timer 4 CC4 event
- 110: EXTI line11
- 111: SWSTART

Bit 16 Reserved, must be kept cleared.

Bit 15 JEXTTRIG: External Trigger Conversion mode for injected channels

This bit is set and cleared by software to enable/disable the external trigger used to start conversion of an injected channel group.

- 0: Conversion on external event disabled
- 1: Conversion on external event enabled

Bits 14:12 JEXTSEL[2:0]: External event select for injected group

These bits select the external event used to trigger the start of conversion of an injected group:

- 000: Timer 1 TRGO event
- 001: Timer 1 CC4 event
- 010: Timer 2 TRGO event
- 011: Timer 2 CC1 event
- 100: Timer 3 CC4 event
- 101: Timer 4 TRGO event
- 110: EXTI line15
- 111: JSWSTART

Bit 11 ALIGN: Data Alignment

This bit is set and cleared by software. Refer to [Figure 146](#) and [Figure 147](#).

- 0: Right Alignment
- 1: Left Alignment

Bits 10:9 Reserved, must be kept cleared.

Bit 8 DMA: Direct Memory access mode

This bit is set and cleared by software. Refer to the DMA controller chapter for more details.

- 0: DMA mode disabled
- 1: DMA mode enabled

Note: In devices with more than one ADC, only ADC1 can generate a DMA request.

Bits 7:4 Reserved, must be kept cleared.

Bit 3 RSTCAL: Reset Calibration

This bit is set by software and cleared by hardware. It is cleared after the calibration registers are initialized.

- 0: Calibration register initialized.
- 1: Initialize calibration register.

Note: If RSTCAL is set when conversion is ongoing, additional cycles are required to clear the calibration registers.

Bit 2 CAL: A/D Calibration

This bit is set by software to start the calibration. It is reset by hardware after calibration is complete.

- 0: Calibration completed
- 1: Enable calibration

Bit 1 CONT: Continuous Conversion

This bit is set and cleared by software. If set conversion takes place continuously till this bit is reset.

- 0: Single conversion mode
- 1: Continuous conversion mode

Bit 0 ADON: A/D Converter ON / OFF

This bit is set and cleared by software. If this bit holds a value of zero and a 1 is written to it then it wakes up the ADC from Power Down state.

Conversion starts when this bit holds a value of 1 and a 1 is written to it. The application should allow a delay of t_{STAB} between power up and start of conversion. Refer to [Figure 142](#).

- 0: Disable ADC conversion/calibration and go to power down mode.
- 1: Enable ADC and to start conversion

Note: If any other bit in this register apart from ADON is changed at the same time, then conversion is not triggered. This is to prevent triggering an erroneous conversion.

16.13.4 ADC sample time register 1 (ADC_SMPR1)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved						SMP17[2:0]			SMP16[2:0]			SMP15[2:1]			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMP 15_0	SMP14[2:0]			SMP13[2:0]			SMP12[2:0]			SMP11[2:0]			SMP10[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 Reserved, must be kept cleared.

Bits 23:0 **SMPx[2:0]: Channel x Sample time selection**

These bits are written by software to select the sample time individually for each channel. During sample cycles channel selection bits must remain unchanged.

- 000: 1.5 cycles
- 001: 7.5 cycles
- 010: 13.5 cycles
- 011: 28.5 cycles
- 100: 41.5 cycles
- 101: 55.5 cycles
- 110: 71.5 cycles
- 111: 239.5 cycles

Notes:

- ADC1 analog inputs Channel16 and Channel17 are internally connected to the temperature sensor and to V_{REFINT} , respectively.
- ADC2 analog input Channel16 and Channel17 are internally connected to V_{SS} .

16.13.5 ADC sample time register 2 (ADC_SMPR2)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		SMP9[2:0]		SMP8[2:0]		SMP7[2:0]		SMP6[2:0]		SMP5[2:1]					
		rw	rw	rw	rw	rw	rw								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMP 5_0		SMP4[2:0]		SMP3[2:0]		SMP2[2:0]		SMP1[2:0]		SMP0[2:0]					
		rw	rw	rw	rw	rw	rw								

Bits 31:30 Reserved, must be kept cleared.

Bits 29:0 **SMPx[2:0]: Channel x Sample time selection**

These bits are written by software to select the sample time individually for each channel. During sample cycles channel selection bits must remain unchanged.

- 000: 1.5 cycles
- 001: 7.5 cycles
- 010: 13.5 cycles
- 011: 28.5 cycles
- 100: 41.5 cycles
- 101: 55.5 cycles
- 110: 71.5 cycles
- 111: 239.5 cycles

16.13.6 ADC injected channel data offset register x (ADC_JOFR x)($x=1..4$)

Address offset: 0x14-0x20

Reset value: 0x0000 0000

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

Bits 31:12 Reserved, must be kept cleared.

Bits 11:0 **JOFFSET x [11:0]**: *Data offset for injected channel x*

These bits are written by software to define the offset to be subtracted from the raw converted data when converting injected channels. The conversion result can be read from in the ADC_JDR x registers.

16.13.7 ADC watchdog high threshold register (ADC_HTR)

Address offset: 0x24

Reset value: 0x0000 0000

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

Bits 31:12 Reserved, must be kept cleared.

Bits 11:0 **HT[11:0]** *Analog watchdog high threshold*

These bits are written by software to define the high threshold for the Analog Watchdog.

16.13.8 ADC watchdog low threshold register (ADC_LTR)

Address offset: 0x28

Reset value: 0x0000 0000

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

Bits 31:12 Reserved, must be kept cleared.

Bits 11:0 **LT[11:0]: Analog watchdog low threshold**

These bits are written by software to define the low threshold for the Analog Watchdog.

16.13.9 ADC regular sequence register 1 (ADC_SQR1)

Address offset: 0x2C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								L[3:0]				SQ16[4:1]			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ16_0	SQ15[4:0]				SQ14[4:0]				SQ13[4:0]						
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 Reserved, must be kept cleared.

Bits 23:20 **L[3:0]: Regular channel sequence length**

These bits are written by software to define the total number of conversions in the regular channel conversion sequence.

0000: 1 conversion

0001: 2 conversions

.....

1111: 16 conversions

Bits 19:15 **SQ16[4:0]: 16th conversion in regular sequence**

These bits are written by software with the channel number (0..17) assigned as the 16th in the conversion sequence.

Bits 14:10 **SQ15[4:0]: 15th conversion in regular sequence**

Bits 9:5 **SQ14[4:0]: 14th conversion in regular sequence**

Bits 4:0 **SQ13[4:0]: 13th conversion in regular sequence**

16.13.10 ADC regular sequence register 2 (ADC_SQR2)

Address offset: 0x30

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		SQ12[4:0]				SQ11[4:0]				SQ10[4:1]					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ10_0		SQ9[4:0]				SQ8[4:0]				SQ7[4:0]					
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30 Reserved, must be kept cleared.

Bits 29:26 **SQ12[4:0]: 12th conversion in regular sequence**

These bits are written by software with the channel number (0..17) assigned as the 12th in the sequence to be converted.

Bits 24:20 **SQ11[4:0]: 11th conversion in regular sequence**

Bits 19:15 **SQ10[4:0]: 10th conversion in regular sequence**

Bits 14:10 **SQ9[4:0]: 9th conversion in regular sequence**

Bits 9:5 **SQ8[4:0]: 8th conversion in regular sequence**

Bits 4:0 **SQ7[4:0]: 7th conversion in regular sequence**

16.13.11 ADC regular sequence register 3 (ADC_SQR3)

Address offset: 0x34

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		SQ6[4:0]				SQ5[4:0]				SQ4[4:1]					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ4_0		SQ3[4:0]				SQ2[4:0]				SQ1[4:0]					
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30 Reserved, must be kept cleared.

Bits 29:25 **SQ6[4:0]: 6th conversion in regular sequence**

These bits are written by software with the channel number (0..17) assigned as the 6th in the sequence to be converted.

Bits 24:20 **SQ5[4:0]: 5th conversion in regular sequence**

Bits 19:15 **SQ4[4:0]: 4th conversion in regular sequence**

Bits 14:10 **SQ3[4:0]: 3rd conversion in regular sequence**

Bits 9:5 **SQ2[4:0]: 2nd conversion in regular sequence**

Bits 4:0 **SQ1[4:0]: 1st conversion in regular sequence**

16.13.12 ADC injected sequence register (ADC_JSQR)

Address offset: 0x38

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved										JL[1:0]	JSQ4[4:1]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
JSQ4[0]	JSQ3[4:0]				JSQ2[4:0]				JSQ1[4:0]						
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:22 Reserved, must be kept cleared.

Bits 21:20 **JL[1:0]: Injected Sequence length**

These bits are written by software to define the total number of conversions in the injected channel conversion sequence.

- 00: 1 conversion
- 01: 2 conversions
- 10: 3 conversions
- 11: 4 conversions

Bits 19:15 **JSQ4[4:0]: 4th conversion in injected sequence**

These bits are written by software with the channel number (0..17) assigned as the 4th in the sequence to be converted.

Note: Unlike a regular conversion sequence, if JL[1:0] length is less than four, the channels are converted in a sequence starting from (4-JL). Example: ADC_JSQR[21:0] = 10 00011 00011 00111 00010 means that a scan conversion will convert the following channel sequence: 7, 3, 3, (not 2, 7, 3)

Bits 14:10 **JSQ3[4:0]: 3rd conversion in injected sequence**

Bits 9:5 **JSQ2[4:0]: 2nd conversion in injected sequence**

Bits 4:0 **JSQ1[4:0]: 1st conversion in injected sequence**

16.13.13 ADC injected data register x (ADC_JDRx) (x= 1..4)

Address offset: 0x3C - 0x48

Reset value: 0x0000 0000

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

Bits 31:16 Reserved, must be kept cleared.

Bits 15:0 **JDATA[15:0]: Injected data**

These bits are read only. They contain the conversion result from injected channel x. The data is left or right-aligned as shown in [Figure 146](#) and [Figure 147](#).

16.13.14 ADC regular data register (ADC_DR)

Address offset: 0x4C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADC2DATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 **ADC2DATA[15:0]: ADC2 data**

- In ADC1: In dual mode, these bits contain the regular data of ADC2. Refer to [Section 16.10: Dual ADC mode](#)
- In ADC2: these bits are not used

Bits 15:0 **DATA[15:0]: Regular data**

These bits are read only. They contain the conversion result from the regular channels. The data is left or right-aligned as shown in [Figure 146](#) and [Figure 147](#).

16.14 ADC register map

The following table summarizes the ADC registers.

Table 54. ADC - register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x00	ADC_SR Reset value																																	
0x04	ADC_CR1 Reset value																																	
0x08	ADC_CR2 Reset value																																	
0x0C	ADC_SMPR1 Reset value																																	
0x10	ADC_SMPR2 Reset value																																	
0x14	ADC_JOFR1 Reset value																																	
0x18	ADC_JOFR2 Reset value																																	
0x1C	ADC_JOFR3 Reset value																																	
0x20	ADC_JOFR4 Reset value																																	
0x24	ADC_HTR Reset value																																	
0x28	ADC_LTR Reset value																																	
0x2C	ADC_SQR1 Reset value															L[3:0]																		
0x30	ADC_SQR2 Reset value																																	
0x34	ADC_SQR3 Reset value																																	
0x38	ADC_JSQR Reset value																	JL[1:0]																

Table 54. ADC - register map and reset values (continued)

Refer to [Table 1 on page 27](#) for the register boundary addresses.

17 USB full speed device interface (USB)

17.1 Introduction

The USB peripheral implements an interface between a full-speed USB 2.0 bus and the APB1 bus.

USB suspend/resume are supported which allows to stop the device clocks for low-power consumption.

17.2 Main features

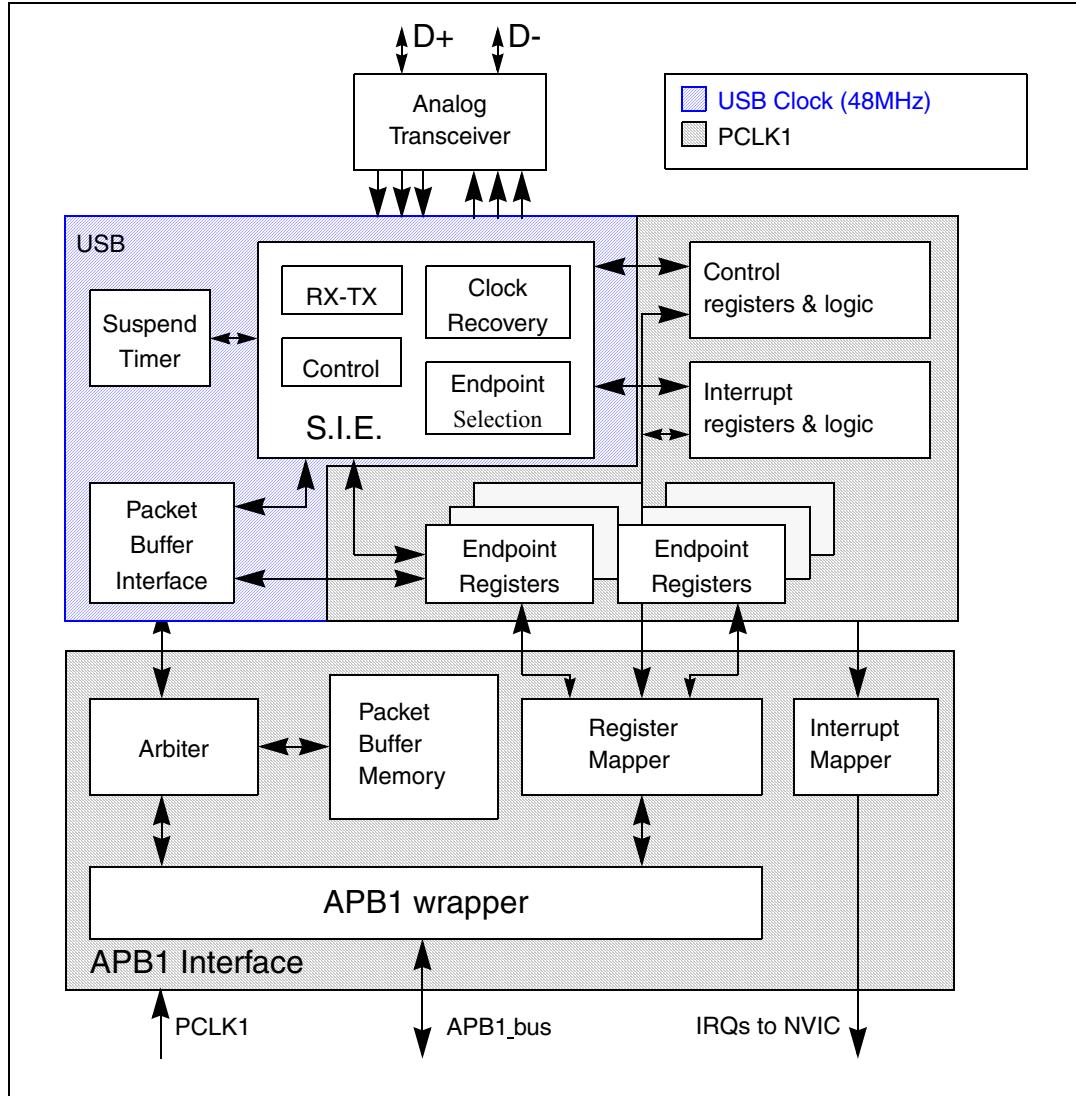
- USB specification version 2.0 full-speed compliant
- Configurable number of endpoints from 1 to 8
- Cyclic redundancy check (CRC) generation/checking, Non-return-to-zero Inverted (NRZI) encoding/decoding and bit-stuffing
- Isochronous transfers support
- Double-buffered bulk/isochronous endpoint support
- USB Suspend/Resume operations
- Frame locked clock pulse generation

Note: *The USB and CAN share a dedicated 512-byte SRAM memory for data transmission and reception, and so they cannot be used concurrently (the shared SRAM is accessed through CAN and USB exclusively). The USB and CAN can be used in the same application but not at the same time.*

17.3 Block diagram

Figure 159 shows the block diagram of the USB peripheral.

Figure 159. USB peripheral block diagram



17.4 Functional description

The USB peripheral provides an USB compliant connection between the host PC and the function implemented by the microcontroller. Data transfer between the host PC and the system memory occurs through a dedicated packet buffer memory accessed directly by the USB peripheral. The size of this dedicated buffer memory must be according to the number of endpoints used and the maximum packet size. This dedicated memory is sized to 512 bytes and up to 16 mono-directional or 8 bidirectional endpoints can be used. The USB peripheral interfaces with the USB host, detecting token packets, handling data transmission/reception, and processing handshake packets as required by the USB standard. Transaction formatting is performed by the hardware, including CRC generation and checking.

Each endpoint is associated with a buffer description block indicating where the endpoint related memory area is located, how large it is or how many bytes must be transmitted.

When a token for a valid function/endpoint pair is recognized by the USB peripheral, the related data transfer (if required and if the endpoint is configured) takes place. The data buffered by the USB peripheral is loaded in an internal 16 bit register and memory access to the dedicated buffer is performed. When all the data has been transferred, if needed, the proper handshake packet over the USB is generated or expected according to the direction of the transfer.

At the end of the transaction, an endpoint-specific interrupt is generated, reading status registers and/or using different interrupt response routines. The microcontroller can determine:

- Which endpoint has to be served
- Which type of transaction took place, if errors occurred (bit stuffing, format, CRC, protocol, missing ACK, over/underrun, etc.)

Special support is offered to Isochronous transfers and high throughput bulk transfers, implementing a double buffer usage, which allows to always have an available buffer for the USB peripheral while the microcontroller uses the other one.

The unit can be placed in low-power mode (SUSPEND mode), by writing in the control register, whenever required. At this time, all static power dissipation is avoided, and the USB clock can be slowed down or stopped. The detection of activity at the USB inputs, while in low-power mode, wakes the device up asynchronously. A special interrupt source can be connected directly to a wakeup line to allow the system to immediately restart the normal clock generation and/or support direct clock start/stop.

17.4.1 Description of USB blocks

The USB peripheral implements all the features related to USB interfacing, which include the following blocks:

- Serial Interface Engine (SIE): The functions of this block include: synchronization pattern recognition, bit-stuffing, CRC generation and checking, PID verification/generation, and handshake evaluation. It must interface with the USB transceivers and uses the virtual buffers provided by the packet buffer interface for local data storage,. This unit also generates signals according to USB peripheral events, such as Start of Frame (SOF), USB_Reset, Data errors etc. and to Endpoint related events like end of transmission or correct reception of a packet; these signals are then used to generate interrupts.
- Timer: This block generates a start-of-frame locked clock pulse and detects a global suspend (from the host) when no traffic has been received for 3 ms.
- Packet Buffer Interface: This block manages the local memory implementing a set of buffers in a flexible way, both for transmission and reception. It can choose the proper buffer according to requests coming from the SIE and locate them in the memory addresses pointed by the Endpoint registers. It increments the address after each exchanged word until the end of packet, keeping track of the number of exchanged bytes and preventing the buffer to overrun the maximum capacity.
- Endpoint-Related Registers: Each endpoint has an associated register containing the endpoint type and its current status. For mono-directional/single-buffer endpoints, a single register can be used to implement two distinct endpoints. The number of registers is 8, allowing up to 16 mono-directional/single-buffer or up to 7 double-buffer

endpoints* in any combination. For example the USB peripheral can be programmed to have 4 double buffer endpoints and 8 single-buffer/mono-directional endpoints.

- Control Registers: These are the registers containing information about the status of the whole USB peripheral and used to force some USB events, such as resume and power-down.
- Interrupt Registers: These contain the Interrupt masks and a record of the events. They can be used to inquire an interrupt reason, the interrupt status or to clear the status of a pending interrupt.

Note:

* *Endpoint 0 is always used for control transfer in single-buffer mode.*

The USB peripheral is connected to the APB1 bus through an APB1 interface, containing the following blocks:

- Packet Memory: This is the local memory that physically contains the Packet Buffers. It can be used by the Packet Buffer interface, which creates the data structure and can be accessed directly by the application software. The size of the Packet Memory is 512 bytes, structured as 256 words by 16 bits.
- Arbiter: This block accepts memory requests coming from the APB1 bus and from the USB interface. It resolves the conflicts by giving priority to APB1 accesses, while always reserving half of the memory bandwidth to complete all USB transfers. This time-duplex scheme implements a virtual dual-port SRAM that allows memory access, while an USB transaction is happening. Multiword APB1 transfers of any length are also allowed by this scheme.
- Register Mapper: This block collects the various byte-wide and bit-wide registers of the USB peripheral in a structured 16-bit wide word set addressed by the APB1.
- Interrupt Mapper: This block is used to select how the possible USB events can generate interrupts and map them to IRQ lines of the NVIC.
- APB1 Wrapper: This provides an interface to the APB1 for the memory and register. It also maps the whole USB peripheral in the APB1 address space.

17.5 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.

17.5.1 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.

17.5.2 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.

As a last step the USB specific 48 MHz clock needs to be activated, using the related control bits provided by device clock management logic.

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, USB function must be enabled, having as USB address 0, implementing only the default control endpoint (endpoint address is 0 too). This is accomplished by setting the Enable Function (EF) bit of the USB_DADDR register and initializing the EP0R register and its related packet buffers accordingly. During USB enumeration process, the host assigns a unique address to this device, which must be written in the ADD[6:0] bits of the USB_DADDR register, and configures any other necessary endpoint.

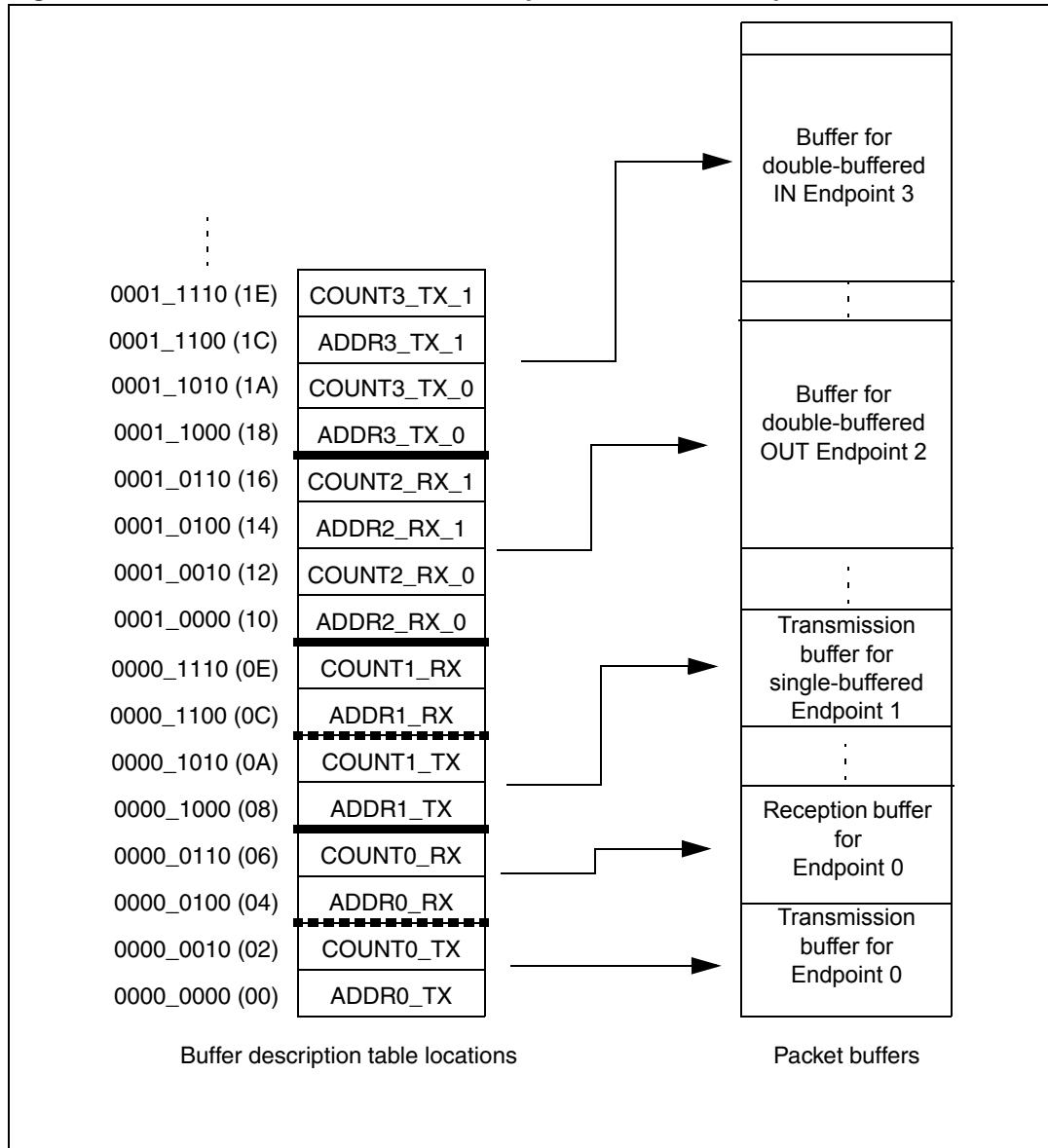
When a RESET interrupt is received, the application software is responsible to enable again the default endpoint of USB function 0 within 10mS from the end of reset sequence which triggered the interrupt.

Structure and usage of packet buffers

Each bidirectional endpoint may receive or transmit data from/to the host. The received data is stored in a dedicated memory buffer reserved for that endpoint, while another memory buffer contains the data to be transmitted by the endpoint. Access to this memory is performed by the packet buffer interface block, which delivers a memory access request and waits for its acknowledgement. Since the packet buffer memory has to be accessed by the microcontroller also, an arbitration logic takes care of the access conflicts, using half APB1 cycle for microcontroller access and the remaining half for the USB peripheral access. In this way, both the agents can operate as if the packet memory is a dual-port SRAM, without being aware of any conflict even when the microcontroller is performing back-to-back accesses. The USB peripheral logic uses a dedicated clock. The frequency of this dedicated clock is fixed by the requirements of the USB standard at 48 MHz, and this can be different from the clock used for the interface to the APB1 bus. Different clock configurations are possible where the APB1 clock frequency can be higher or lower than the USB peripheral one.

Note: *Due to USB data rate and packet memory interface requirements, the APB1 clock frequency must be greater than 8 MHz to avoid data overrun/underrun problems.*

Each endpoint is associated with two packet buffers (usually one for transmission and the other one for reception). Buffers can be placed anywhere inside the packet memory because their location and size is specified in a buffer description table, which is also located in the packet memory at the address indicated by the USB_BTABLE register. Each table entry is associated to an endpoint register and it is composed of four 16-bit words so that table start address must always be aligned to an 8-byte boundary (the lowest three bits of USB_BTABLE register are always “000”). Buffer descriptor table entries are described in the [Section 17.6.3: Buffer descriptor table](#). If an endpoint is unidirectional and it is neither an Isochronous nor a double-buffered bulk, only one packet buffer is required (the one related to the supported transfer direction). Other table locations related to unsupported transfer directions or unused endpoints, are available to the user. Isochronous and double-buffered bulk endpoints have special handling of packet buffers (Refer to [Section 17.5.4: Isochronous transfers](#) and [Section 17.5.3: Double-buffered endpoints](#) respectively). The relationship between buffer description table entries and packet buffer areas is depicted in [Figure 160](#).

Figure 160. Packet buffer areas with examples of buffer description table locations

Each packet buffer is used either during reception or transmission starting from the bottom. The USB peripheral will never change the contents of memory locations adjacent to the allocated memory buffers; if a packet bigger than the allocated buffer length is received (buffer overrun condition) the data will be copied to the memory only up to the last available location.

Endpoint initialization

The first step to initialize an endpoint is to write appropriate values to the ADDRn_TX/ADDRn_RX registers so that the USB peripheral finds the data to be transmitted already available and the data to be received can be buffered. The EP_TYPE bits in the USB_EPnR register must be set according to the endpoint type, eventually using the EP_KIND bit to enable any special required feature. On the transmit side, the endpoint must be enabled using the STAT_TX bits in the USB_EPnR register and COUNTn_TX must be initialized. For reception, STAT_RX bits must be set to enable reception and COUNTn_RX must be written with the allocated buffer size using the BL_SIZE and NUM_BLOCK fields. Unidirectional endpoints, except Isochronous and double-buffered bulk endpoints, need to initialize only bits and registers related to the supported direction. Once the transmission and/or reception are enabled, register USB_EPnR and locations ADDRn_TX/ADDRn_RX, COUNTn_TX/COUNTn_RX (respectively), should not be modified by the application software, as the hardware can change their value on the fly. When the data transfer operation is completed, notified by a CTR interrupt event, they can be accessed again to re-enable a new operation.

IN packets (data transmission)

When receiving an IN token packet, if the received address matches a configured and valid endpoint one, the USB peripheral accesses the contents of ADDRn_TX and COUNTn_TX locations inside buffer descriptor table entry related to the addressed endpoint. The content of these locations is stored in its internal 16 bit registers ADDR and COUNT (not accessible by software). The packet memory is accessed again to read the first word to be transmitted (Refer to [Structure and usage of packet buffers on page 387](#)) and starts sending a DATA0 or DATA1 PID according to USB_EPnR bit DTOG_TX. When the PID is completed, the first byte from the word, read from buffer memory, is loaded into the output shift register to be transmitted on the USB bus. After the last data byte is transmitted, the computed CRC is sent. If the addressed endpoint is not valid, a NAK or STALL handshake packet is sent instead of the data packet, according to STAT_TX bits in the USB_EPnR register.

The ADDR internal register is used as a pointer to the current buffer memory location while COUNT is used to count the number of remaining bytes to be transmitted. Each word read from the packet buffer memory is transmitted over the USB bus starting from the least significant byte. Transmission buffer memory is read starting from the address pointed by ADDRn_TX for COUNTn_TX/2 words. If a transmitted packet is composed of an odd number of bytes, only the lower half of the last word accessed will be used.

On receiving the ACK receipt by the host, the USB_EPnR register is updated in the following way: DTOG_TX bit is toggled, the endpoint is made invalid by setting STAT_TX=10 (NAK) and bit CTR_TX is set. The application software must first identify the endpoint, which is requesting microcontroller attention by examining the EP_ID and DIR bits in the USB_ISTR register. Servicing of the CTR_TX event starts clearing the interrupt bit; the application software then prepares another buffer full of data to be sent, updates the COUNTn_TX table location with the number of byte to be transmitted during the next transfer, and finally sets STAT_TX to '11' (VALID) to re-enable transmissions. While the STAT_TX bits are equal to '10' (NAK), any IN request addressed to that endpoint is NAKed, indicating a flow control condition: the USB host will retry the transaction until it succeeds. It is mandatory to execute the sequence of operations in the above mentioned order to avoid losing the notification of a second IN transaction addressed to the same endpoint immediately following the one which triggered the CTR interrupt.

OUT and SETUP packets (data reception)

These two tokens are handled by the USB peripheral more or less in the same way; the differences in the handling of SETUP packets are detailed in the following paragraph about control transfers. When receiving an OUT/SETUP PID, if the address matches a valid endpoint, the USB peripheral accesses the contents of the ADDRn_RX and COUNTn_RX locations inside the buffer descriptor table entry related to the addressed endpoint. The content of the ADDRn_RX is stored directly in its internal register ADDR. While COUNT is now reset and the values of BL_SIZE and NUM_BLOCK bit fields, which are read within COUNTn_RX content are used to initialize BUF_COUNT, an internal 16 bit counter, which is used to check the buffer overrun condition (all these internal registers are not accessible by software). Data bytes subsequently received by the USB peripheral are packed in words (the first byte received is stored as least significant byte) and then transferred to the packet buffer starting from the address contained in the internal ADDR register while BUF_COUNT is decremented and COUNT is incremented at each byte transfer. When the end of DATA packet is detected, the correctness of the received CRC is tested and only if no errors occurred during the reception, an ACK handshake packet is sent back to the transmitting host. In case of wrong CRC or other kinds of errors (bit-stuff violations, frame errors, etc.), data bytes are still copied in the packet memory buffer, at least until the error detection point, but ACK packet is not sent and the ERR bit in USB_ISTR register is set. However, there is usually no software action required in this case: the USB peripheral recovers from reception errors and remains ready for the next transaction to come. If the addressed endpoint is not valid, a NAK or STALL handshake packet is sent instead of the ACK, according to bits STAT_RX in the USB_EPnR register and no data is written in the reception memory buffers.

Reception memory buffer locations are written starting from the address contained in the ADDRn_RX for a number of bytes corresponding to the received data packet length, CRC included (i.e. data payload length + 2), or up to the last allocated memory location, as defined by BL_SIZE and NUM_BLOCK, whichever comes first. In this way, the USB peripheral never writes beyond the end of the allocated reception memory buffer area. If the length of the data packet payload (actual number of bytes used by the application) is greater than the allocated buffer, the USB peripheral detects a buffer overrun condition. in this case, a STALL handshake is sent instead of the usual ACK to notify the problem to the host, no interrupt is generated and the transaction is considered failed.

When the transaction is completed correctly, by sending the ACK handshake packet, the internal COUNT register is copied back in the COUNTn_RX location inside the buffer description table entry, leaving unaffected BL_SIZE and NUM_BLOCK fields, which normally do not require to be re-written, and the USB_EPnR register is updated in the following way: DTOG_RX bit is toggled, the endpoint is made invalid by setting STAT_RX = '10' (NAK) and bit CTR_RX is set. If the transaction has failed due to errors or buffer overrun condition, none of the previously listed actions take place. The application software must first identify the endpoint, which is requesting microcontroller attention by examining the EP_ID and DIR bits in the USB_ISTR register. The CTR_RX event is serviced by first determining the transaction type (SETUP bit in the USB_EPnR register); the application software must clear the interrupt flag bit and get the number of received bytes reading the COUNTn_RX location inside the buffer description table entry related to the endpoint being processed. After the received data is processed, the application software should set the STAT_RX bits to '11' (Valid) in the USB_EPnR, enabling further transactions. While the STAT_RX bits are equal to '10' (NAK), any OUT request addressed to that endpoint is NAKed, indicating a flow control condition: the USB host will retry the transaction until it succeeds. It is mandatory to execute the sequence of operations in the above mentioned order to avoid losing the notification of a second OUT transaction addressed to the same endpoint following immediately the one which triggered the CTR interrupt.

Control transfers

Control transfers are made of a SETUP transaction, followed by zero or more data stages, all of the same direction, followed by a status stage (a zero-byte transfer in the opposite direction). SETUP transactions are handled by control endpoints only and are very similar to OUT ones (data reception) except that the values of DTOG_TX and DTOG_RX bits of the addressed endpoint registers are set to 1 and 0 respectively, to initialize the control transfer, and both STAT_TX and STAT_RX are set to '10' (NAK) to let software decide if subsequent transactions must be IN or OUT depending on the SETUP contents. A control endpoint must check SETUP bit in the USB_EPnR register at each CTR_RX event to distinguish normal OUT transactions from SETUP ones. A USB device can determine the number and direction of data stages by interpreting the data transferred in the SETUP stage, and is required to STALL the transaction in the case of errors. To do so, at all data stages before the last, the unused direction should be set to STALL, so that, if the host reverses the transfer direction too soon, it gets a STALL as a status stage. While enabling the last data stage, the opposite direction should be set to NAK, so that, if the host reverses the transfer direction (to perform the status stage) immediately, it is kept waiting for the completion of the control operation. If the control operation completes successfully, the software will change NAK to VALID, otherwise to STALL. At the same time, if the status stage will be an OUT, the STATUS_OUT (EP_KIND in the USB_EPnR register) bit should be set, so that an error is generated if a status transaction is performed with not-zero data. When the status transaction is serviced, the application clears the STATUS_OUT bit and sets STAT_RX to VALID (to accept a new command) and STAT_TX to NAK (to delay a possible status stage immediately following the next setup).

Since the USB specification states that a SETUP packet cannot be answered with a handshake different from ACK, eventually aborting a previously issued command to start the new one, the USB logic doesn't allow a control endpoint to answer with a NAK or STALL packet to a SETUP token received from the host.

When the STAT_RX bits are set to '01' (STALL) or '10' (NAK) and a SETUP token is received, the USB accepts the data, performing the required data transfers and sends back an ACK handshake. If that endpoint has a previously issued CTR_RX request not yet acknowledged by the application (i.e. CTR_RX bit is still set from a previously completed reception), the USB discards the SETUP transaction and does not answer with any handshake packet regardless of its state, simulating a reception error and forcing the host to send the SETUP token again. This is done to avoid losing the notification of a SETUP transaction addressed to the same endpoint immediately following the transaction, which triggered the CTR_RX interrupt.

17.5.3 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 55. Double-buffering buffer flag definition

Buffer flag	'Transmission' endpoint	'Reception' endpoint
DTOG	DTOG_TX (USB_EPnRbit 6)	DTOG_RX (USB_EPnRbit 14)
SW_BUF	USB_EPnR bit 14	USB_EPnR bit 6

The memory buffer which is currently being used by the USB peripheral is defined by DTOG buffer flag, while the buffer currently in use by application software is identified by SW_BUF buffer flag. The relationship between the buffer flag value and the used packet buffer is the same in both cases, and it is listed in the following table.

Table 56. 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 *	ADDRn_TX_0 / COUNTn_TX_0 Buffer description table locations.
	1	1	None *	ADDRn_TX_0 / COUNTn_TX_0 Buffer description table locations.
OUT	0	1	ADDRn_RX_0 / COUNTn_RX_0 Buffer description table locations.	ADDRn_RX_1 / COUNTn_RX_1 Buffer description table locations.
	1	0	ADDRn_RX_1 / COUNTn_RX_1 Buffer description table locations.	ADDRn_RX_0 / COUNTn_RX_0 Buffer description table locations.
	0	0	None *	ADDRn_RX_0 / COUNTn_RX_0 Buffer description table locations.
	1	1	None *	ADDRn_RX_1 / COUNTn_RX_1 Buffer description table locations.

Note:

* Endpoint in NAK Status.

Double-buffering feature for a bulk endpoint is activated by:

- Writing EP_TYPE bit field at '00' in its USB_EPnR register, to define the endpoint as a bulk, and
- Setting EP_KIND bit at '1' (DBL_BUF), in the same register.

The application software is responsible for DTOG and SW_BUF bits initialization according to the first buffer to be used; this has to be done considering the special toggle-only property that these two bits have. The end of the first transaction occurring after having set DBL_BUF, triggers the special flow control of double-buffered bulk endpoints, which is used for all other transactions addressed to this endpoint until DBL_BUF remain set. At the end of each transaction the CTR_RX or CTR_TX bit of the addressed endpoint USB_EPnR register is set, depending on the enabled direction. At the same time, the affected DTOG bit in the USB_EPnR register is hardware toggled making the USB peripheral buffer swapping completely software independent. Unlike common transactions, and the first one after DBL_BUF setting, STAT bit pair is not affected by the transaction termination and its value

remains ‘11’ (Valid). However, as the token packet of a new transaction is received, the actual endpoint status will be masked as ‘10’ (NAK) when a buffer conflict between the USB peripheral and the application software is detected (this condition is identified by DTOG and SW_BUF having the same value, see [Table 56 on page 394](#)). The application software responds to the CTR event notification by clearing the interrupt flag and starting any required handling of the completed transaction. When the application packet buffer usage is over, the software toggles the SW_BUF bit, writing ‘1’ to it, to notify the USB peripheral about the availability of that buffer. In this way, the number of NAKed transactions is limited only by the application elaboration time of a transaction data: if the elaboration time is shorter than the time required to complete a transaction on the USB bus, no re-transmissions due to flow control will take place and the actual transfer rate will be limited only by the host PC.

The application software can always override the special flow control implemented for double-buffered bulk endpoints, writing an explicit status different from ‘11’ (Valid) into the STAT bit pair of the related USB_EPnR register. In this case, the USB peripheral will always use the programmed endpoint status, regardless of the buffer usage condition.

17.5.4 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 57](#).

Table 57. Isochronous memory buffers usage

Endpoint Type	DTOG bit value	Packet buffer used by the USB peripheral	Packet buffer used by the application software
IN	0	ADDRn_TX_0 / COUNTn_TX_0 buffer description table locations.	ADDRn_TX_1 / COUNTn_TX_1 buffer description table locations.
	1	ADDRn_TX_1 / COUNTn_TX_1 buffer description table locations.	ADDRn_TX_0 / COUNTn_TX_0 buffer description table locations.
OUT	0	ADDRn_RX_0 / COUNTn_RX_0 buffer description table locations.	ADDRn_RX_1 / COUNTn_RX_1 buffer description table locations.
	1	ADDRn_RX_1 / COUNTn_RX_1 buffer description table locations.	ADDRn_RX_0 / COUNTn_RX_0 buffer description table locations.

As it happens with double-buffered bulk endpoints, the USB_EPnR registers used to implement Isochronous endpoints are forced to be used as unidirectional ones. In case it is required to have Isochronous endpoints enabled both for reception and transmission, two USB_EPnR registers must be used.

The application software is responsible for the DTOG bit initialization according to the first buffer to be used; this has to be done considering the special toggle-only property that these two bits have. At the end of each transaction, the CTR_RX or CTR_TX bit of the addressed endpoint USB_EPnR register is set, depending on the enabled direction. At the same time, the affected DTOG bit in the USB_EPnR register is hardware toggled making buffer swapping completely software independent. STAT bit pair is not affected by transaction completion; since no flow control is possible for Isochronous transfers due to the lack of handshake phase, the endpoint remains always '11' (Valid). CRC errors or buffer-overrun conditions occurring during Isochronous OUT transfers are anyway considered as correct transactions and they always trigger an CTR_RX event. However, CRC errors will anyway set the ERR bit in the USB_ISTR register to notify the software of the possible data corruption.

17.5.5 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 500 µA. This requirement is of fundamental importance for bus-powered devices, while self-powered devices are not required to comply to this strict power consumption constraint. In suspend mode, the host PC sends the notification to not send any traffic on the USB bus for more than 3mS: since a SOF packet must be sent every mS during normal operations, the USB peripheral detects the lack of 3 consecutive SOF packets as a suspend request from the host PC and set the SUSP bit to '1' in USB_ISTR register, causing an interrupt if enabled. Once the device is suspended, its normal operation can be restored by a so called RESUME sequence, which can be started from the host PC or directly from the peripheral itself, but it is always terminated by the host PC. The suspended USB peripheral must be anyway able to detect a RESET sequence, reacting to this event as a normal USB reset event.

The actual procedure used to suspend the USB peripheral is device dependent since according to the device composition, different actions may be required to reduce the total consumption.

A brief description of a typical suspend procedure is provided below, focused on the USB-related aspects of the application software routine responding to the SUSP notification of the USB peripheral:

1. Set the FSUSP bit in the USB_CNTR register to 1. This action activates the suspend mode within the USB peripheral. As soon as the suspend mode is activated, the check on SOF reception is disabled to avoid any further SUSP interrupts being issued while the USB is suspended.
2. Remove or reduce any static power consumption in blocks different from the USB peripheral.
3. Set LP_MODE bit in USB_CNTR register to 1 to remove static power consumption in the analog USB transceivers but keeping them able to detect resume activity.
4. Optionally turn off external oscillator and device PLL to stop any activity inside the device.

When an USB event occurs while the device is in SUSPEND mode, the RESUME procedure must be invoked to restore nominal clocks and regain normal USB behavior. Particular care must be taken to insure that this process does not take more than 10mS when the wakening event is an USB reset sequence (See “Universal Serial Bus Specification” for more details). The start of a resume or reset sequence, while the USB peripheral is suspended, clears the LP_MODE bit in USB_CNTR register asynchronously. Even if this event can trigger an WKUP interrupt if enabled, the use of an interrupt response routine must be carefully evaluated because of the long latency due to system clock restart; to have the shorter latency before re-activating the nominal clock it is suggested to put the resume procedure just after the end of the suspend one, so its code is immediately executed as soon as the system clock restarts. To prevent ESD discharges or any other kind of noise from waking-up the system (the exit from suspend mode is an asynchronous event), a suitable analog filter on data line status is activated during suspend; the filter width is about 70ns.

The following is a list of actions a resume procedure should address:

1. Optionally turn on external oscillator and/or device PLL.
2. Clear FSUSP bit of USB_CNTR register.
3. If the resume triggering event has to be identified, bits RXDP and RXDM in the USB_FNR register can be used according to [Table 58](#), 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 58. Resume event detection

[RXDP,RXDM] Status	Wakeup event	Required resume software action
“00”	Root reset	None
“10”	None (noise on bus)	Go back in Suspend mode
“01”	Root resume	None
“11”	Not Allowed (noise on bus)	Go back in Suspend mode

A device may require to exit from suspend mode as an answer to particular events not directly related to the USB protocol (e.g. a mouse movement wakes up the whole system).

In this case, the resume sequence can be started by setting the RESUME bit in the USB_CNTR register to ‘1’ and resetting it to 0 after an interval between 1mS and 15mS (this interval can be timed using ESOF interrupts, occurring with a 1mS period when the system clock is running at nominal frequency). Once the RESUME bit is clear, the resume sequence will be completed by the host PC and its end can be monitored again using the RXDP and RXDM bits in the USB_FNR register.

Note: *The RESUME bit must be anyway used only after the USB peripheral has been put in suspend mode, setting the FSUSP bit in USB_CNTR register to 1.*

17.6 USB register description

The USB peripheral registers can be divided into the following groups:

- Common Registers: Interrupt and Control registers
- Endpoint Registers: Endpoint configuration and status
- Buffer Descriptor Table: Location of packet memory used to locate data buffers

All register addresses are expressed as offsets with respect to the USB peripheral registers base address 0xC000 8000, except the buffer descriptor table locations, which starts at the address specified by the USB_BTABLE register. Due to the common limitation of APB1 bridges on word addressability, all register addresses are aligned to 32-bit word boundaries although they are 16-bit wide. The same address alignment is used to access packet buffer memory locations, which are located starting from 0xC000 8800.

Refer to [Section 1.1 on page 23](#) for a list of abbreviations used in register descriptions.

17.6.1 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
CTRM	PMAOVRM	ERRM	WKUPM	SUSPM	RESETM	SOFM	ESOFM	Reserved	RESUME	FSUSP	LP_MODE	PDWN	FRES		

Bit 15 **CTRM: Correct Transfer Interrupt Mask**

0: Correct Transfer (CTR) Interrupt disabled.

1: CTR Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 14 **PMAOVRM: Packet Memory Area Over / Underrun Interrupt Mask**

0: PMAOVR Interrupt disabled.

1: PMAOVR Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 13 **ERRM: Error Interrupt Mask**

- 0: ERR Interrupt disabled.
- 1: ERR Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 12 **WKUPM: Wakeup Interrupt Mask**

- 0: WKUP Interrupt disabled.
- 1: WKUP Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 11 **SUSPM: Suspend mode Interrupt Mask**

- 0: Suspend Mode Request (SUSP) Interrupt disabled.
- 1: SUSP Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 10 **RESETM: USB Reset Interrupt Mask**

- 0: RESET Interrupt disabled.
- 1: RESET Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 9 **SOFM: Start Of Frame Interrupt Mask**

- 0: SOF Interrupt disabled.
- 1: SOF Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 8 **ESOFM: Expected Start Of Frame Interrupt Mask**

- 0: Expected Start of Frame (ESOF) Interrupt disabled.
- 1: ESOF Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bits 7:5 Reserved.

Bit 4 **RESUME: Resume request**

The microcontroller can set this bit to send a Resume signal to the host. It must be activated, according to USB specifications, for no less than 1mS and no more than 15mS after which the Host PC is ready to drive the resume sequence up to its end.

Bit 3 **FSUSP: Force suspend**

Software must set this bit when the SUSP interrupt is received, which is issued when no traffic is received by the USB peripheral for 3 mS.

- 0: No effect.
- 1: Enter suspend mode. Clocks and static power dissipation in the analog transceiver are left unaffected. If suspend power consumption is a requirement (bus-powered device), the application software should set the LP_MODE bit after FSUSP as explained below.

Bit 2 **LP_MODE: Low-power mode**

This mode is used when the suspend-mode power constraints require that all static power dissipation is avoided, except the one required to supply the external pull-up resistor. This condition should be entered when the application is ready to stop all system clocks, or reduce their frequency in order to meet the power consumption requirements of the USB suspend condition. The USB activity during the suspend mode (WKUP event) asynchronously resets this bit (it can also be reset by software).

- 0: No Low-power mode.
- 1: Enter Low-power mode.

Bit 1 PDWN: Power down

This bit is used to completely switch off all USB-related analog parts if it is required to completely disable the USB peripheral for any reason. When this bit is set, the USB peripheral is disconnected from the transceivers and it cannot be used.

0: Exit Power Down.

1: Enter Power down mode.

Bit 0 FRES: Force USB Reset

0: Clear USB reset.

1: Force a reset of the USB peripheral, exactly like a RESET signalling on the USB. The USB peripheral is held in RESET state until software clears this bit. A “USB-RESET” interrupt is generated, if enabled.

USB interrupt status register (USB_ISTR)

Address offset: 0x44

Reset value: 0x0000 0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
CTR	PMA OVR	ERR	WKUP	SUSP	RESET	SOF	ESOF	Reserved		DIR	EP_ID[3:0]					
r	rc	rc	rc	rc	rc	rc	rc	Res.		r	r	r	r	r	r	

This register contains the status of all the interrupt sources allowing application software to determine, which events caused an interrupt request.

The upper part of this register contains single bits, each of them representing a specific event. These bits are set by the hardware when the related event occurs; if the corresponding bit in the USB_CNTR register is set, a generic interrupt request is generated. The interrupt routine, examining each bit, will perform all necessary actions, and finally it will clear the serviced bits. If any of them is not cleared, the interrupt is considered to be still pending, and the interrupt line will be kept high again. If several bits are set simultaneously, only a single interrupt will be generated.

Endpoint transaction completion can be handled in a different way to reduce interrupt response latency. The CTR bit is set by the hardware as soon as an endpoint successfully completes a transaction, generating a generic interrupt request if the corresponding bit in USB_CNTR is set. An endpoint dedicated interrupt condition is activated independently from the CTRM bit in the USB_CNTR register. Both interrupt conditions remain active until software clears the pending bit in the corresponding USB_EPnR register (the CTR bit is actually a read only bit). USB peripheralFor endpoint-related interrupts, the software can use the Direction of Transaction (DIR) and EP_ID read-only bits to identify, which endpoint made the last interrupt request and called the corresponding interrupt service routine.

The user can choose the relative priority of simultaneously pending USB_ISTR events by specifying the order in which software checks USB_ISTR bits in an interrupt service routine. Only the bits related to events, which are serviced, are cleared. At the end of the service routine, another interrupt will be requested, to service the remaining conditions.

To avoid spurious clearing of some bits, it is recommended to clear them with a load instruction where all bits which must not be altered are written with 1, and all bits to be cleared are written with ‘0’ (these bits can only be cleared by software). Read-modify-write cycles should be avoided because between the read and the write operations another bit

could be set by the hardware and the next write will clear it before the microprocessor has the time to serve the event.

The following describes each bit in detail:

Bit 15 CTR: Correct Transfer

This bit is set by the hardware to indicate that an endpoint has successfully completed a transaction; using DIR and EP_ID bits software can determine which endpoint requested the interrupt. This bit is read-only.

Bit 14 PMAOVR: Packet Memory Area Over / Underrun

This bit is set if the microcontroller has not been able to respond in time to an USB memory request. The USB peripheral handles this event in the following way: During reception an ACK handshake packet is not sent, during transmission a bit-stuff error is forced on the transmitted stream; in both cases the host will retry the transaction. The PMAOVR interrupt should never occur during normal operations. Since the failed transaction is retried by the host, the application software has the chance to speed-up device operations during this interrupt handling, to be ready for the next transaction retry; however this does not happen during Isochronous transfers (no isochronous transaction is anyway retried) leading to a loss of data in this case. This bit is read/write but only '0' can be written and writing '1' has no effect.

Bit 13 ERR: Error

This flag is set whenever one of the errors listed below has occurred:

NANS: No ANSwer. The timeout for a host response has expired.

CRC: Cyclic Redundancy Check error. One of the received CRCs, either in the token or in the data, was wrong.

BST: Bit Stuffing error. A bit stuffing error was detected anywhere in the PID, data, and/or CRC.

FVIO: Framing format Violation. A non-standard frame was received (EOP not in the right place, wrong token sequence, etc.).

The USB software can usually ignore errors, since the USB peripheral and the PC host manage retransmission in case of errors in a fully transparent way. This interrupt can be useful during the software development phase, or to monitor the quality of transmission over the USB bus, to flag possible problems to the user (e.g. loose connector, too noisy environment, broken conductor in the USB cable and so on). This bit is read/write but only '0' can be written and writing '1' has no effect.

Bit 12 WKUP: Wake up

This bit is set to 1 by the hardware when, during suspend mode, activity is detected that wakes up the USB peripheral. This event asynchronously clears the LP_MODE bit in the CTLR register and activates the USB_WAKEUP line, which can be used to notify the rest of the device (e.g. wakeup unit) about the start of the resume process. This bit is read/write but only '0' can be written and writing '1' has no effect.

Bit 11 SUSP: Suspend mode request

This bit is set by the hardware when no traffic has been received for 3mS, indicating a suspend mode request from the USB bus. The suspend condition check is enabled immediately after any USB reset and it is disabled by the hardware when the suspend mode is active (FSUSP=1) until the end of resume sequence. This bit is read/write but only '0' can be written and writing '1' has no effect.

Bit 10 RESET: USB RESET request

Set when the USB peripheral detects an active USB RESET signal at its inputs. The USB peripheral, in response to a RESET, just resets its internal protocol state machine, generating an interrupt if RESETM enable bit in the USB_CNTR register is set. Reception and transmission are disabled until the RESET bit is cleared. All configuration registers do not reset: the microcontroller must explicitly clear these registers (this is to ensure that the RESET interrupt can be safely delivered, and any transaction immediately followed by a RESET can be completed). The function address and endpoint registers are reset by an USB reset event.

This bit is read/write but only '0' can be written and writing '1' has no effect.

Bit 9 *SOF: Start Of Frame*

This bit signals the beginning of a new USB frame and it is set when a SOF packet arrives through the USB bus. The interrupt service routine may monitor the SOF events to have a 1mS synchronization event to the USB host and to safely read the USB_FNR register which is updated at the SOF packet reception (this could be useful for isochronous applications). This bit is read/write but only '0' can be written and writing '1' has no effect.

Bit 8 *ESOF: Expected Start Of Frame*

This bit is set by the hardware when an SOF packet is expected but not received. The host sends an SOF packet each mS, but if the hub does not receive it properly, the 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 Timer is not yet locked. This bit is read/write but only '0' can be written and writing '1' has no effect.

Bits 7:5 Reserved.

Bit 4 *DIR: Direction of transaction.*

This bit is written by the hardware according to the direction of the successful transaction, which generated the interrupt request.

If DIR bit=0, CTR_TX bit is set in the USB_EPnR register related to the interrupting endpoint. The interrupting transaction is of IN type (data transmitted by the USB peripheral to the host PC). If DIR bit=1, CTR_RX bit or both CTR_TX/CTR_RX are set in the USB_EPnR register related to the interrupting endpoint. The interrupting transaction is of OUT type (data received by the USB peripheral from the host PC) or two pending transactions are waiting to be processed.

This information can be used by the application software to access the USB_EPnR bits related to the triggering transaction since it represents the direction having the interrupt pending. This bit is read-only.

Bits 3:0 *EP_ID[3:0]: Endpoint Identifier.*

These bits are written by the hardware according to the endpoint number, which generated the interrupt request. If several endpoint transactions are pending, the hardware writes the endpoint identifier related to the endpoint having the highest priority defined in the following way: Two endpoint sets are defined, in order of priority: Isochronous and double-buffered bulk endpoints are considered first and then the other endpoints are examined. If more than one endpoint from the same set is requesting an interrupt, the EP_ID bits in USB_ISTR register are assigned according to the lowest requesting endpoint register, EP0R having the highest priority followed by EP1R and so on. The application software can assign a register to each endpoint according to this priority scheme, so as to order the concurring endpoint requests in a suitable way. These bits are read only.

USB frame number register (USB_FNR)

Address offset: 0x48

Reset value: 0x0XXX where X is undefined

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
RXDP	RXDM	LCK	LSOF[1:0]		FN[10:0]													
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r			

Bit 15 **RXDP: Receive Data + Line Status**

This bit can be used to observe the status of received data plus upstream port data line. It can be used during end-of-suspend routines to help determining the wakeup event.

Bit 14 **RXDM: Receive Data - Line Status**

This bit can be used to observe the status of received data minus upstream port data line. It can be used during end-of-suspend routines to help determining the wakeup event.

Bit 13 **LCK: Locked**

This bit is set by the hardware when at least two consecutive SOF packets have been received after the end of an USB reset condition or after the end of an USB resume sequence. Once locked, the frame timer remains in this state until an USB reset or USB suspend event occurs.

Bits 12:11 **LSOF[1:0]: Lost SOF**

These bits are written by the hardware when an ESOF interrupt is generated, counting the number of consecutive SOF packets lost. At the reception of an SOF packet, these bits are cleared.

Bits 10:0 **FN[10:0]: Frame Number**

This bit field contains the 11-bits frame number contained in the last received SOF packet. The frame number is incremented for every frame sent by the host and it is useful for Isochronous transfers. This bit field is updated on the generation of an SOF interrupt.

USB device address (USB_DADDR)

Address offset: 0x4C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								EF	ADD6	ADD5	ADD4	ADD3	ADD2	ADD1	ADD0
							Res.	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:8 Reserved

Bit 7 **EF: Enable Function**

This bit is set by the software to enable the USB device. The address of this device is contained in the following ADD[6:0] bits. If this bit is at '0' no transactions are handled, irrespective of the settings of USB_EPnR registers.

Bits 6:0 **ADD[6:0]: Device Address**

These bits contain the USB function address assigned by the host PC during the enumeration process. Both this field and the Endpoint Address (EA) field in the associated USB_EPnR register must match with the information contained in a USB token in order to handle a transaction to the required endpoint.

Buffer table address (USB_BTABLE)

Address offset: 0x50

Reset value: 0x0000

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

Bits 15:3 **BTABLE[15:3]: Buffer Table.**

These bits contain the start address of the buffer allocation table inside the dedicated packet memory. This table describes each endpoint buffer location and size and it must be aligned to an 8 byte boundary (the 3 least significant bits are always '0'). At the beginning of every transaction addressed to this device, the USP peripheral reads the element of this table related to the addressed endpoint, to get its buffer start location and the buffer size (Refer to [Structure and usage of packet buffers on page 387](#)).

Bits 2:0 Reserved, forced by hardware to 0.

17.6.2 Endpoint-specific registers

The number of these registers varies according to the number of endpoints that the USB peripheral is designed to handle. The USB peripheral supports up to 8 bidirectional endpoints. Each USB device must support a control endpoint whose address (EA bits) must be set to 0. The USB peripheral behaves in an undefined way if multiple endpoints are enabled having the same endpoint number value. For each endpoint, an USB_EPnR register is available to store the endpoint specific information.

USB endpoint n register (USB_EPnR), n=[0..7]

Address offset: 0x00 to 0x1C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTR_RX	DTOG_RX	STAT_X[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR_TX	DTOG_TX	STAT_TX[1:0]		EA[3:0]			
r-c	t	t	t	r	rw	rw	rw	r-c	t	t	t	rw	rw	rw	rw

They are also reset when an USB reset is received from the USB bus or forced through bit FRES in the CTLR register, except the CTR_RX and CTR_TX bits, which are kept unchanged to avoid missing a correct packet notification immediately followed by an USB reset event. Each endpoint has its USB_EPnR register where *n* is the endpoint identifier.

Read-modify-write cycles on these registers should be avoided because between the read and the write operations some bits could be set by the hardware and the next write would modify them before the CPU has the time to detect the change. For this purpose, all bits affected by this problem have an ‘invariant’ value that must be used whenever their modification is not required. It is recommended to modify these registers with a load instruction where all the bits, which can be modified only by the hardware, are written with their ‘invariant’ value.

Bit 15 CTR_RX: Correct Transfer for reception

This bit is set by the hardware when an OUT/SETUP transaction is successfully completed on this endpoint; the software can only clear this bit. If the CTRM bit in USB_CNTR register is set accordingly, a generic interrupt condition is generated together with the endpoint related interrupt condition, which is always activated. The type of occurred transaction, OUT or SETUP, can be determined from the SETUP bit described below.

A transaction ended with a NAK or STALL handshake does not set this bit, since no data is actually transferred, as in the case of protocol errors or data toggle mismatches.

This bit is read/write but only ‘0’ can be written, writing 1 has no effect.

Bit 14 DTOG_RX: Data Toggle, for reception transfers

If the endpoint is not Isochronous, this bit contains the expected value of the data toggle bit (0=DATA0, 1=DATA1) for the next data packet to be received. Hardware toggles this bit, when the ACK handshake is sent to the USB host, following a data packet reception having a matching data PID value; if the endpoint is defined as a control one, hardware clears this bit at the reception of a SETUP PID addressed to this endpoint.

If the endpoint is using the double-buffering feature this bit is used to support packet buffer swapping too (Refer to [Section 17.5.3: 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 17.5.4: 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 59: Reception status encoding on page 408](#). 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 17.5.3: 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 60: Endpoint type encoding on page 408](#). Endpoint 0 must always be a control endpoint and each USB function must have at least one control endpoint which has address 0, but there may be other control endpoints if required. Only control endpoints handle SETUP transactions, which are ignored by endpoints of other kinds. SETUP transactions cannot be answered with NAK or STALL. If a control endpoint is defined as NAK, the USB peripheral will not answer, simulating a receive error, in the receive direction when a SETUP transaction is received. If the control endpoint is defined as STALL in the receive direction, then the SETUP packet will be accepted anyway, transferring data and issuing the CTR interrupt. The reception of OUT transactions is handled in the normal way, even if the endpoint is a control one.

Bulk and interrupt endpoints have very similar behavior and they differ only in the special feature available using the EP_KIND configuration bit.

The usage of Isochronous endpoints is explained in [Section 17.5.4: 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 61](#) 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 17.5.3: 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 17.5.3: 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 17.5.4: 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 62](#). 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 17.5.3: 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 59. Reception status encoding

STAT_RX[1:0]	Meaning
00	DISABLED : all reception requests addressed to this endpoint are ignored.
01	STALL : the endpoint is stalled and all reception requests result in a STALL handshake.
10	NAK : the endpoint is naked and all reception requests result in a NAK handshake.
11	VALID : this endpoint is enabled for reception.

Table 60. Endpoint type encoding

EP_TYPE[1:0]	Meaning
00	BULK
01	CONTROL
10	ISO
11	INTERRUPT

Table 61. 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 62. Transmission status encoding

STAT_TX[1:0]	Meaning
00	DISABLED : all transmission requests addressed to this endpoint are ignored.
01	STALL : the endpoint is stalled and all transmission requests result in a STALL handshake.
10	NAK : the endpoint is naked and all transmission requests result in a NAK handshake.
11	VALID : this endpoint is enabled for transmission.

17.6.3 Buffer descriptor table

Although the buffer descriptor table is located inside the packet buffer memory, its entries can be considered as additional registers used to configure the location and size of the packet buffers used to exchange data between the USB macro cell and the STM32F10xxx. Due to the common APB bridge limitation on word addressability, all packet memory locations are accessed by the APB using 32-bit aligned addresses, instead of the actual memory location addresses utilized by the USB peripheral for the USB_BTABLE register and buffer description table locations.

In the following pages two location addresses are reported: the one to be used by application software while accessing the packet memory, and the local one relative to USB Peripheral access. To obtain the correct STM32F10xxx memory address value to be used in the application software while accessing the packet memory, the actual memory location

address must be multiplied by two. The first packet memory location is located at 0x4000 6000. The buffer descriptor table entry associated with the USB_EPnR registers is described below.

A thorough explanation of packet buffers and the buffer descriptor table usage can be found in [Structure and usage of packet buffers on page 387](#).

Transmission buffer address n (USB_ADDRn_TX)

Address offset: [USB_BTABLE] + n*16

USB local Address: [USB_BTABLE + n*8]

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

Bits 15:1 ADDRn_TX[15:1]: Transmission Buffer Address

These bits point to the starting address of the packet buffer containing data to be transmitted by the endpoint associated with the USB_EPnR register at the next IN token addressed to it.

Bit 0 Must always be written as '0' since packet memory is word-wide and all packet buffers must be word-aligned.

Transmission byte count n (USB_COUNTn_TX)

Address offset: [USB_BTABLE] + n*16 + 4

USB local Address: [USB_BTABLE] + n*8 + 2

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

Bits 15:10 These bits are not used since packet size is limited by USB specifications to 1023 bytes. Their value is not considered by the USB peripheral.

Bits 9:0 COUNTn_TX[9:0]: Transmission Byte Count

These bits contain the number of bytes to be transmitted by the endpoint associated with the USB_EPnR register at the next IN token addressed to it.

Note: Double-buffered and Isochronous IN Endpoints have two USB_COUNTn_TX registers: named USB_COUNTn_TX_1 and USB_COUNTn_TX_0 with the following content.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	COUNTn_TX_1[9:0]
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	COUNTn_TX_0[9:0]
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	rw

Reception buffer address n (USB_ADDRn_RX)

Address offset: [USB_BTABLE] + n*16 + 8

USB local Address: [USB_BTABLE] + n*8 + 4

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

Bits 15:1 ADDRn_RX[15:1]: Reception Buffer Address

These bits point to the starting address of the packet buffer, which will contain the data received by the endpoint associated with the USB_EPnR register at the next OUT/SETUP token addressed to it.

Bit 0 This bit must always be written as '0' since packet memory is word-wide and all packet buffers must be word-aligned.

Reception byte count n (USB_COUNTn_RX)

Address offset: [USB_BTABLE] + n*16 + 12

USB local Address: [USB_BTABLE] + n*8 + 6

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

This table location is used to store two different values, both required during packet reception. The most significant bits contains the definition of allocated buffer size, to allow buffer overflow detection, while the least significant part of this location is written back by the USB peripheral at the end of reception to give the actual number of received bytes. Due to the restrictions on the number of available bits, buffer size is represented using the number of allocated memory blocks, where block size can be selected to choose the trade-off between fine-granularity/small-buffer and coarse-granularity/large-buffer. The size of allocated buffer is a part of the endpoint descriptor and it is normally defined during the enumeration process according to its maxPacketSize parameter value (See “Universal Serial Bus Specification”).

Bit 15 **BL_SIZE: BLock SIZE.**

This bit selects the size of memory block used to define the allocated buffer area.

- If BL_SIZE=0, the memory block is 2 byte large, which is the minimum block allowed in a word-wide memory. With this block size the allocated buffer size ranges from 2 to 62 bytes.
- If BL_SIZE=1, the memory block is 32 byte large, which allows to reach the maximum packet length defined by USB specifications. With this block size the allocated buffer size ranges from 32 to 1024 bytes, which is the longest packet size allowed by USB standard specifications.

Bits 14:10 **NUM_BLOCK[4:0]: Number of blocks.**

These bits define the number of memory blocks allocated to this packet buffer. The actual amount of allocated memory depends on the BL_SIZE value as illustrated in [Table 63](#).

Bits 9:0 **COUNTn_RX[9:0]: Reception Byte Count**

These bits contain the number of bytes received by the endpoint associated with the USB_EPnR register during the last OUT/SETUP transaction addressed to it.

Note: Double-buffered and Isochronous IN Endpoints have two USB_COUNTn_TX registers: named USB_COUNTn_TX_1 and USB_COUNTn_TX_0 with the following content.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BLSIZE _1	NUM_BLOCK_1[4:0]						COUNTn_RX_1[9:0]									
rw	rw	rw	rw	rw	rw	r	r	r	r	r	r	r	r	r	r	r
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BLSIZE _0	NUM_BLOCK_0[4:0]						COUNTn_RX_0[9:0]									
rw	rw	rw	rw	rw	rw	r	r	r	r	r	r	r	r	r	r	r

Table 63. Definition of allocated buffer memory

Value of NUM_BLOCK[4:0]	Memory allocated when BL_SIZE=0	Memory allocated when BL_SIZE=1
0 ('00000')	Not allowed	32 bytes
1 ('00001')	2 bytes	64 bytes
2 ('00010')	4 bytes	96 bytes
3 ('00011')	6 bytes	128 bytes
...
15 ('01111')	30 bytes	512 bytes
16 ('10000')	32 bytes	N/A
17 ('10001')	34 bytes	N/A
18 ('10010')	36 bytes	N/A
...
30 ('11110')	60 bytes	N/A
31 ('11111')	62 bytes	N/A

17.7 USB Register map

Table 64. USB register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
0x00	USB_EP0R																																				
	Reset value																																				
0x04	USB_EP1R																																				
	Reset value																																				
0x08	USB_EP2R																																				
	Reset value																																				
0x0C	USB_EP3R																																				
	Reset value																																				
0x10	USB_EP4R																																				
	Reset value																																				
0x14	USB_EP5R																																				
	Reset value																																				
0x18	USB_EP6R																																				
	Reset value																																				
0x1C	USB_EP7R																																				
	Reset value																																				
0x20-0x3F																																					
0x40	USB_CNTR																	CTR	0	CTRM	0	PMAOVRM	0	RESERVED	0	RESUME	0	FSUSP	0	LPMODE	0	PDWN	1	FRES	1		
	Reset value																																				
0x44	USB_ISTR																																				
	Reset value																																				
0x48	USB_FNR																	RXDP	0	CTRM	0	PMAOVRM	0	RESERVED	0	RESUME	0	FSUSP	0	LPMODE	0	PDWN	1	FRES	1		
	Reset value																	RXDM	0	CTRM	0	PMAOVRM	0	RESERVED	0	RESUME	0	FSUSP	0	LPMODE	0	PDWN	1	FRES	1		
0x4C	USB_DADDR																	LCK	0	ERR	0	WKUPM	0	RESERVED	0	RESUME	0	FSUSP	0	LPMODE	0	PDWN	1	FRES	1		
	Reset value																		0	0	0	0	SUSP	0	RESUME	0	FSUSP	0	LPMODE	0	PDWN	1	FRES	1			
0x50	USB_BTABLE																																				
	Reset value																																				

Note: Refer to Table 1 on page 27 for the register boundary addresses.

18 Serial peripheral interface (SPI)

18.1 Introduction

The serial peripheral interface (SPI) allows half/ full-duplex, synchronous, serial communication with external devices. The interface can be configured as the master and in this case it provides the communication clock (SCK) to the external slave device. The interface is also capable of operating in multimaster configuration.

It may be used for a variety of purposes, including Simplex synchronous transfers on two lines with a possible bidirectional data line or reliable communication using CRC checking.

18.2 Main features

18.2.1 SPI features

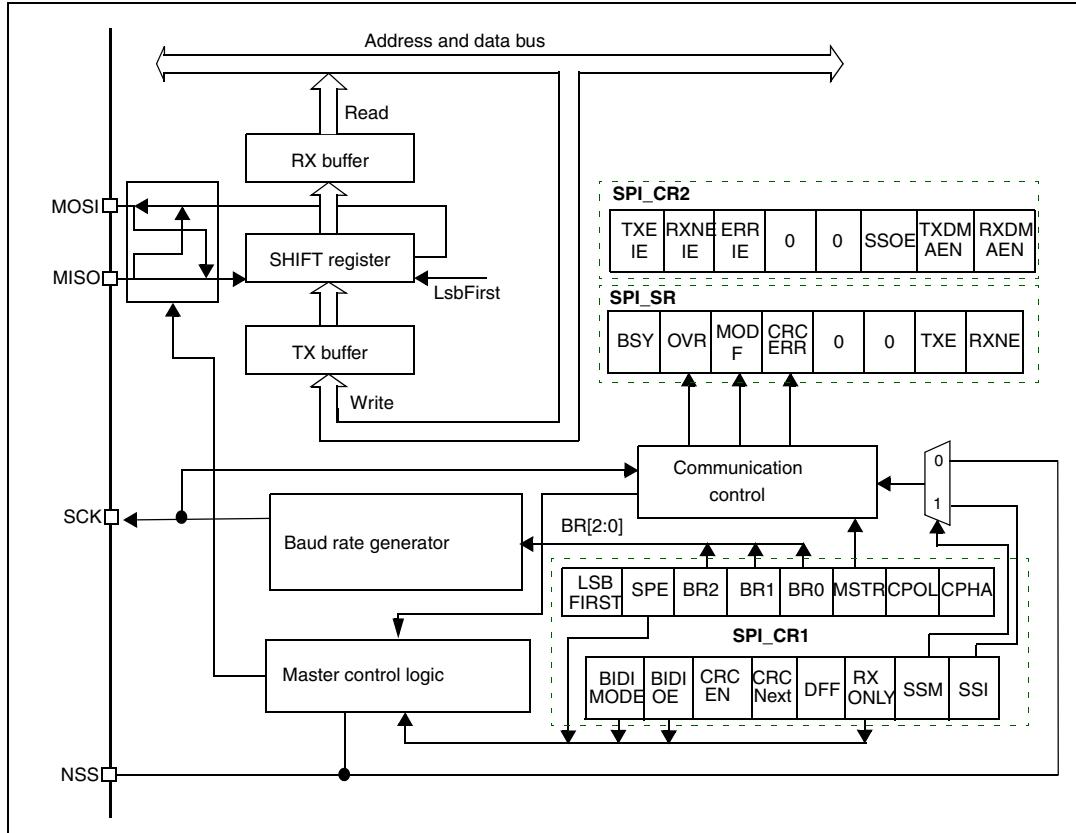
- Full-duplex synchronous transfers on three lines
- Simplex synchronous transfers on two lines with or without a bidirectional data line
- 8- or 16-bit transfer frame format selection
- Master or slave operation
- Multimaster mode capability
- 8 master mode baud rate prescalers ($f_{PCLK}/2$ max.)
- Slave mode frequency ($f_{PCLK}/2$ max)
- Faster communication for both master and slave: maximum SPI speed up to 18 MHz
- NSS management by hardware or software for both master and slave: dynamic change of master/slave operations
- Programmable clock polarity and phase
- Programmable data order with MSB-first or LSB-first shifting
- Dedicated transmission and reception flags with interrupt capability
- SPI bus busy status flag
- Hardware CRC feature for reliable communication:
 - CRC value can be transmitted as last byte in Tx mode
 - Automatic CRC error checking for last received byte
- Master mode fault, overrun and CRC error flags with interrupt capability
- 1-byte transmission and reception buffer with DMA capability: Tx and Rx requests

18.3 SPI functional description

18.3.1 General description

The block diagram of the SPI is shown in [Figure 161](#).

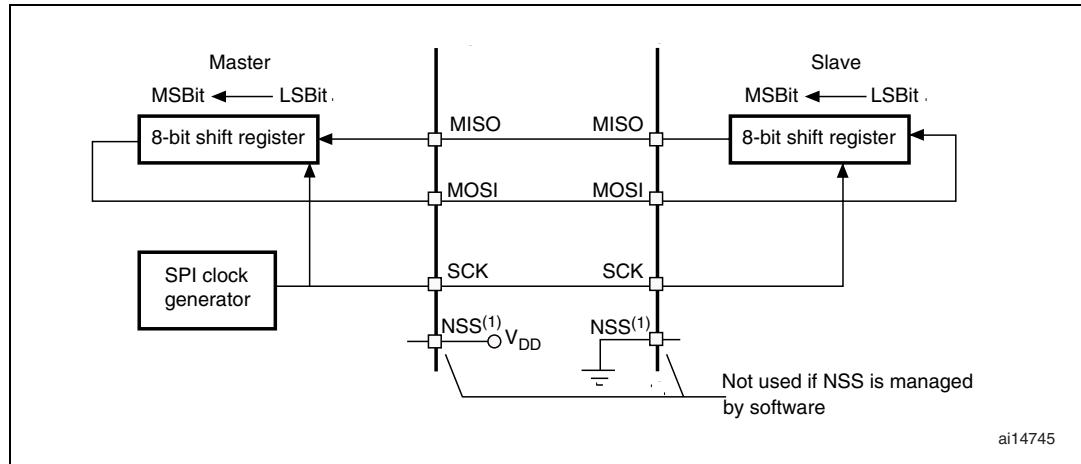
Figure 161. SPI block diagram



Usually, the SPI is connected to external devices through 4 pins:

- MISO: Master In / Slave Out data. This pin can be used to transmit data in slave mode and receive data in master mode.
- MOSI: Master Out / Slave In data. This pin can be used to transmit data in master mode and receive data in slave mode.
- SCK: Serial Clock out by SPI masters and input by SPI slaves.
- NSS: Slave select. This is an optional pin to select master/ slave mode. This pin acts as a ‘chip select’ to let the SPI master communicate with slaves individually and to avoid contention on the data lines. Slave NSS inputs can be driven by standard I/O ports on the master Device. The NSS pin may also be used as an output if enabled (SSOE bit) and driven low if the SPI is in master configuration. In this manner, all NSS pins from devices connected to the Master NSS pin see a low level and become slaves when they are configured in NSS hardware mode.

A basic example of interconnections between a single master and a single slave is illustrated in [Figure 162](#).

Figure 162. Single master/ single slave application

1. Here, the NSS pin is configured as an input.

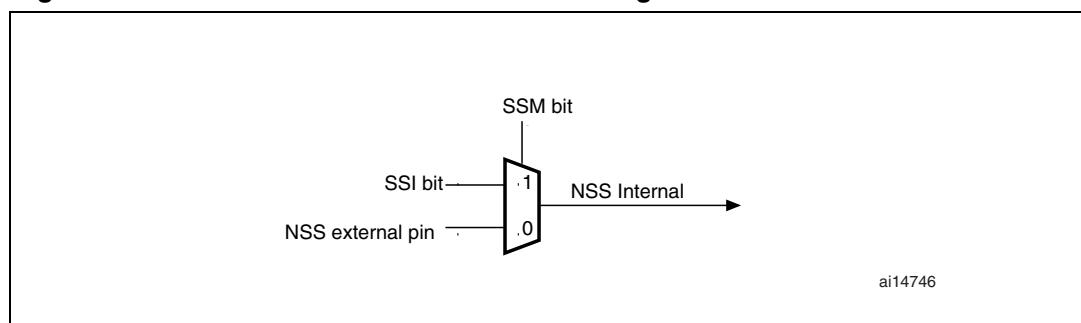
The MOSI pins are connected together and the MISO pins are connected together. In this way data is transferred serially between master and slave (most significant bit first).

The communication is always initiated by the master. When the master device transmits data to a slave device via the MOSI pin, the slave device responds via the MISO pin. This implies full-duplex communication with both data out and data in synchronized with the same clock signal (which is provided by the master device via the SCK pin).

Slave select (NSS) pin management

The NSS pin can be used for both input (in hardware mode) and output. SS output is enabled or disabled through the SSOE bit in the SPI_CR2 register. Multimaster configurations are only possible when SS output is disabled. The NSS pin is driven low when used as an output (SSOE bit) and the SPI is in master mode configuration. Thus all NSS pins in SPI devices become slaves when they are configured in NSS mode.

As an alternative to using the NSS pin to control the Slave Select signal (NSS pin), the application can choose to manage the Slave Select signal by software. This is configured by the SSM bit in the SPI_CR1 register (see [Figure 163](#)). In software management, the external NSS pin is free for other application uses and the internal NSS signal level is driven by writing to the SSI bit in the SPI_CR1 register.

Figure 163. Hardware/software slave select management

Clock phase and clock polarity

Four possible timing relationships may be chosen by software, using the CPOL and CPHA bits in the SPI_CR1 register. The CPOL (clock polarity) bit controls the steady state value of the clock when no data is being transferred. This bit affects both master and slave modes. If CPOL is reset, the SCK pin has a low-level idle state. If CPOL is set, the SCK pin has a high-level idle state.

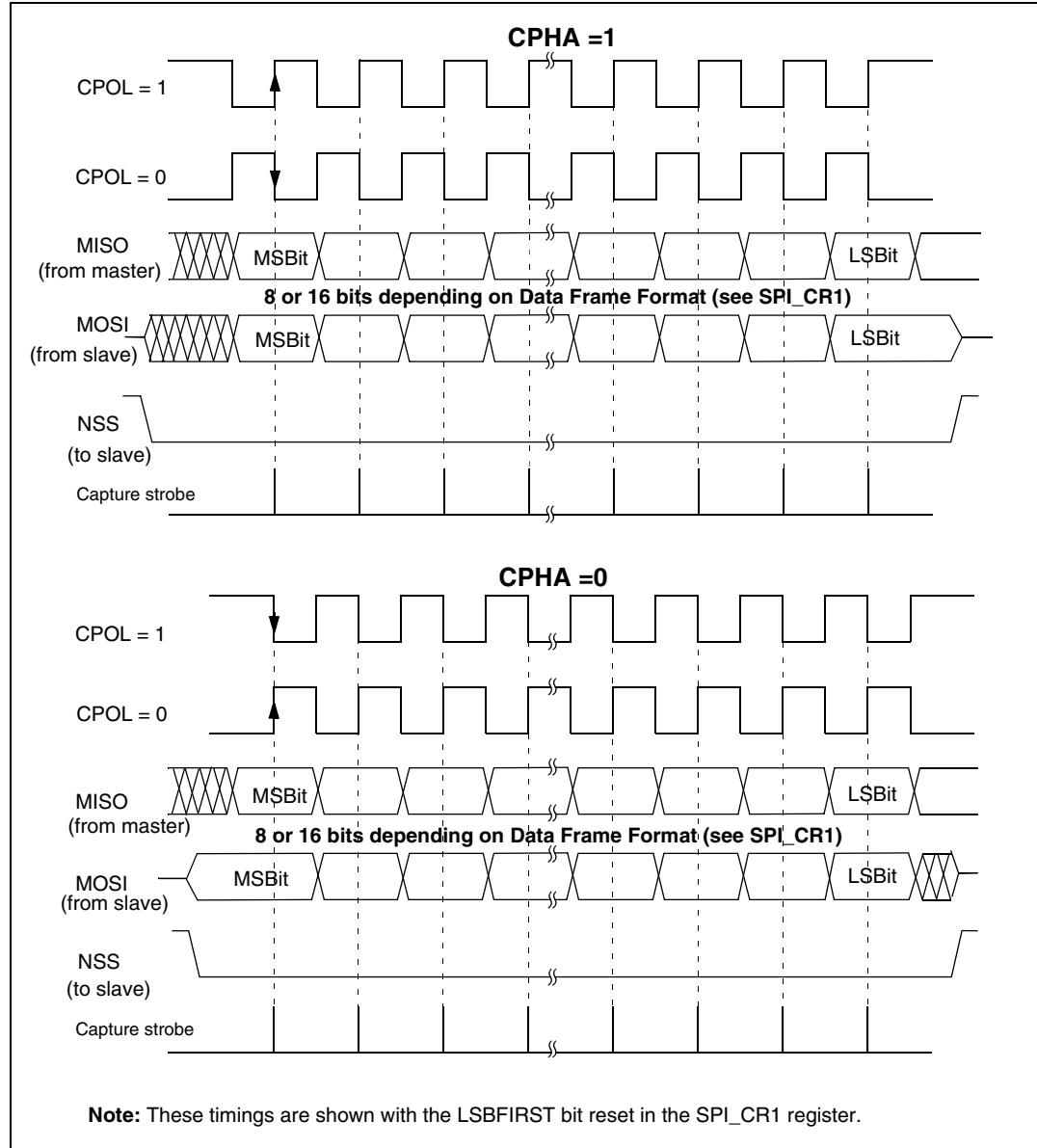
If the CPHA (clock phase) bit is set, the second edge on the SCK pin (falling edge if the CPOL bit is reset, rising edge if the CPOL bit is set) is the MSBit capture strobe. Data are latched on the occurrence of the first clock transition. If the CPHA bit is reset, the first edge on the SCK pin (falling edge if CPOL bit is set, rising edge if CPOL bit is reset) is the MSBit capture strobe. Data is latched on the occurrence of the second clock transition.

The combination of the CPOL (clock polarity) and CPHA (clock phase) bits selects the data capture clock edge.

Figure 164, shows an SPI transfer with the four combinations of the CPHA and CPOL bits. The diagram may be interpreted as a master or slave timing diagram where the SCK pin, the MISO pin, the MOSI pin are directly connected between the master and the slave device.

- Note:*
- 1 *Prior to changing the CPOL/CPHA bits the SPI must be disabled by resetting the SPE bit.*
 - 2 *Master and slave must be programmed with the same timing mode.*
 - 3 *The idle state of SCK must correspond to the polarity selected in the SPI_CR1 register (by pulling up SCK if CPOL=1 or pulling down SCK if CPOL=0).*
 - 4 *The Data Frame Format (8- or 16-bit) is selected through the DFF bit in SPI_CR1 register, and determines the data length during transmission/reception.*

Figure 164. Data clock timing diagram



Data frame format

Data can be shifted out either MSB-first or LSB-first depending on the value of the LSBFIRST bit in the SPI_CR1 Register.

Each data frame is 8 or 16 bits long depending on the size of the data programmed using the DFF bit in the SPI_CR1 register. The selected data frame format is applicable for transmission and/or reception.

18.3.2 SPI slave mode

In slave configuration, the serial clock is received on the SCK pin from the master device. The value set in the BR[2:0] bits in the SPI_CR1 register, does not affect the data transfer rate.

Procedure

1. Set the DFF bit to define 8- or 16-bit data frame format
2. Select the CPOL and CPHA bits to define one of the four relationships between the data transfer and the serial clock (see [Figure 164](#)). For correct data transfer, the CPOL and CPHA bits must be configured in the same way in the slave device and the master device.
3. The frame format (MSB-first or LSB-first depending on the value of the LSBFIRST bit in the SPI_CR1 register) must be the same as the master device.
4. In Hardware mode (refer to [Slave select \(NSS\) pin management on page 416](#)), the NSS pin must be connected to a low level signal during the complete byte transmit sequence. In Software mode, set the SSM bit and clear the SSI bit in the SPI_CR1 register.
5. Clear the MSTR bit and set the SPE bit (both in the SPI_CR1 register) to assign the pins to alternate functions.

In this configuration the MOSI pin is a data input and the MISO pin is a data output.

Transmit sequence

The data byte is parallel-loaded into the Tx buffer during a write cycle.

The transmit sequence begins when the slave device receives the clock signal and the most significant bit of the data on its MOSI pin. The remaining bits (the 7 bits in 8-bit data frame format, and the 15 bits in 16-bit data frame format) are loaded into the shift-register. The TXE flag in the SPI_SR register is set on the transfer of data from the Tx Buffer to the shift register and an interrupt is generated if the TXEIE bit in the SPI_CR2 register is set.

For the receiver, when data transfer is complete:

- The Data in shift register is transferred to Rx Buffer and the RXNE flag (SPI_SR register) is set
- An Interrupt is generated if the RXEIE bit is set in the SPI_CR2 register.

After the last sampling clock edge the RXNE bit is set, a copy of the data byte received in the shift register is moved to the Rx buffer. When the SPI_DR register is read, the SPI peripheral returns this buffered value.

Clearing of the RXNE bit is performed by reading the SPI_DR register.

18.3.3 SPI master mode

In the master configuration, the serial clock is generated on the SCK pin.

Procedure

1. Select the BR[2:0] bits to define the serial clock baud rate (see SPI_CR1 register).
2. Select the CPOL and CPHA bits to define one of the four relationships between the data transfer and the serial clock (see [Figure 164](#)).
3. Set the DFF bit to define 8- or 16-bit data frame format
4. Configure the LSBFIRST bit in the SPI_CR1 register to define the frame format
5. If the NSS pin is required in input mode, in hardware mode, connect the NSS pin to a high-level signal during the complete byte transmit sequence. In software mode, set the

- SSM and SSI bits in the SPI_CR1 register.
If the NSS pin is required in output mode, the SSOE bit only should be set.
6. The MSTR and SPE bits must be set (they remain set only if the NSS pin is connected to a high-level signal).

In this configuration the MOSI pin is a data output and the MISO pin is a data input.

Transmit sequence

The transmit sequence begins when a byte is written in the Tx Buffer.

The data byte is parallel-loaded into the shift register (from the internal bus) during the first bit transmission and then shifted out serially to the MOSI pin MSB first or LSB first depending on the LSBFIRST bit in the SPI_CR1 register. The TXE flag is set on the transfer of data from the Tx Buffer to the shift register and an interrupt is generated if the TXEIE bit in the SPI_CR2 register is set.

For the receiver, when data transfer is complete:

- The data in the shift register is transferred to the RX Buffer and the RXNE flag is set
- An interrupt is generated if the RXNEIE bit is set in the SPI_CR2 register

At the last sampling clock edge the RXNE bit is set, a copy of the data byte received in the shift register is moved to the Rx buffer. When the SPI_DR register is read, the SPI peripheral returns this buffered value.

Clearing the RXNE bit is performed by reading the SPI_DR register.

A continuous transmit stream can be maintained if the next data to be transmitted is put in the Tx buffer once the transmission is started. Note that TXE flag should be '1' before any attempt to write the Tx buffer is made.

18.3.4 Simplex communication

The SPI is capable of operating in simplex mode in 2 configurations.

- 1 clock and 1 bidirectional data wire
- 1 clock and 1 data wire (receive-only in full-duplex mode)

1 clock and 1 bidirectional data wire

This mode is enabled by setting the BIDIMODE bit in the SPI_CR1 register. In this mode SCK is used for the clock and MOSI in master or MISO in slave mode is used for data communication. The transfer direction (Input/Output) is selected by the BIDIOE bit in the SPI_CR2 register. When this bit is 1, the data line is output otherwise it is input.

1 clock and 1 data wire (receive-only in full-duplex mode)

In order to free an I/O pin so it can be used for other purposes, it is possible to disable the SPI output function by setting the RXONLY bit in the SPI_CR1 register. In this case, SPI will function in Receive-only mode. When the RXONLY bit is reset, the SPI will function in full duplex mode.

To start the communication in receive-only mode, it is necessary to enable the SPI. In the master mode, the communication starts immediately and will stop when the SPE bit is reset and the current reception terminates. In slave mode, the SPI will continue to receive as long as the NSS is pulled down (or the SSI bit is reset) and the SCK is running.

Note: The SPI can be used in Tx-only mode when the RXONLY bit in the SPI_CR1 register is reset, the RX pin (MISO in master or MOSI in slave) can be used as GPIO. In this case, when the data register is read, it does not contain the received value.

18.3.5 Status flags

Three status flags are provided for the application to completely monitor the state of the SPI bus.

BUSY flag

This flag indicates the state of the SPI communication layer. When it is set, it indicates that the SPI is busy communicating and/or there is a valid data byte in the Tx buffer waiting to be transmitted. The purpose of this flag is to indicate if there is any communication ongoing on the SPI bus or not. This flag is set as soon as:

1. Data is written in the SPI_DR register in master mode
2. The SCK clock is present in slave mode

The BUSY flag is reset each time a byte is transmitted/received. This flag is set and reset by hardware. It can be monitored to avoid write collision errors. Writing to this flag has no effect. The BUSY flag is meaningful only when the SPE bit is set.

Note: In master receiver mode (1-line bidirectional), the BUSY flag must NOT be checked.

Tx buffer empty flag (TXE)

When it is set, this flag indicates that the Tx buffer is empty and the next data to be transmitted can be loaded into the buffer. The TXE flag is reset when the Tx buffer already contains data to be transmitted. This flag is reset when the SPI is disabled (SPE bit is reset).

Rx buffer not empty (RXNE)

When set, this flag indicates that there are valid received data in the Rx Buffer. It is reset when SPI Data register is read.

18.3.6 CRC calculation

A CRC calculator has been implemented for communication reliability in full-duplex mode only. Separate CRC calculators are implemented for transmitted data and received data. The CRC is calculated using a programmable polynomial serially on each bit. It is calculated on the sampling clock edge defined by the CPHA and CPOL bits in the SPI_CR1 register.

Note: This SPI offers two kinds of CRC calculation standard which depend directly on the data frame format selected for the transmission and/or reception: 8-bit data (CR8) and 16-bit data (CRC16-CCITT).

CRC calculation is enabled by setting the CRCE bit in the SPI_CR1 register. This action resets the CRC registers (SPI_RXCRCR and SPI_TXCRCR). When the CRCNEXT bit in SPI_CR1 is set, the SPI_TXCRCR value is transmitted at the end of the current byte transmission.

The CRCERR flag in the SPI_SR register is set if the value received in the shift register during the SPI_TXCRCR value transmission does not match the SPI_RXCRCR value.

If data are present in the TX buffer, the CRC value is transmitted only after the transmission of the data byte. During CRC transmission, the CRC calculator is switched off and the register value remains unchanged.

Note: *Please refer to the product specifications for availability of this feature.*

SPI communication using CRC is possible through the following procedure:

- Program the CPOL, CPHA, LSBFirst, BR, SSM, SSI and MSTR values
- Program the polynomial in the SPI_CRCPR register
- Enable the CRC calculation by setting the CRCEN bit in the SPI_CR1 register. This also clears the SPI_RXCRCR and SPI_TXCRCR registers
- Enable the SPI by setting the SPE bit in the SPI_CR1 register
- Start the communication and sustain the communication until all but one byte or half-word have been transmitted or received.
- On writing the last byte or half-word to the TX buffer, set the CRCNext bit in the SPI_CR1 register to indicate that after transmission of the last byte, the CRC should be transmitted. CRC calculation is frozen during the CRC transmission.
- After transmitting the last byte or half word, the SPI transmits the CRC. The CRCNEXT bit is reset. The CRC is also received and compared against the SPI_RXCRCR value. If the value does not match, the CRCERR flag in SPI_SR is set and an interrupt can be generated when the ERRIE bit in the SPI_CR2 register is set.

Note: *With high bit rate frequencies, the user must take care when transmitting CRC. As the number of used CPU cycles has to be as low as possible in the CRC transfer phase, the calling of software functions in the CRC transmission sequence is forbidden to avoid errors in the last data and CRC reception.*

For high bit rate frequencies, the DMA mode is advised to avoid degradation of SPI speed performance due to CPU accesses impacting the SPI bandwidth.

18.3.7 SPI communication using DMA (direct memory addressing)

To operate at its maximum speed, the SPI needs to be fed with the data for transmission and the data received on the Rx buffer should be read to avoid overrun. To facilitate the transfers, the SPI is implemented with a DMA facility with a simple request/acknowledge protocol. DMA access is requested when the enable bit in the SPI_CR2 register is enabled. There are separate requests for the Tx buffer and the Rx buffer.

Note: *For high bitrate frequencies, it is advised to use the DMA mode to avoid degradation of SPI speed performance due to CPU accesses impacting the SPI bandwidth.*

DMA capability with CRC

When SPI communication in full-duplex mode is enabled with the CRC communication and the DMA mode, the transmission and reception of the CRC bytes at the end of communication are done automatically.

At the end of data and CRC transfers, the CRCERR flag in SPI_SR is set if corruption occurs during the transfer.

18.3.8 Error flags

Master mode fault (MODF)

Master mode fault occurs when the master device has its NSS pin pulled low (in hardware mode) or SSI bit low (in software mode), this automatically sets the MODF bit. Master mode fault affects the SPI peripheral in the following ways:

- The MODF bit is set and an SPI interrupt is generated if the ERRIE bit is set.
- The SPE bit is reset. This blocks all output from the device and disables the SPI interface.
- The MSTR bit is reset, thus forcing the device into slave mode.

Use the following software sequence to clear the MODF bit:

1. Make a read or write access to the SPI_SR register while the MODF bit is set.
2. Then write to the SPI_CR1 register.

To avoid any multiple slave conflicts in a system comprising several MCUs, the NSS pin must be pulled high during the MODF bit clearing sequence. The SPE and MSTR bits can be restored to their original state during or after this clearing sequence.

As a security, hardware does not allow the setting of the SPE and MSTR bits while the MODF bit is set.

In a slave device the MODF bit cannot be set. However, in a multimaster configuration, the device can be in slave mode with this MODF bit set. In this case, the MODF bit indicates that there might have been a multimaster conflict for system control. An interrupt routine can be used to recover cleanly from this state by performing a reset or returning to a default state.

Overrun condition

An overrun condition occurs when the master device has sent data bytes and the slave device has not cleared the RXNE bit resulting from the previous data byte transmitted. When an overrun condition occurs:

- OVR bit is set and an interrupt is generated if the ERRIE bit is set.

In this case, the receiver buffer contents will not be updated with the newly received data from the master device. A read to the SPI_DR register returns this byte. All other subsequently transmitted bytes are lost.

Clearing the OVR bit is done by a read of the SPI_DR register followed by a read access to the SPI_SR register.

CRC error

This flag is used to verify the validity of the value received when the CRCEN bit in the SPI_CR1 register is set. In full-duplex mode, the CRCERR flag in the SPI_SR register is set if the value received in the shift register (after transmission of the transmitter SPI_TXCRCR value) does not match the receiver SPI_RXCRCR value.

18.3.9 Disabling the SPI

When transfer is terminated, the application can stop the communication by disabling the SPI peripheral. This is done by resetting the SPE bit. Disabling the SPI peripheral while the last data transfer is still ongoing does not affect the data reliability if the device is *not* in Master transmit mode.

Note: In Master transmit mode (full-duplex or simplex transmit only), the application must make sure that no data transfer is ongoing by checking the BSY flag in the SPI_SR register before disabling the SPI master.

18.3.10 Interrupts

Table 65. SPI interrupt requests

Interrupt event	Event flag	Enable Control bit
Transmit buffer empty flag	TXE	TXEIE
Receive buffer not empty flag	RXNE	RXNEIE
Master Mode fault event	MODF	ERRIE
Overrun error	OVR	
CRC error flag	CRCERR	

18.4 SPI register description

Refer to [Section 1.1 on page 23](#) for a list of abbreviations used in register descriptions.

18.4.1 SPI Control Register 1 (SPI_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	BIDI OE	CRC EN	CRC NEXT	DFF	RX ONLY	SSM	SSI	LSB FIRST	SPE	BR [2:0]	MSTR	CPOL	CPHA		

rw rw

Bit 15 **BIDIMODE:** Bidirectional data mode enable

- 0: 2-line unidirectional data mode selected
- 1: 1-line bidirectional data mode selected

Bit 14 **BIDIOE:** Output enable in bidirectional mode

This bit combined with BIDI mode bit selects the direction of transfer in bidirectional mode

- 0: Output disabled (receive-only mode)
- 1: Output enabled (transmit-only mode)

In master mode, the MOSI pin is used and in slave mode, MISO pin is used.

Bit 13 **CRCEN:** Hardware CRC calculation enable

- 0: CRC calculation disabled
- 1: CRC calculation Enabled

Notes:

- This bit should be written only when SPI is disabled (SPE = '0') for correct operation
- This bit is used in full-duplex mode only.

Bit 12 CRCNEXT: Transmit CRC next

- 0: Next transmit value is from Tx buffer
- 1: Next transmit value is from Tx CRC register

Notes:

This bit has to be written as soon as the last data is written into the SPI_DR register.

This bit is only used in full-duplex mode.

Bit 11 DFF: Data Frame Format

- 0: 8-bit data frame format is selected for transmission/reception
- 1: 16-bit data frame format is selected for transmission/reception

Note: This bit should be written only when SPI is disabled (SPE = '0') for correct operation

Bit 10 RXONLY: Receive only

This bit combined with BIDI mode bit selects the direction of transfer in 2 line unidirectional mode.
This bit is also useful in a multislave system in which this particular slave is not accessed, the output from the accessed slave is not corrupted.

- 0: Full duplex (Transmit and receive)
- 1: Output disabled (Receive only mode)

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

Bit 8 SSI: Internal slave select

This bit has effect only when SSM bit is set. The value of this bit is forced onto the NSS pin and the I/O value of the NSS pin is ignored.

Bit 7 LSBFIRST: Frame Format

- 0: MSB transmitted first
- 1: LSB transmitted first

Note: This bit should not be changed when the communication is ongoing.

Bit 6 SPE: SPI Enable

- 0: Peripheral disabled
- 1: Peripheral enabled

Bits 5:3 BR[2:0]: Baud Rate Control

- 000: $f_{PCLK}/2$
- 001: $f_{PCLK}/4$
- 010: $f_{PCLK}/8$
- 011: $f_{PCLK}/16$
- 100: $f_{PCLK}/32$
- 101: $f_{PCLK}/64$
- 110: $f_{PCLK}/128$
- 111: $f_{PCLK}/256$

Note: These bits should not be changed when the communication is ongoing.

Bit 2 MSTR: Master Selection

- 0: Slave configuration
- 1: Master configuration

Note: This bit should not be changed when the communication is ongoing.

Bit1 CPOL: Clock Polarity

- 0: SCK to 0 when idle
- 1: SCK to 1 when idle

Note: This bit should not be changed when the communication is ongoing.

Bit0 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 the communication is ongoing.

18.4.2 SPI control register 2 (SPI_CR2)

Address offset: 0x04

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TXEIE	RXNE IE	ERRIE	Reserved	SSOE	TXDMA EN	RXDMA EN	
								rw	rw	rw		rw	rw	rw	

Bits 15:8 Reserved. Forced to 0 by hardware.

Bit 7 **TXEIE**: *Tx buffer empty interrupt enable*

0: TXE interrupt masked

1: TXE interrupt not masked. This allows a interrupt request to be generated when the TXE flag is set.

Note: To function correctly, the TXEIE and TXDMAEN bits should not be set at the same time.

Bit 6 **RXNEIE**: *RX buffer not empty interrupt enable*

0: RXNE interrupt masked

1: RXNE interrupt not masked. This allows a interrupt request to be generated when the RXNE flag is set.

Note: To function correctly, the TXEIE and TXDMAEN bits should not be set at the same time.

Bit 5 **ERRIE**: *Error interrupt enable*

This bit controls the generation of an interrupt when an error condition occurs (CRCERR, OVR, MODF).

0: Error interrupt is masked

1: Error interrupt is enabled.

Bits 4:3 Reserved. Forced to 0 by hardware.

Bit 2 **SSOE**: *SS Output Enable*

0: SS output is disabled in master mode and the cell can work in multimaster configuration

1: SS output is enabled in master mode and when the cell is enabled. The cell cannot work in a multimaster environment.

Bit 1 **TXDMAEN**: *Tx Buffer DMA Enable*

When this bit is set, the DMA request is made whenever the TXE flag is set.

0: Tx buffer DMA disabled

1: Tx buffer DMA enabled

Bit 0 **RXDMAEN**: *Rx Buffer DMA Enable*

When this bit is set, the DMA request is made whenever the RXNE flag is set.

0: Rx buffer DMA disabled

1: Rx buffer DMA enabled

18.4.3 SPI status register (SPI_SR)

Address offset: 0x08

Reset value: 0000 0010 (0x0002)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								BSY	OVR	MODF	CRC ERR	Reserved	TXE	RXNE	
		r		rc		rc		rc				r	r		

Bits 15:8 Reserved. Forced to 0 by hardware.

Bit 7 **BSY**: *Busy flag*

- 0: SPI not busy
 - 1: SPI is busy in communication or Tx buffer is not empty
- This flag is set and reset by hardware.

Bit 6 **OVR**: *Overrun flag*

- 0: No Overrun occurred
- 1: Overrun occurred

This flag is set by hardware and reset by a software sequence. Refer to [Section 18.3.8 on page 423](#) for 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 18.3.8 on page 423](#) for software sequence.

Bit 4 **CRCERR**: *CRC error flag*

- 0: CRC value received matches the SPI_RXCRCR value
 - 1: CRC value received does not match the SPI_RXCRCR value
- This flag is set by hardware and cleared by software writing 0.

Note:

This bit has a meaning in full-duplex mode only.

Bit 3:2 Reserved. Forced to 0 by hardware.

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

18.4.4 SPI data register (SPI_DR)

Address offset: 0x0C

Reset value: 0x0000

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

Bits 15:0 **DR[15:0]: Data register**

Data received or to be transmitted.

The data register is split into 2 buffers - one for writing (Transmit Buffer) and another one for reading (Receive buffer). A write to the data register will write into the Tx buffer and a read from the data register will return the value held in the Rx buffer.

Notes:

Depending on the data frame format selection bit (DFF in SPI_CR1 register), the data sent or received is either 8-bit or 16-bit. This selection has to be made before enabling SPI to ensure correct operation.

For an 8-bit data frame, the buffers are 8-bit and only the LSB of the register (SPI_DR[7:0]) is used for transmission/reception. When in reception mode, the MSB of the register (SPI_DR[15:8]) is forced to 0.

For a 16-bit data frame, the buffers are 16-bit and the entire register, SPI_DR[15:0] is used for transmission/reception.

SPI CRC polynomial register (SPI_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 (0007h) is the reset value of this register. Another polynomial can be configured as required.

18.4.5 SPI Rx CRC register (SPI_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 SPI_CR1 register is written to 1. The CRC is calculated serially using the polynomial programmed in the SPI_CRCPR register.

Only the 8 LSB bits are considered when the data frame format is set to be 8-bit data (DFF bit of SPI_CR1 is cleared). CRC calculation is done based on CRC8.

The entire 16-bits of this register are considered when a 16-bit data frame format is selected (DFF bit of the SPI_CR1 register is set). CRC calculation is done based on CRC16 - CCITT standard.

Note: A read to this register when the BSY Flag is set could return an incorrect value.

18.4.6 SPI Tx CRC register (SPI_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 SPI_CR1 is written to 1. The CRC is calculated serially using the polynomial programmed in the SPI_CRCPR register.

Only the 8 LSB bits are considered when the data frame format is set to be 8-bit data (DFF bit of SPI_CR1 is cleared). CRC calculation is done based on CRC8.

The entire 16-bits of this register are considered when a 16-bit data frame format is selected (DFF bit of the SPI_CR1 register is set). CRC calculation is done based on CRC16 - CCITT standard.

Note: A read to this register when the BSY flag is set could return an incorrect value.

18.5 SPI register map

Table 66. SPI register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
00h	SPI_CR1 Reset value																BIDIMODE	0	BIDIOE	0	CRCEN	0	CRCNEXT	0	DFF	0	RXOnly	0	SSM	0	SSI	0	BR [2:0]				
04h	SPI_CR2 Reset value																																				
08h	SPI_SR Reset value																																				
0Ch	SPI_DR Reset value																																				
10h	SPI_CRCPR Reset value																																				
14h	SPI_RXCRCR Reset value																																				
18h	SPI_TXCRCR Reset value																																				

Refer to [Table 1 on page 27](#) for the register boundary addresses.

19 Universal synchronous asynchronous receiver transmitter (USART)

19.1 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 fractional baud rate generator.

It supports synchronous one-way communication and half-duplex single wire communication. It also supports the LIN (local interconnection network), Smartcard Protocol and IrDA (infrared data association) SIR ENDEC specifications, and modem operations (CTS/RTS). It allows multiprocessor communication.

High speed data communication is possible by using the DMA for multibuffer configuration.

19.2 Main features

- Full duplex, asynchronous communications
- NRZ standard format (Mark/Space)
- Fractional baud rate generator systems
 - A common programmable transmit and receive baud rates up to 4.5 MBits/s
- Programmable data word length (8 or 9 bits)
- Configurable stop bits - support for 1 or 2 stop bits
- LIN Master Synchronous Break send capability and LIN slave break detection capability
 - 13-bit break generation and 10/11 bit break detection when USART is hardware configured for LIN
- Transmitter clock output for synchronous transmission
- IrDA SIR Encoder Decoder
 - Support for 3/16 bit duration for normal mode
- Smartcard Emulation Capability
 - The Smartcard interface supports the asynchronous protocol Smartcards as defined in ISO 7816-3 standards
 - 0.5, 1.5 Stop Bits for Smartcard operation
- Single wire Half Duplex Communication
- Configurable multibuffer communication using DMA (direct memory access)
 - Buffering of received/transmitted bytes in reserved SRAM using centralized DMA
- Separate enable bits for Transmitter and Receiver
- Transfer detection flags:
 - Receive buffer full
 - Transmit buffer empty
 - End of Transmission flags

- Parity control:
 - Transmits parity bit
 - Checks parity of received data byte
- Four error detection flags:
 - Overrun error
 - Noise error
 - Frame error
 - Parity error
- Ten interrupt sources with flags:
 - CTS changes
 - LIN break detection
 - Transmit data register empty
 - Transmission complete
 - Receive data register full
 - Idle line received
 - Overrun error
 - Framing error
 - Noise error
 - Parity error
- Multiprocessor communication - enter into mute mode if address match does not occur
- Wake up from mute mode (by idle line detection or address mark detection)
- Two receiver wakeup modes: Address bit (MSB, 9th bit), Idle line

19.3 General description

The interface is externally connected to another device by three pins (see [Figure 165](#)). Any USART bidirectional communication requires a minimum of two pins: Receive Data In (RX) and Transmit Data Out (TX):

RX: Receive Data Input is the serial data input. Oversampling techniques are used for data recovery by discriminating between valid incoming data and noise.

TX: Transmit Data Output. When the transmitter is disabled, the output pin returns to its I/O port configuration. When the transmitter is enabled and nothing is to be transmitted, the TX pin is at high level. In single-wire and smartcard modes, this I/O is used to transmit and receive the data (at USART level, data are then received on SW_RX).

Through these pins, serial data is transmitted and received in normal USART mode as frames comprising:

- An Idle Line prior to transmission or reception
- A start bit
- A data word (8 or 9 bits) least significant bit first
- 0.5, 1, 1.5, 2 Stop bits indicating that the frame is complete
- This interface uses a fractional baud rate generator - with a 12-bit mantissa and 4-bit fraction
- A status register (USART_SR)
- Data Register (USART_DR)
- A baud rate register (USART_BRR) - 12-bit mantissa and 4-bit fraction.
- A Guardtime Register (USART_GTPR) in case of Smartcard mode.

Refer to [Section 19.5: USART register description on page 459](#) for the definitions of each bit.

The following pin is required to interface in synchronous mode:

- **SCLK:** Transmitter clock output. This pin outputs the transmitter data clock for synchronous transmission corresponding to SPI master mode (no clock pulses on start bit and stop bit, and a software option to send a clock pulse on the last data bit). In parallel data can be received synchronously on RX. This can be used to control peripherals that have shift registers (e.g. LCD drivers). The clock phase and polarity are software programmable. In smartcard mode, SCLK can provide the clock to the smartcard.

The following pins are required to interface in IrDA mode:

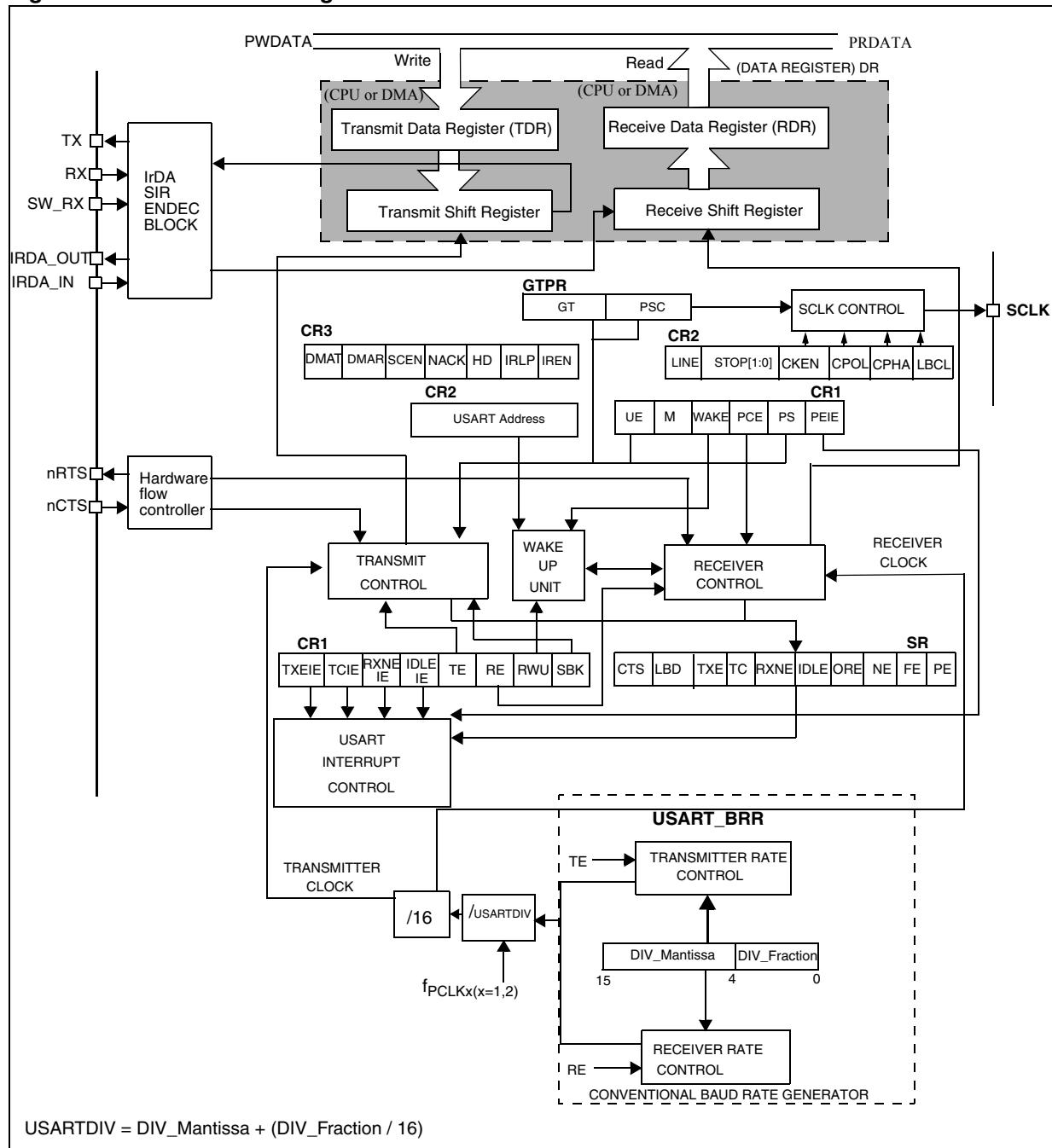
- **IrDA_RDI:** Receive Data Input is the data input in IrDA mode.
- **IrDA_TDO:** Transmit Data Output in IrDA mode.

The following pins are required in Hardware flow control mode:

- **nCTS:** Clear To Send blocks the data transmission at the end of the current transfer when high
- **nRTS:** Request to send indicates that the USART is ready to receive a data (when low).

19.3.1 Block diagram

Figure 165. USART block diagram



19.3.2 USART character description

Word length may be selected as being either 8 or 9 bits by programming the M bit in the USART_CR1 register (see [Figure 166](#)).

The TX pin is in low state during the start bit. It is in high state during the stop bit.

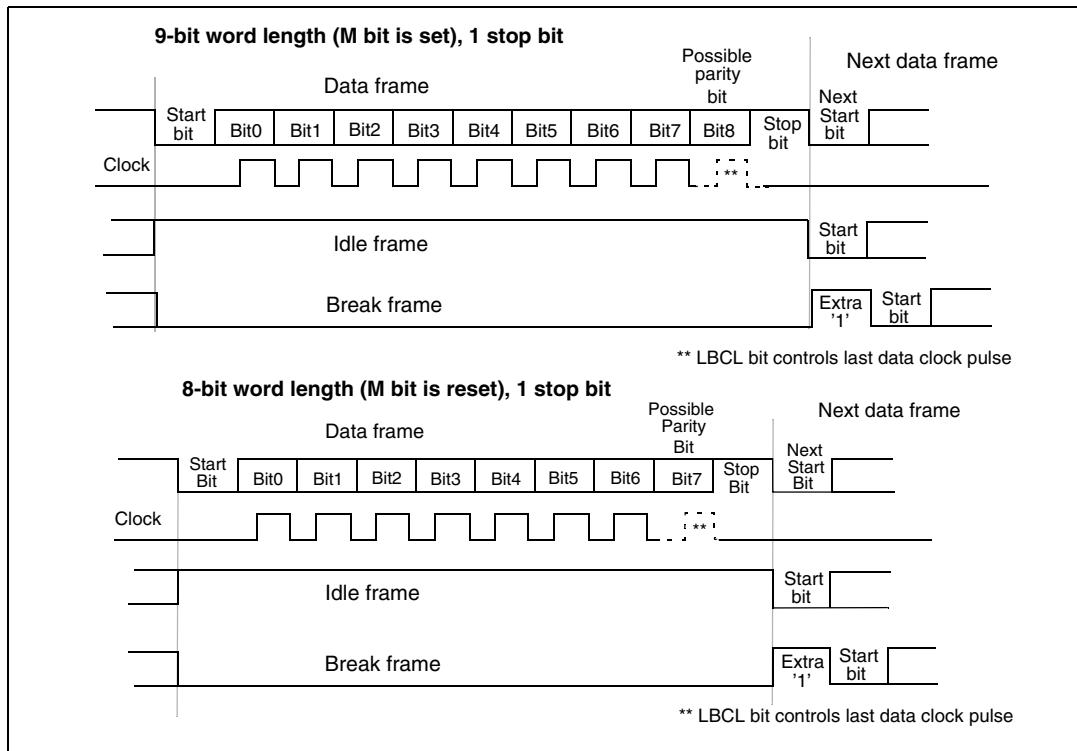
An **Idle character** is interpreted as an entire frame of “1”s followed by the start bit of the next frame which contains data (The number of “1” ‘s will include the number of stop bits).

A **Break character** is interpreted on receiving “0”s for a frame period. At the end of the break frame the transmitter inserts either 1 or 2 stop bits (logic “1” bit) to acknowledge the start bit.

Transmission and reception are driven by a common baud rate generator, the clock for each is generated when the enable bit is set respectively for the transmitter and receiver.

The details of each block is given below.

Figure 166. Word length programming



19.3.3 Transmitter

The transmitter can send data words of either 8 or 9 bits depending on the M bit status. When the transmit enable bit (TE) is set, the data in the transmit shift register is output on the TX pin and the corresponding clock pulses are output on the SCLK pin.

Character transmission

During an USART transmission, data shifts out least significant bit first on the TX pin. In this mode, the USART_DR register consists of a buffer (TDR) between the internal bus and the transmit shift register (see [Figure 165](#)).

Every character is preceded by a start bit which is a logic level low for one bit period. The character is terminated by a configurable number of stop bits.

The following stop bits are supported by USART.

- Note:
- 1 *The TE bit should not be reset during transmission of data. Resetting the TE bit during the transmission will corrupt the data on the TX pin as the baud rate counters will get frozen. The current data being transmitted will be lost.*
 - 2 *An idle frame will be sent after the TE bit is enabled.*

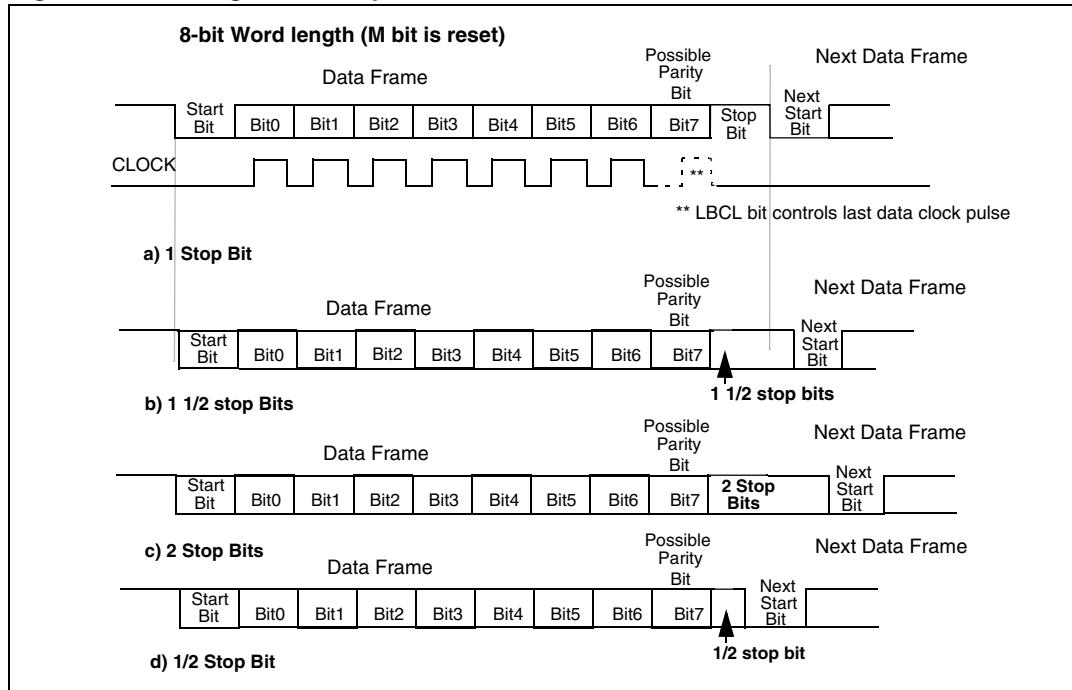
Configurable stop bits

The number of stop bits to be transmitted with every character can be programmed in Control register 2, bits 13,12.

1. **1 stop bit:** This is the default value of number of stop bits.
2. **2 Stop bits:** This will be supported by normal USART, single-wire and modem modes.
3. **0.5 stop bit:** To be used when receiving data in Smartcard mode.
4. **1.5 stop bits:** To be used when transmitting data in Smartcard mode.

An idle frame transmission will include the stop bits.

A break transmission will be 10 low bits followed by the configured number of stop bits (when m = 0) and 11 low bits followed by the configured number of stop bits (when m = 1). It is not possible to transmit long breaks (break of length greater than 10/11 low bits).

Figure 167. Configurable stop bits**Procedure:**

1. Enable the USART by writing the UE bit in USART_CR1 register to 1.
2. Program the M bit in USART_CR1 to define the word length.
3. Program the number of stop bits in USART_CR2.
4. Select DMA enable (DMAT) in USART_CR3 if Multi buffer Communication is to take place. Configure the DMA register as explained in multibuffer communication.
5. Set the TE bit in USART_CR1 to send an idle frame as first transmission.
6. Select the desired baud rate using the USART_BRR register.
7. Write the data to send in the USART_DR register (this clears the TXE bit). Repeat this for each data to be transmitted in case of single buffer.

Single byte communication

Clearing the TXE bit is always performed by a write to the data register.

The TXE bit is set by hardware and it indicates:

- The data has been moved from TDR to the shift register and the data transmission has started.
- The TDR register is empty.
- The next data can be written in the USART_DR register without overwriting the previous data.

This flag generates an interrupt if the TXEIE bit is set.

When a transmission is taking place, a write instruction to the USART_DR register stores the data in the TDR register and which is copied in the shift register at the end of the current transmission.

When no transmission is taking place, a write instruction to the USART_DR register places the data directly in the shift register, the data transmission starts, and the TXE bit is immediately set.

When a frame transmission is complete (after the stop bit) the TC bit is set and an interrupt is generated if the TCIE is set in the USART_CR1 register.

Clearing the TC bit is performed by the following software sequence:

1. A read to the USART_SR register
2. A write to the USART_DR register

Note: *The TC bit can also be cleared by writing a '0' to it. This clearing sequence is recommended only for Multibuffer communication.*

Break characters

Setting the SBK bit transmits a break character. The break frame length depends on the M bit (see [Figure 166](#)).

If the SBK bit is set to '1' a break character is sent on the TX line after completing the current character transmission. This bit is reset by hardware when the break character is completed (during the stop bit of the break character). The USART inserts a logic 1 bit at the end of the last break frame to guarantee the recognition of the start bit of the next frame.

Note: *If the software resets the SBK bit before the commencement of break transmission, the break character will not be transmitted. For two consecutive breaks, the SBK bit should be set after the stop bit of the previous break.*

Idle characters

Setting the TE bit drives the USART to send an idle frame before the first data frame.

19.3.4 Receiver

The USART can receive data words of either 8 or 9 bits depending on the M bit in the USART_CR1 register.

Character reception

During an USART reception, data shifts in least significant bit first through the RX pin. In this mode, the USART_DR register consists of a buffer (RDR) between the internal bus and the received shift register.

Procedure:

1. Enable the USART by writing the UE bit in USART_CR1 register to 1.
2. Program the M bit in USART_CR1 to define the word length.
3. Program the number of stop bits in USART_CR2.
4. Select DMA enable (DMAT) in USART_CR3 if multibuffer communication is to take place. Configure the DMA register as explained in multibuffer communication. STEP 3
5. Select the desired baud rate using the baud rate register USART_BRR
6. Set the RE bit USART_CR1. This enables the receiver which begins searching for a start bit.

When a character is received

- The RXNE bit is set. It indicates that the content of the shift register is transferred to the RDR. In other words, data has been received and can be read (as well as its associated error flags).
- An interrupt is generated if the RXNEIE bit is set.
- The error flags can be set if a frame error, noise or an overrun error has been detected during reception.
- In multibuffer, RXNE is set after every byte received and is cleared by the DMA read to the Data Register.
- In single buffer mode, clearing the RXNE bit is performed by a software read to the USART_DR register. The RXNE flag can also be cleared by writing a zero to it. The RXNE bit must be cleared before the end of the reception of the next character to avoid an overrun error.

Note: The RE bit should not be reset while receiving data. If the RE bit is disabled during reception, the reception of the current byte will be aborted.

Break character

When a break character is received, the USART handles it as a framing error.

Idle character

When an idle frame is detected, there is the same procedure as a data received character plus an interrupt if the IDLEIE bit is set.

Overrun error

An overrun error occurs when a character is received when RXNE has not been reset. Data can not be transferred from the shift register to the RDR register until the RXNE bit is cleared.

The RXNE flag is set after every byte received. An overrun error occurs if RXNE flag is set when the next data is received or the previous DMA request has not been serviced. When an overrun error occurs:

- The ORE bit is set.
- The RDR content will not be lost. The previous data is available when a read to USART_DR is performed.
- The shift register will be overwritten. After that point, any data received during overrun is lost.
- An interrupt is generated if either the RXNEIE bit is set or both the EIE and DMAR bits are set.
- The ORE bit is reset by a read to the USART_SR register followed by a USART_DR register read operation.

Note: The ORE bit, when set, indicates that at least 1 data has been lost. There are two possibilities:

- if RXNE=1, then the last valid data is stored in the receive register RDR and can be read,
- if RXNE=0, then it means that the last valid data has already been read and thus there is nothing to be read in the RDR. This case can occur when the last valid data is read in the RDR at the same time as the new (and lost) data is received. It may also occur

when the new data is received during the reading sequence (between the USART_SR register read access and the USART_DR read access).

Noise error

Over-sampling techniques are used (except in synchronous mode) for data recovery by discriminating between valid incoming data and noise.

Figure 168. Data sampling for noise detection

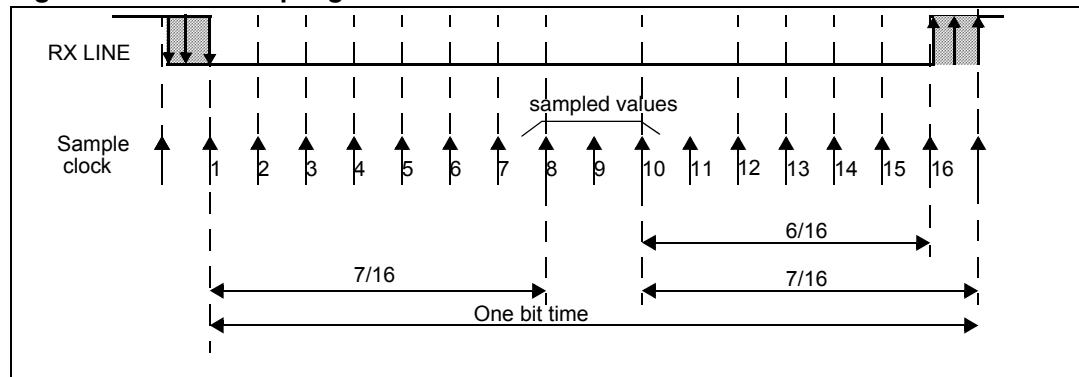


Table 67. Noise detection from sampled data

Sampled value	NE status	Received bit value	Data validity
000	0	0	Valid
001	1	0	Not Valid
010	1	0	Not Valid
011	1	1	Not Valid
100	1	0	Not Valid
101	1	1	Not Valid
110	1	1	Not Valid
111	0	1	Valid

When noise is detected in a frame:

- The NE is set at the rising edge of the RXNE bit.
- The invalid data is transferred from the Shift register to the USART_DR register.
- No interrupt is generated in case of single byte communication. However this bit rises at the same time as the RXNE bit which itself generates an interrupt. In case of multibuffer communication an interrupt will be issued if the EIE bit is set in the USART_CR3 register.

The NE bit is reset by a USART_SR register read operation followed by a USART_DR register read operation.

Framing error

A framing error is detected when:

The stop bit is not recognized on reception at the expected time, following either a de-synchronization or excessive noise.

When the framing error is detected:

- The FE bit is set by hardware
- The invalid data is transferred from the Shift register to the USART_DR register.
- No interrupt is generated in case of single byte communication. However this bit rises at the same time as the RXNE bit which itself generates an interrupt. In case of multibuffer communication an interrupt will be issued if the EIE bit is set in the USART_CR3 register.

The FE bit is reset by a USART_SR register read operation followed by a USART_DR register read operation.

Configurable stop bits during reception

The number of stop bits to be received can be configured through the control bits of Control Register 2 - it can be either 1 or 2 in normal mode and 0.5 or 1.5 in Smartcard mode.

1. **0.5 stop Bit (reception in Smartcard mode):** No sampling is done for 0.5 stop bit. As a consequence, no framing error and no break frame can be detected when 0.5 stop bit is selected.
2. **1 stop Bit:** Sampling for 1 stop Bit is done on the 8th, 9th and 10th samples.
3. **1.5 stop Bits (transmission in Smartcard mode):** When transmitting in smartcard mode, the device must check that the data is correctly sent. Thus the receiver block must be enabled (RE = 1 in the USART_CR1 register) and the stop bit is checked to test if the smartcard has detected a parity error. In the event of a parity error, the smartcard forces the data signal low during the sampling - NACK signal-, which is flagged as a framing error. Then, the FE flag is set with the RXNE at the end of the 1.5 stop bit. Sampling for 1.5 stop bits is done on the 16th, 17th and 18th samples (1 baud clock period after the beginning of the stop bit). The 1.5 stop bit can be decomposed into 2 parts: one 0.5 baud clock period during which nothing happens, followed by 1 normal stop bit period during which sampling occurs halfway through. Refer to [Section 19.3.11: Smartcard on page 451](#) for more details.
4. **2 stop Bits:** Sampling for 2 stop bits is done on the 8th, 9th and 10th samples of the first stop bit. If a framing error is detected during the first stop bit the framing error flag will be set. The second stop bit is not checked for framing error. The RXNE flag will be set at the end of the first stop bit.

19.3.5 Fractional baud rate generation

The baud rate for the receiver and transmitter (Rx and Tx) are both set to the same value as programmed in the Mantissa and Fraction values of USARTDIV.

$$\text{Tx/ Rx baud} = \frac{f_{CK}}{(16 * \text{USARTDIV})}$$

legend: $f_{PCLKx(x=1,2)}$ - Input clock to the peripheral (PCLK1 for USART2, 3, 4, 5 or PCLK2 for USART1)

USARTDIV is an unsigned fixed point number that is coded on the USART_BRR register.

How to derive USARTDIV from USART_BRR register values

Example 1:

If DIV_Mantissa = 27d and DIV_Fraction= 12d (USART_BRR=1BCh), then

Mantissa (USARTDIV) = 27d

Fraction (USARTDIV) = $12/16 = 0.75d$

Therefore USARTDIV = 27.75d

Example 2:

To program USARTDIV = 25.62d,

This leads to:

DIV_Fraction = $16 * 0.62d = 9.92d$, nearest real number 10d = 0xA

DIV_Mantissa = mantissa (25.620d) = 25d = 0x19

Then, USART_BRR = 0x19A

Example 3:

To program USARTDIV = 50.99d

This leads to:

DIV_Fraction = $16 * 0.99d = 15.84d \Rightarrow$ nearest real number, 16d = 0x10

DIV_Mantissa = mantissa (50.990d) = 50d = 0x32

Note: The Baud Counters will be updated with the new value of the Baud Registers after a write to USART_BRR. Hence the Baud Rate Register value should not be changed during a transaction.

Table 68. Error calculation for programmed baud rates

Baud rate		$f_{PCLK} = 36\text{ MHz}$			$f_{PCLK} = 72\text{ MHz}$		
S.No	in Kbps	Actual	Value programmed in the Baud Rate Register	% Error =(Calculated - Desired)B.Rate /Desired B.Rate	Actual	Value programmed in the Baud Rate Register	% Error
1.	2.4	2.400	937.5	0%	2.4	1875	0%
2.	9.6	9.600	234.375	0%	9.6	468.75	0%
3.	19.2	19.2	117.1875	0%	19.2	234.375	0%
4.	57.6	57.6	39.0625	0%	57.6	78.125	0.%
5.	115.2	115.384	19.5	0.15%	115.2	39.0625	0%
6.	230.4	230.769	9.75	0.16%	230.769	19.5	0.16%
7.	460.8	461.538	4.875	0.16%	461.538	9.75	0.16%
8.	921.6	923.076	2.4375	0.16%	923.076	4.875	0.16%
9.	2250	2250	1	0%	2250	2	0%
10.	4500	NA	NA	NA	4500	1	0%

- Note:**
- 1 The lower the CPU clock the lower will be the accuracy for a particular Baud rate. The upper limit of the achievable baud rate can be fixed with this data.
 - 2 Only USART1 is clocked with PCLK2 (72 MHz Max). Other USARTs are clocked with PCLK1 (36 MHz Max).

19.3.6 Multiprocessor communication

There is a possibility of performing multiprocessor communication with the USART (several USARTs connected in a network). For instance one of the USARTs can be the master, its TX output is connected to the RX input of the other USART. The others are slaves, their respective TX outputs are logically ANDed together and connected to the RX input of the master.

In multiprocessor configurations it is often desirable that only the intended message recipient should actively receive the full message contents, thus reducing redundant USART service overhead for all non addressed receivers.

The non addressed devices may be placed in mute mode by means of the muting function. In mute mode:

- None of the reception status bits can be set.
- All the receive interrupts are inhibited.
- The RWU bit in USART_CR1 register is set to 1. RWU can be controlled automatically by hardware or written by the software under certain conditions.

The USART can enter or exit from mute mode using one of two methods, depending on the WAKE bit in the USART_CR1 register:

- Idle Line detection if the WAKE bit is reset,
- Address Mark detection if the WAKE bit is set.

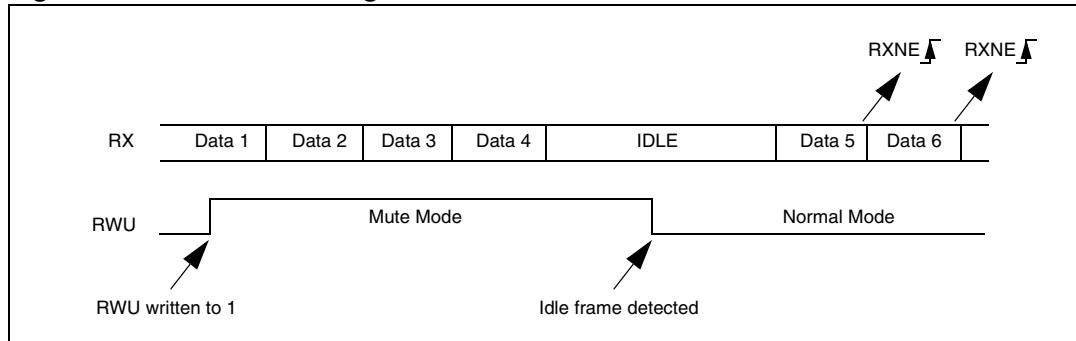
Idle line detection (WAKE=0)

The USART enters mute mode when the RWU bit is written to 1.

It wakes up when an Idle frame is detected. Then the RWU bit is cleared by hardware but the IDLE bit is not set in the USART_SR register. RWU can also be written to 0 by software.

An example of mute mode behavior using idle line detection is given in [Figure 169](#).

Figure 169. Mute mode using Idle line detection



Address mark detection (WAKE=1)

In this mode, bytes are recognized as addresses if their MSB is a '1' else they are considered as data. In an address byte, the address of the targeted receiver is put on the 4 LSB. This 4-bit word is compared by the receiver with its own address which is programmed in the ADD bits in the USART_CR2 register.

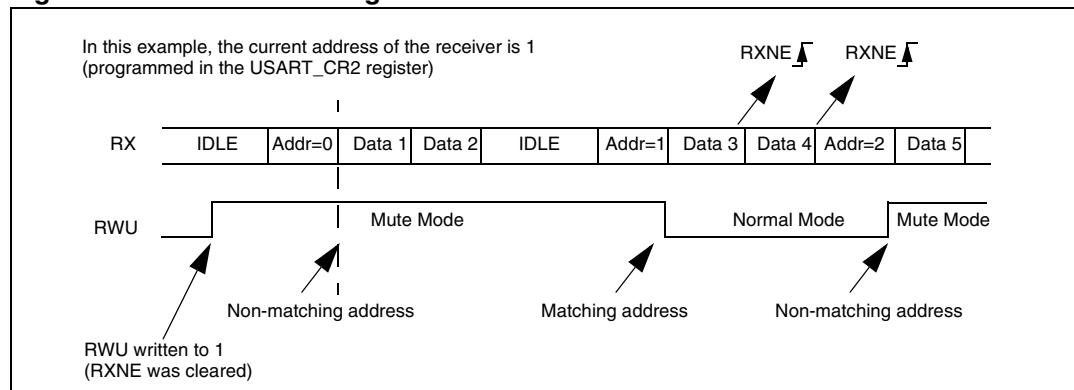
The USART enters mute mode when an address character is received which does not match its programmed address. The RXNE flag is not set for this address byte and no interrupt nor DMA request is issued as the USART would have entered mute mode.

It exits from mute mode when an address character is received which matches the programmed address. Then the RWU bit is cleared and subsequent bytes are received normally. The RXNE bit is set for the address character since the RWU bit has been cleared.

The RWU bit can be written to as 0 or 1 when the receiver buffer contains no data (RXNE=0 in the USART_SR register). Otherwise the write attempt is ignored.

An example of mute mode behavior using address mark detection is given in [Figure 170](#).

Figure 170. Mute mode using Address mark detection



19.3.7 Parity control

Parity control (generation of parity bit in transmission and parity checking in reception) can be enabled by setting the PCE bit in the USART_CR1 register. Depending on the frame length defined by the M bit, the possible USART frame formats are as listed in [Table 69](#).

Table 69. Frame formats

M bit	PCE bit	USART frame
0	0	SB 8 bit data STB
0	1	SB 7-bit data PB STB
1	0	SB 9-bit data STB
1	1	SB 8-bit data PB STB

Legends: SB: Start Bit, STB: Stop Bit, PB: Parity Bit

Note: *In case of wake up by an address mark, the MSB bit of the data is taken into account and not the parity bit*

Even parity: the parity bit is calculated to obtain an even number of “1s” inside the frame made of the 7 or 8 LSB bits (depending on whether M is equal to 0 or 1) and the parity bit.

Ex: data=00110101; 4 bits set => parity bit will be 0 if even parity is selected (PS bit in USART_CR1 = 0).

Odd parity: the parity bit is calculated to obtain an odd number of “1s” inside the frame made of the 7 or 8 LSB bits (depending on whether M is equal to 0 or 1) and the parity bit.

Ex: data=00110101; 4 bits set => parity bit will be 1 if odd parity is selected (PS bit in USART_CR1 = 1).

Transmission mode: If the PCE bit is set in USART_CR1, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit (even number of “1s” if even parity is selected (PS=0) or an odd number of “1s” if odd parity is selected (PS=1)). If the parity check fails, the PE flag is set in the USART_SR register and an interrupt is generated if PEIE is set in the USART_CR1 register.

19.3.8 LIN (local interconnection network) mode

The LIN mode is selected by setting the LINEN bit in the USART_CR2 register. In LIN mode, the following bits must be kept cleared:

- CLKEN in the USART_CR2 register,
- STOP[1:0], SCEN, HDSEL and IREN in the USART_CR3 register.

LIN transmission

The same procedure explained in [Section 19.3.3](#) has to be applied for LIN Master transmission than for normal USART transmission with the following differences:

- Clear the M bit to configure 8-bit word length.
- Set the LINEN bit to enter LIN mode. In this case, setting the SBK bit sends 13 ‘0’ bits as a break character. Then a bit of value ‘1’ is sent to allow the next start detection.

LIN reception

When the LIN mode is enabled, the break detection circuit is activated. The detection is totally independent from the normal USART receiver. A break can be detected whenever it occurs, during idle state or during a frame.

When the receiver is enabled (RE=1 in USART_CR1), the circuit looks at the RX input for a start signal. The method for detecting start bits is the same when searching break characters or data. After a start bit has been detected, the circuit samples the next bits exactly like for the data (on the 8th, 9th and 10th samples). If 10 (when the LBDL = 0 in USART_CR2) or 11 (when LBDL=1 in USART_CR2) consecutive bits are detected as ‘0’, and are followed by a delimiter character, the LBD flag is set in USART_SR. If the LBDIE bit=1, an interrupt is generated. Before validating the break, the delimiter is checked for as it signifies that the RX line has returned to a high level.

If a ‘1’ is sampled before the 10 or 11 have occurred, the break detection circuit cancels the current detection and searches for a start bit again.

If the LIN mode is disabled (LINEN=0), the receiver continues working as normal USART, without taking into account the break detection.

If the LIN mode is enabled (LINEN=1), as soon as a framing error occurs (i.e. stop bit detected at ‘0’, which will be the case for any break frame), the receiver stops until the break

detection circuit receives either a ‘1’, if the break word was not complete, or a delimiter character if a break has been detected.

The behavior of the break detector state machine and the break flag is shown on the [Figure 171: Break detection in LIN mode \(11-bit break length - LBDL bit is set\) on page 447](#).

Examples of break frames are given on [Figure 172: Break detection in LIN mode vs. Framing error detection on page 448](#).

Figure 171. Break detection in LIN mode (11-bit break length - LBDL bit is set)

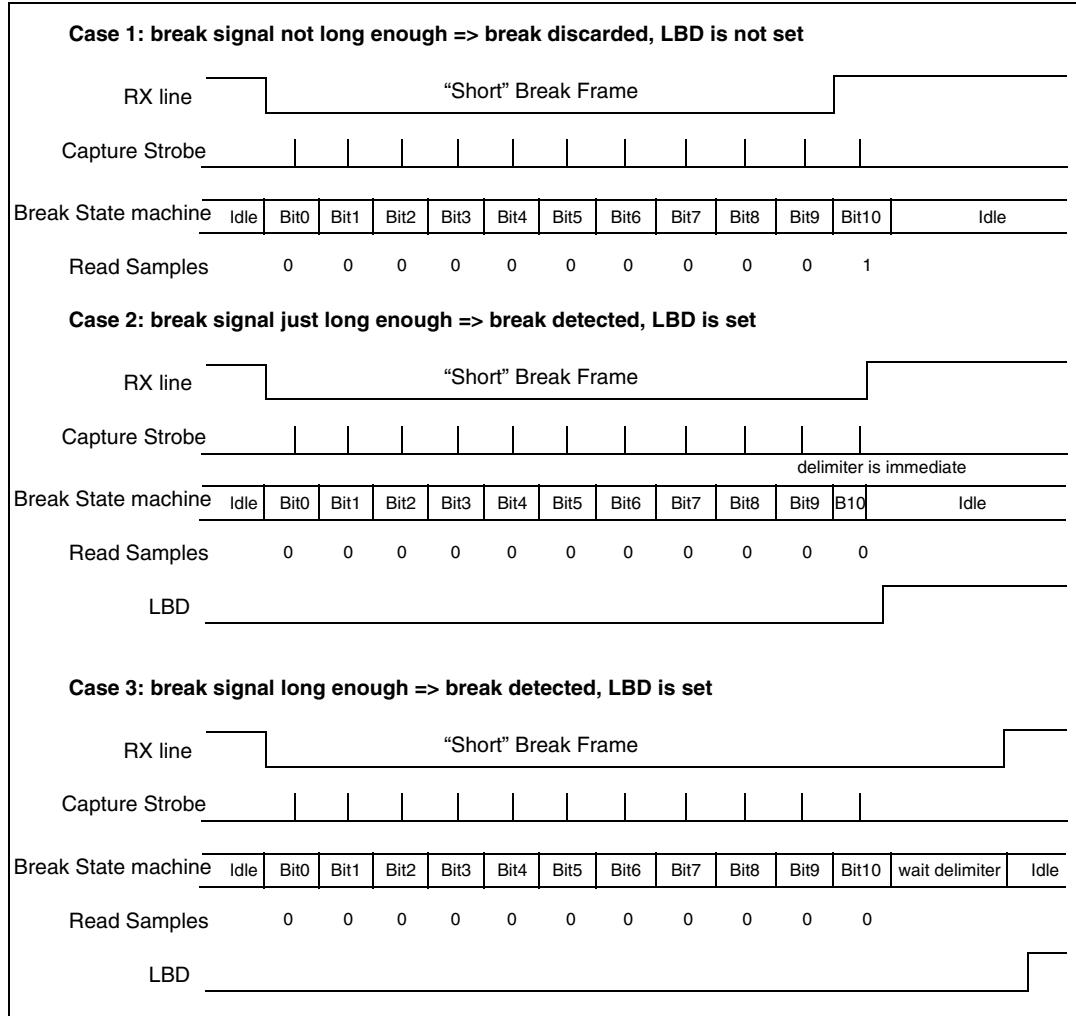
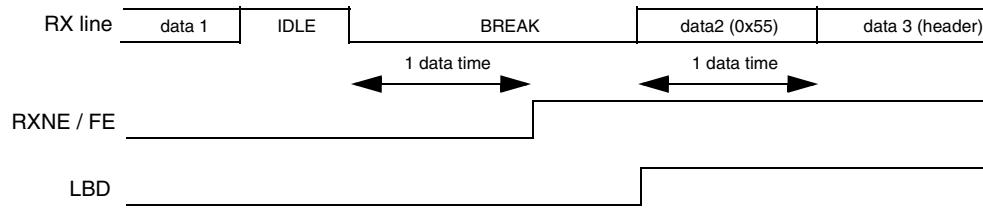


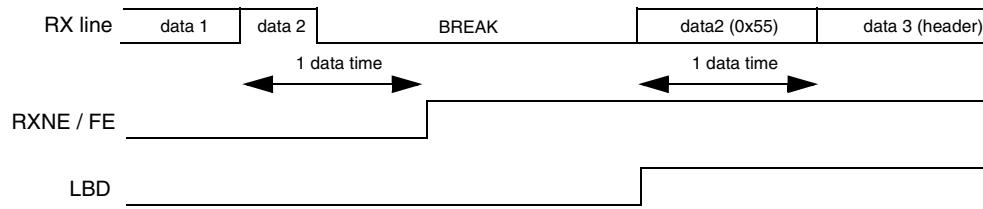
Figure 172. Break detection in LIN mode vs. Framing error detection

In these examples, we suppose that LBDL=1 (11-bit break length), M=0 (8-bit data)

Case 1: break occurring after an Idle



Case 1: break occurring while a data is being received



19.3.9 USART synchronous mode

The synchronous mode is selected by writing the CLKEN bit in the USART_CR2 register to 1. In synchronous mode, the following bits must be kept cleared:

- LINEN bit in the USART_CR2 register,
- SCEN, HDSEL and IREN bits in the USART_CR3 register.

The USART allows the user to control a bidirectional synchronous serial communications in master mode. The SCLK pin is the output of the USART transmitter clock. No clock pulses are sent to the SCLK pin during start bit and stop bit. Depending on the state of the LBCL bit in the USART_CR2 register clock pulses will or will not be generated during the last valid data bit (address mark). The CPOL bit in the USART_CR2 register allows the user to select the clock polarity, and the CPHA bit in the USART_CR2 register allows the user to select the phase of the external clock (see [Figure 173](#), [Figure 174](#) & [Figure 175](#)).

During idle, preamble and send break, the external SCLK clock is not activated.

In synchronous mode the USART transmitter works exactly like in asynchronous mode. But as SCLK is synchronized with TX (according to CPOL and CPHA), the data on TX is synchronous.

In this mode the USART receiver works in a different manner compared to the asynchronous mode. If RE=1, the data is sampled on SCLK (rising or falling edge, depending on CPOL and CPHA), without any oversampling. A setup and a hold time must be respected (which depends on the baud rate: 1/16 bit time).

Note: 1 The SCLK pin works in conjunction with the TX pin. Thus, the clock is provided only if the transmitter is enabled (TE=1) and a data is being transmitted (the data register USART_DR

(has been written). This means that it is not possible to receive a synchronous data without transmitting data.

- 2 *The LBCL, CPOL and CPHA bits have to be selected when both the transmitter and the receiver are disabled ($TE=RE=0$) to ensure that the clock pulses function correctly. These bits should not be changed while the transmitter or the receiver is enabled.*
- 3 *It is advised that TE and RE are set in the same instruction in order to minimize the setup and the hold time of the receiver.*
- 4 *The USART supports master mode only: it cannot receive or send data related to an input clock (SCLK is always an output).*

Figure 173. USART example of synchronous transmission

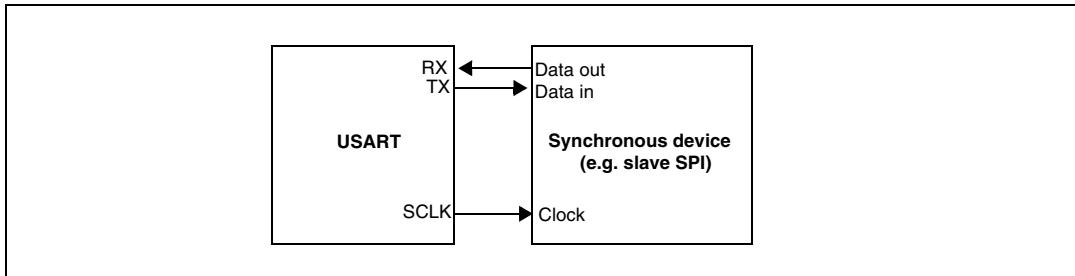


Figure 174. USART data clock timing diagram (M=0)

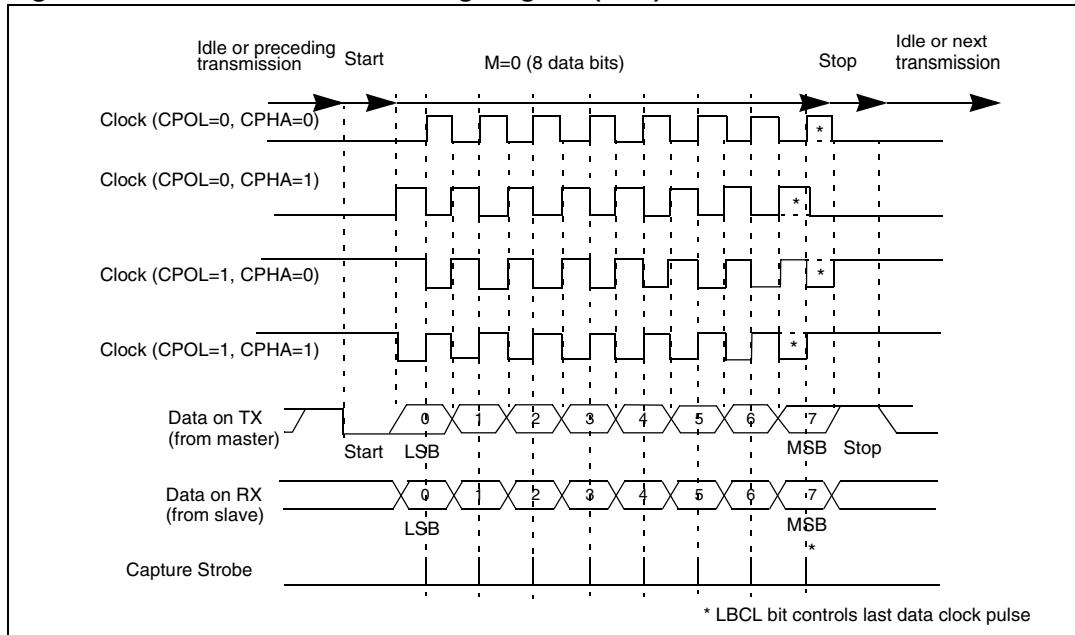
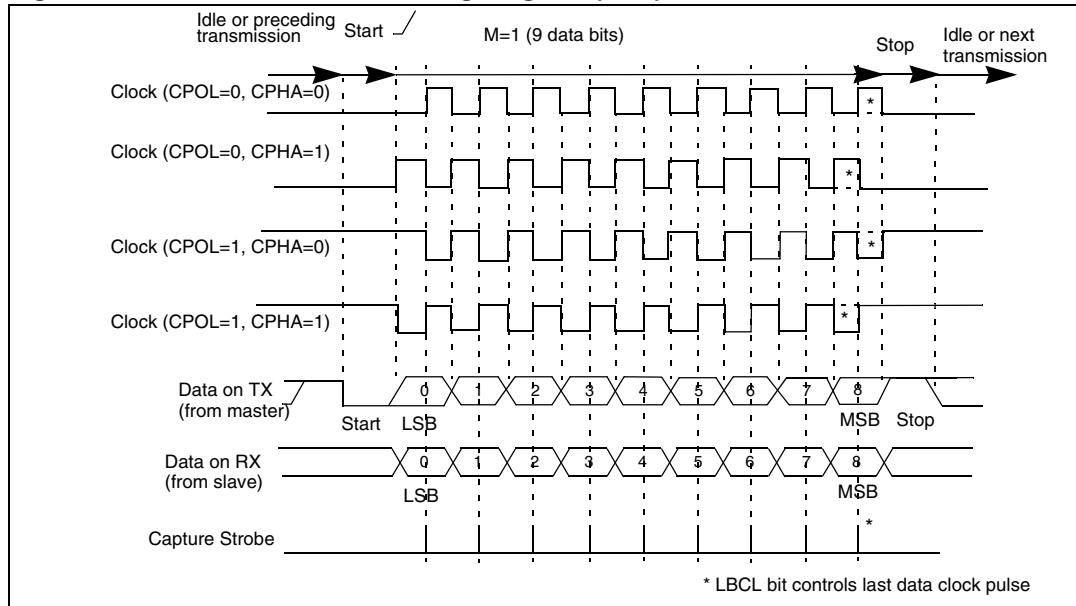
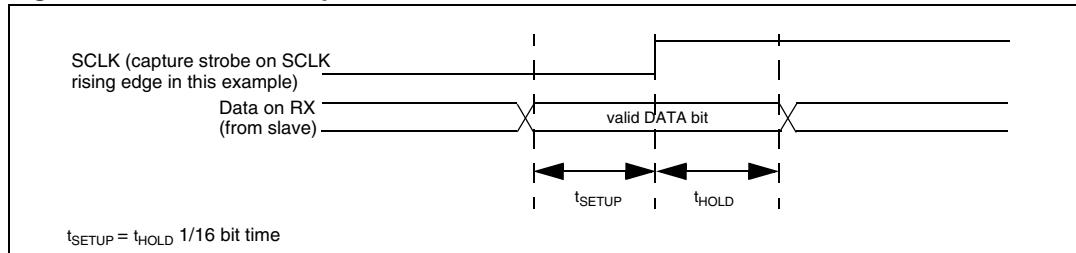


Figure 175. USART data clock timing diagram (M=1)**Figure 176. RX data setup/hold time**

Note:

The function of SCLK is different in Smartcard mode. Refer to the Smartcard mode chapter for more details.

19.3.10 Single wire half duplex communication

The single-wire half-duplex mode is selected by setting the HDSEL bit in the USART_CR3 register. In this mode, the following bits must be kept cleared:

- LINEN and CLKEN bits in the USART_CR2 register,
- SCEN and IREN bits in the USART_CR3 register.

The USART can be configured to follow a single wire half duplex protocol. The selection between half and full duplex communication is done with a control bit 'HALF DUPLEX SEL' (HDSEL in USART_CR3).

As soon as HDSEL is written to 1:

- RX is no longer used,
- TX is always released when no data is transmitted. Thus, it acts as a standard I/O in idle or in reception. It means that the I/O must be configured so that TX is configured as floating input (or output high open-drain) when not driven by the USART.

Apart from this, the communications are similar to what is done in normal USART mode. The conflicts on the line must be managed by the software (by the use of a centralized

arbiter, for instance). In particular, the transmission is never blocked by hardware and continue to occur as soon as a data is written in the data register while the TE bit is set.

19.3.11 Smartcard

The smartcard mode is selected by setting the SCEN bit in the USART_CR3 register. In smartcard mode, the following bits must be kept cleared:

- LINEN bit in the USART_CR2 register,
- HDSEL and IREN bits in the USART_CR3 register.

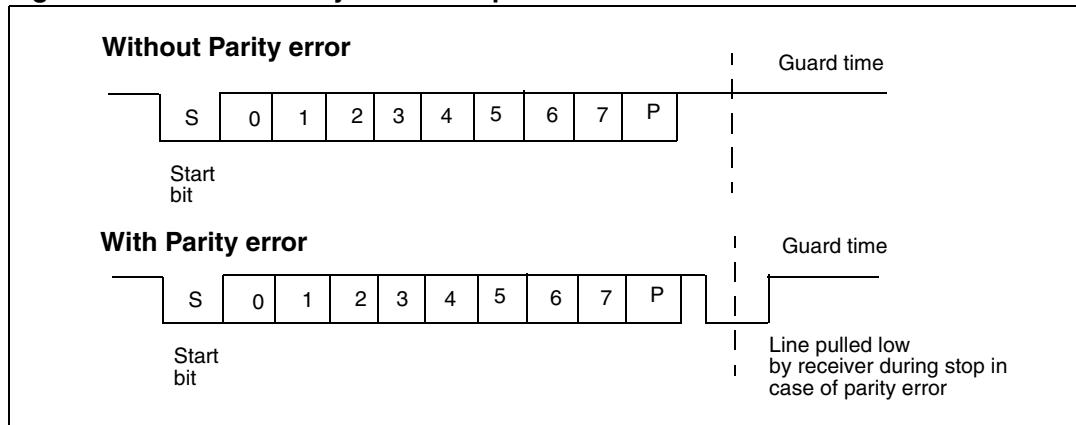
Moreover, the CLKEN bit may be set in order to provide a clock to the smartcard.

The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO7816-3 standard. USART should be configured as:

- 8 bits plus parity: where M=1 and PCE=1 in the USART_CR1 register and either:
 - 0.5 stop bits when receiving: where STOP='01' in the USART_CR2 register
 - 1.5 stop bits when transmitting: where STOP='11' in the USART_CR2 register.

Figure 177 shows examples of what can be seen on the data line with and without parity error.

Figure 177. ISO 7816-3 asynchronous protocol



When connected to a smartcard, the TX output of the USART drives a bidirectional line that the smartcard also drives into. To do so, SW_RX must be connected on the same I/O than TX at product level. The Transmitter output enable TX_EN is asserted during the transmission of the start bit and the data byte, and is deasserted during the stop bit (weak pull up), so that the receive can drive the line in case of a parity error. If TX_EN is not used, TX is driven at high level during the stop bit: Thus the receiver can drive the line as long as TX is configured in open-drain.

Smartcard is a single wire half duplex communication protocol.

- Transmission of data from the transmit shift register is guaranteed to be delayed by a minimum of 1/2 baud clock. In normal operation a full transmit shift register will start shifting on the next baud clock edge. In Smartcard mode this transmission is further delayed by a guaranteed 1/2 baud clock.
- If a parity error is detected during reception of a frame programmed with a 1/2 stop bit period, the transmit line is pulled low for a baud clock period after the completion of the receive frame, i.e. at the end of the 1/2 stop bit period. This is to indicate to the Smartcard that the data transmitted to USART has not been correctly received. This

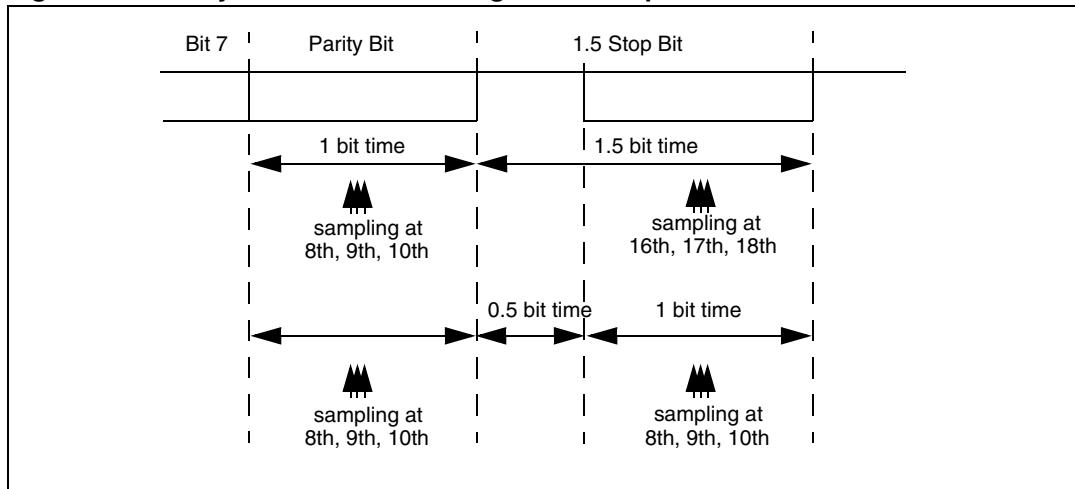
NACK signal (pulling transmit line low for 1 baud clock) will cause a framing error on the transmitter side (configured with 1.5 stop bits). The application can handle re-sending of data according to the protocol. A parity error is ‘NACK’ed by the receiver if the NACK control bit is set, otherwise a NACK is not transmitted.

- The assertion of the TC flag can be delayed by programming the Guard Time register. In normal operation, TC is asserted when the transmit shift register is empty and no further transmit requests are outstanding. In Smartcard mode an empty transmit shift register triggers the guard time counter to count up to the programmed value in the Guard Time register. TC is forced low during this time. When the guard time counter reaches the programmed value TC is asserted high.
- The de-assertion of TC flag is unaffected by Smartcard mode.
- If a framing error is detected on the transmitter end (due to a NACK from the receiver), the NACK will not be detected as a start bit by the receive block of the transmitter. According to the ISO protocol, the duration of the received NACK can be 1 or 2 baud clock periods.
- On the receiver side, if a parity error is detected and a NACK is transmitted the receiver will not detect the NACK as a start bit.

- Note:**
- 1 *A break character is not significant in Smartcard mode. A 0x00 data with a framing error will be treated as data and not as a break.*
 - 2 *No IDLE frame is transmitted when toggling the TE bit. The IDLE frame (as defined for the other configurations) is not defined by the ISO protocol.*

Figure 178 details how the NACK signal is sampled by the USART. In this example the USART is transmitting a data and is configured with 1.5 stop bits. The receiver part of the USART is enabled in order to check the integrity of the data and the NACK signal.

Figure 178. Parity error detection using the 1.5 stop bits



The USART can provide a clock to the smartcard through the SCLK output. In smartcard mode, SCLK is not associated to the communication but is simply derived from the internal peripheral input clock through a 5-bit prescaler. The division ratio is configured in the prescaler register USART_GTPR. SCLK frequency can be programmed from $f_{CK}/2$ to $f_{CK}/62$, where f_{CK} is the peripheral input clock.

19.3.12 IrDA SIR ENDEC block

The IrDA mode is selected by setting the IREN bit in the USART_CR3 register. In IrDA mode, the following bits must be kept cleared:

- LINEN, STOP and CLKEN bits in the USART_CR2 register,
- SCEN and HDSEL bits in the USART_CR3 register.

The IrDA SIR physical layer specifies use of a Return to Zero, Inverted (RZI) modulation scheme that represents logic 0 as an infrared light pulse (see [Figure 179](#)).

The SIR Transmit encoder modulates the Non Return to Zero (NRZ) transmit bit stream output from USART. The output pulse stream is transmitted to an external output driver and infrared LED. USART supports only bit rates up to 115.2Kbps for the SIR ENDEC. In normal mode the transmitted pulse width is specified as 3/16 of a bit period.

The SIR receive decoder demodulates the return-to-zero bit stream from the infrared detector and outputs the received NRZ serial bit stream to USART. The decoder input is normally HIGH (marking state) in the idle state. The transmit encoder output has the opposite polarity to the decoder input. A start bit is detected when the decoder input is low.

- IrDA is a half duplex communication protocol. If the Transmitter is busy (i.e. the USART is sending data to the IrDA encoder), any data on the IrDA receive line will be ignored by the IrDA decoder and if the Receiver is busy (USART is receiving decoded data from the USART), data on the TX from the USART to IrDA will not be encoded by IrDA. While receiving data, transmission should be avoided as the data to be transmitted could be corrupted.
- A '0' is transmitted as a high pulse and a '1' is transmitted as a '0'. The width of the pulse is specified as 3/16th of the selected bit period in normal mode (see [Figure 180](#)).
- The SIR decoder converts the IrDA compliant receive signal into a bit stream for USART.
- The SIR receive logic interprets a high state as a logic one and low pulses as logic zeros.
- The transmit encoder output has the opposite polarity to the decoder input. The SIR output is in low state when idle.
- The IrDA specification requires the acceptance of pulses greater than 1.41 us. The acceptable pulse width is programmable. Glitch detection logic on the receiver end filters out pulses of width less than 2 PSC periods (PSC is the prescaler value programmed in the IrDA low-power Baud Register, USART_GTPR). Pulses of width less than 1 PSC period are always rejected, but those of width greater than one and less than two periods may be accepted or rejected, those greater than 2 periods will be accepted as a pulse. The IrDA encoder/decoder doesn't work when PSC=0.
- The receiver can communicate with a low-power transmitter.
- In IrDA mode, the STOP bits in the USART_CR2 register must be configured to "1 stop bit".

IrDA low-power mode

Transmitter:

In low-power mode the pulse width is not maintained at 3/16 of the bit period. Instead, the width of the pulse is 3 times the low-power baud rate which can be a minimum of 1.42 MHz. Generally this value is 1.8432 MHz (1.42 MHz < PSC < 2.12 MHz). A low-power mode programmable divisor divides the system clock to achieve this value.

Receiver:

Receiving in low-power mode is similar to receiving in normal mode. For glitch detection the USART should discard pulses of duration shorter than $1/\text{PSC}$. A valid low is accepted only if its duration is greater than 2 periods of the IrDA low-power Baud clock (PSC value in USART_GTPR).

- Note:**
- 1 *A pulse of width less than two and greater than one PSC period(s) may or may not be rejected.*
 - 2 *The receiver set up time should be managed by software. The IrDA physical layer specification specifies a minimum of 10 ms delay between transmission and reception (IrDA is a half duplex protocol).*

Figure 179. IrDA SIR ENDEC- block diagram

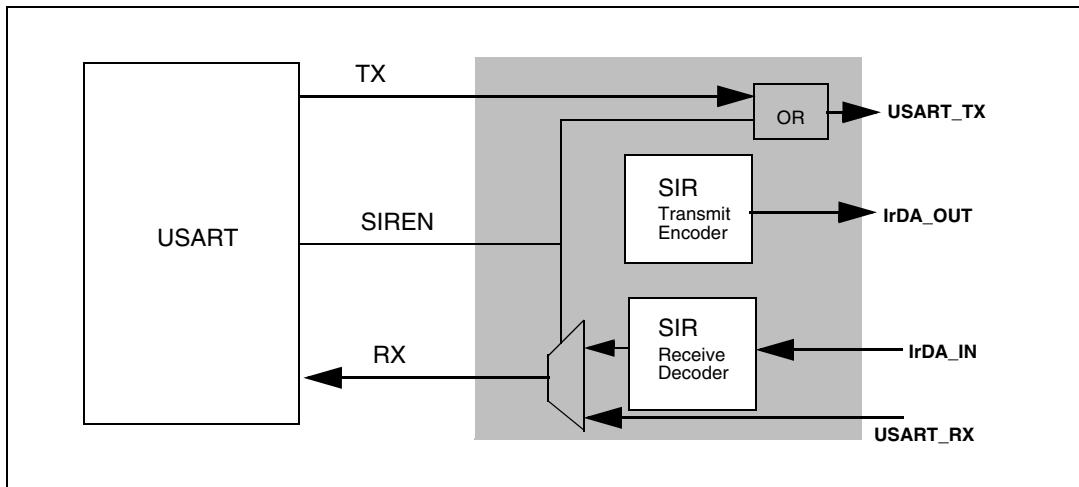
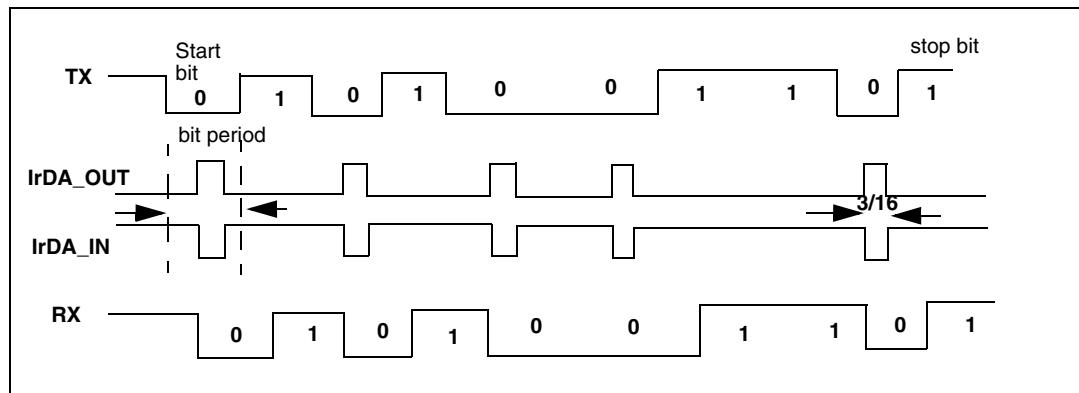


Figure 180. IrDA data modulation (3/16) -Normal Mode



19.3.13 Continuous communication using DMA

The USART is capable to continue communication using the DMA. The DMA requests for Rx buffer and Tx buffer are generated independently.

- Note:**
- You should refer to product specs for availability of the DMA controller. If DMA is not available in the product, you should use the USART as explained in [Section 19.3.3](#) or [19.3.4](#). In the USART_SR register, you can clear the TXE/ RXNE flags to achieve continuous communication.*

Transmission using DMA

DMA mode can be enabled for transmission by setting DMAT bit in the USART_CR3 register. Data is loaded from a SRAM area configured using the DMA peripheral (refer to the DMA specification) to the USART_DR register whenever the TXE bit is set. To map a DMA channel for USART transmission, use the following procedure (x denotes the channel number):

1. Write the USART_DR register address in the DMA control register to configure it as the destination of the transfer. The data will be moved to this address from memory after each TXE event.
2. Write the memory address in the DMA control register to configure it as the source of the transfer. The data will be loaded into the USART_DR register from this memory area after each TXE event.
3. Configure the total number of bytes to be transferred to the DMA control register.
4. Configure the channel priority in the DMA register
5. Configure DMA interrupt generation after half/ full transfer as required by the application.
6. Activate the channel in the DMA register.

When the number of data transfers programmed in the DMA Controller is reached, the DMA controller generates an interrupt on the DMA channel interrupt vector. The DMAT bit should be cleared by software in the USART_CR3 register during the interrupt subroutine.

Note:

If DMA is used for transmission, do not enable the TXEIE bit.

Reception using DMA

DMA mode can be enabled for reception by setting the DMAR bit in USART_CR3 register. Data is loaded from the USART_DR register to a SRAM area configured using the DMA peripheral (refer to the DMA specification) whenever a data byte is received. To map a DMA channel for USART reception, use the following procedure (x denotes the channel number):

1. Write the USART_DR register address in the DMA control register to configure it as the source of the transfer. The data will be moved from this address to the memory after each RXNE event.
2. Write the memory address in the DMA control register to configure it as the destination of the transfer. The data will be loaded from USART_DR to this memory area after each RXNE event.
3. Configure the total number of bytes to be transferred in the DMA control register.
4. Configure the channel priority in the DMA control register
5. Configure interrupt generation after half/ full transfer as required by the application.
6. Activate the channel in the DMA control register.

When the number of data transfers programmed in the DMA Controller is reached, the DMA controller generates an interrupt on the DMA channel interrupt vector. The DMAR bit should be cleared by software in the USART_CR3 register during the interrupt subroutine.

Note:

If DMA is used for reception, do not enable the RXNEIE bit.

Error flagging and interrupt generation in multibuffer communication

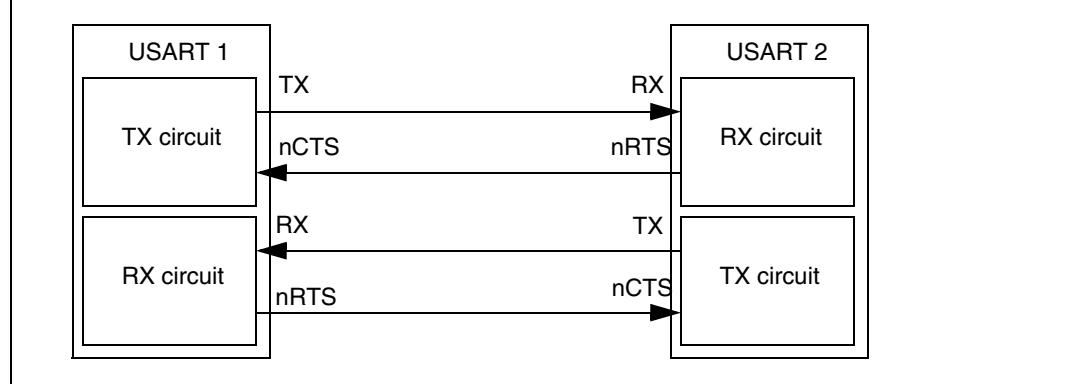
In case of multibuffer communication if any error occurs during the transaction the error flag will be asserted after the current byte. An interrupt will be generated if the interrupt enable flag is set. For framing error, overrun error and noise flag which are asserted with RXNE in

case of single byte reception, there will be separate error flag interrupt enable bit (EIE bit in the USART_CR3 register), which if set will issue an interrupt after the current byte with either of these errors.

19.3.14 Hardware flow control

It is possible to control the serial data flow between 2 devices by using the nCTS input and the nRTS output. The [Figure 181](#) shows how to connect 2 devices in this mode:

Figure 181. Hardware flow control between 2 USART

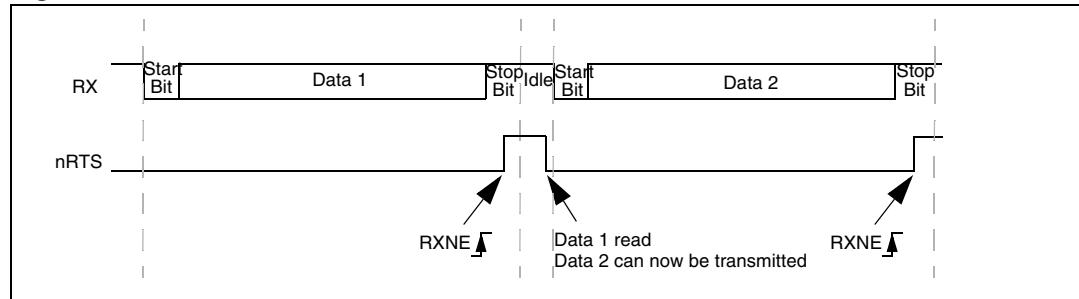


RTS and CTS flow control can be enabled independently by writing respectively RTSE and CTSE bits to 1 (in the USART_CR3 register).

RTS flow control

If the RTS flow control is enabled (RTSE=1), then nRTS is asserted (tied low) as long as the USART receiver is ready to receive a new data. When the receive register becomes empty, nRTS is deasserted, indicating that the transmission is expected to stop at the end of the current frame. [Figure 182](#) shows an example of communication with RTS flow control enabled.

Figure 182. RTS flow control

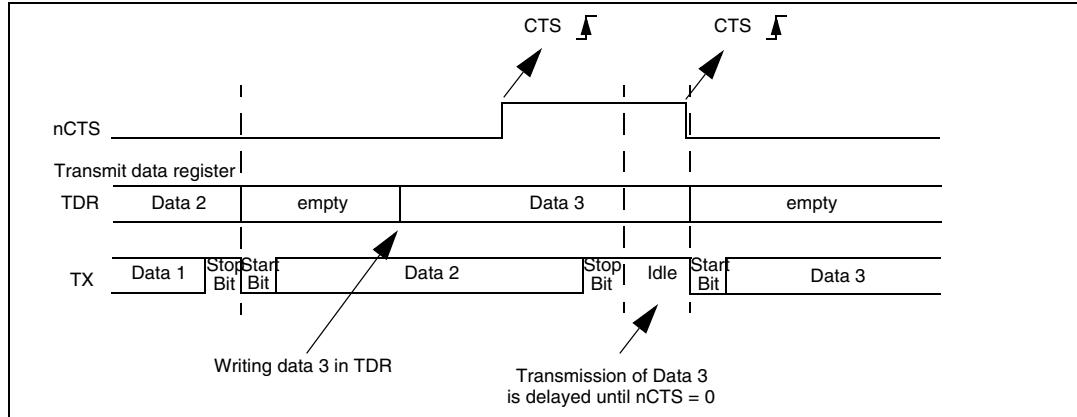


CTS flow control

If the CTS flow control is enabled (CTSE=1), then the transmitter checks the nCTS input before transmitting the next frame. If nCTS is asserted (tied low), then the next data is transmitted (assuming that a data is to be transmitted, in other words, if TXE=0), else the transmission does not occur. When nCTS is deasserted during a transmission, the current transmission is completed before the transmitter stops.

When CTSE=1, the CTSIF status bit is automatically set by hardware as soon as the nCTS input toggles. It indicates when the receiver becomes ready or not ready for communication. An interrupt is generated if the CTSIE bit in the USART_CR3 register is set. The figure below shows an example of communication with CTS flow control enabled.

Figure 183. CTS flow control



19.4 Interrupt requests

Table 70. USART interrupt requests

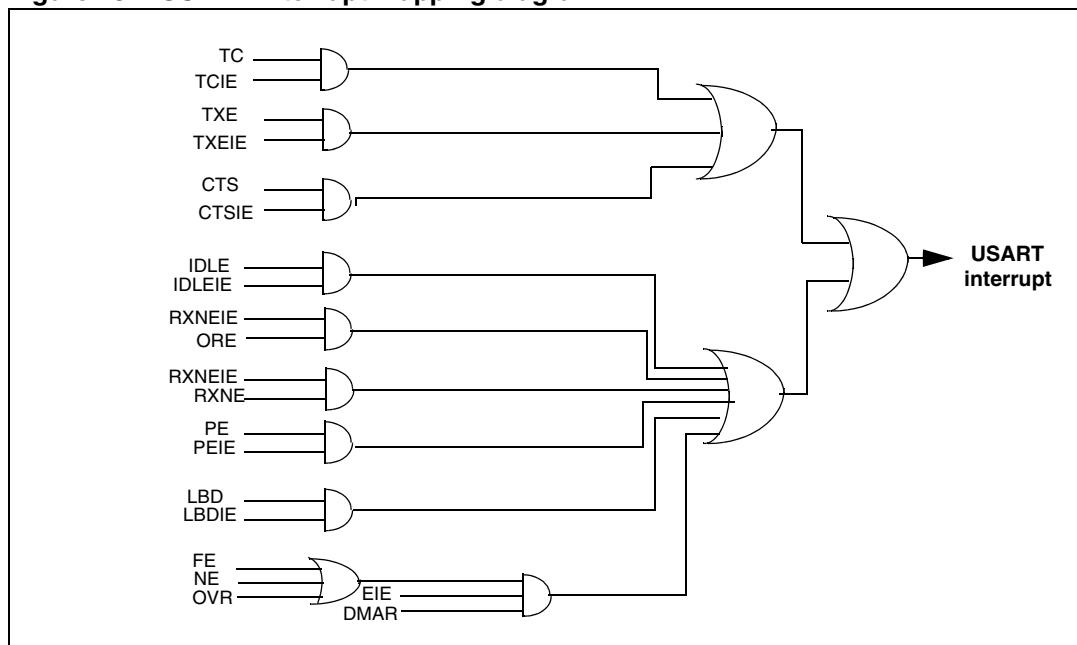
Interrupt event	Event flag	Enable Control bit
Transmit Data Register Empty	TXE	TXEIE
CTS flag	CTS	CTSIE
Transmission Complete	TC	TCIE
Received Data Ready to be Read	RXNE	RXNEIE
Overrun Error Detected	ORE	
Idle Line Detected	IDLE	IDLEIE
Parity Error	PE	PEIE
Break Flag	LBD	LBDIE
Noise Flag, Overrun error and Framing Error in multibuffer communication	NE or ORE or FE	EIE

The USART interrupt events are connected to the same interrupt vector (see [Figure 184](#)).

- During transmission: Transmission Complete, Clear to Send or Transmit Data Register empty interrupt.
- While receiving: Idle Line detection, Overrun error, Receive Data register not empty, Parity error, LIN break detection, Noise Flag (only in multi buffer communication) and Framing Error (only in multi buffer communication).

These events generate an interrupt if the corresponding Enable Control Bit is set.

Figure 184. USART interrupt mapping diagram



19.5 USART register description

Refer to [Section 1.1 on page 23](#) for a list of abbreviations used in register descriptions.

19.5.1 Status register (USART_SR)

Address offset: 0x00

Reset value: 0x00C0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE	
rc					rc	r	rc	r	r	r	r	r	r	r	r

Bits 31:10 Reserved, forced by hardware to 0.

Bit 9 **CTS: CTS Flag**

This bit is set by hardware when the nCTS input toggles, if the CTSE bit is set. It is cleared by software (by writing it to 0). An interrupt is generated if CTSIE=1 in the USART_CR3 register.

0: No change occurred on the nCTS status line

1: A change occurred on the nCTS status line

Bit 8 **LBD: LIN Break Detection Flag**

LIN Break Detection Flag (Status flag)

0: LIN Break not detected

1: LIN break detected

Note: An interrupt is generated when LBD=1 if LBDIE=1

Bit 7 **TXE: Transmit Data Register Empty**

This bit is set by hardware when the content of the TDR register has been transferred into the shift register. An interrupt is generated if the TXEIE bit =1 in the USART_CR1 register. It is cleared by a write to the USART_DR register.

0: Data is not transferred to the shift register

1: Data is transferred to the shift register)

Note: This bit is used during single buffer transmission.

Bit 6 **TC: Transmission Complete**

This bit is set by hardware when transmission of a frame containing Data is complete. An interrupt is generated if TCIE=1 in the USART_CR1 register. It is cleared by a software sequence (an read to the USART_SR register followed by a write to the USART_DR register).

0: Transmission is not complete

1: Transmission is complete

Bit 5 **RXNE: Read Data Register Not Empty**

This bit is set by hardware when the content of the RDR shift register has been transferred to the USART_DR register. An interrupt is generated if RXNEIE=1 in the USART_CR1 register. It is cleared by a read to the USART_DR register.

0: Data is not received

1: Received data is ready to be read.

Bit 4 IDLE: *IDLE line detected.*

This bit is set by hardware when an Idle Line is detected. An interrupt is generated if the IDLEIE=1 in the USART_CR1 register. It is cleared by a software sequence (an read to the USART_SR register followed by a read to the USART_DR register).

0: No Idle Line is detected

1: Idle Line is detected

Note: The IDLE bit will not be set again until the RXNE bit has been set itself (i.e. a new idle line occurs).

Bit 3 ORE: *OverRun Error.*

This bit is set by hardware when the word currently being received in the shift register is ready to be transferred into the RDR register while RXNE=1. An interrupt is generated if RXNEIE=1 in the USART_CR1 register. It is cleared by a software sequence (an read to the USART_SR register followed by a read to the USART_DR register).

0: No Overrun error

1: Overrun error is detected

Note: When this bit is set, the RDR register content will not be lost but the shift register will be overwritten. An interrupt is generated on ORE flag in case of Multi Buffer communication if the EIE bit is set.

Bit 2 NE: *Noise Error Flag.*

This bit is set by hardware when noise is detected on a received frame. It is cleared by a software sequence (an read to the USART_SR register followed by a read to the USART_DR register).

0: No noise is detected

1: Noise is detected

Note: This bit does not generate interrupt as it appears at the same time as the RXNE bit which itself generates an interrupting interrupt is generated on NE flag in case of Multi Buffer communication if the EIE bit is set.

Bit 1 FE: *Framing Error.*

This bit is set by hardware when a de-synchronization, excessive noise or a break character is detected. It is cleared by a software sequence (an read to the USART_SR register followed by a read to the USART_DR register).

0: No Framing error is detected

1: Framing error or break character is detected

Note:

This bit does not generate interrupt as it appears at the same time as the RXNE bit which itself generates an interrupt. If the word currently being transferred causes both frame error and overrun error, it will be transferred and only the ORE bit will be set.

An interrupt is generated on FE flag in case of Multi Buffer communication if the EIE bit is set.

Bit 0 PE: *Parity Error.*

This bit is set by hardware when a parity error occurs in receiver mode. It is cleared by a software sequence (a read to the status register followed by a read to the USART_DR data register). An interrupt is generated if PEIE=1 in the USART_CR1 register.

0: No parity error

1: Parity error

19.5.2 Data register (USART_DR)

Address offset: 0x04

Reset value: Undefined

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

Bits 31:9 Reserved, forced by hardware to 0.

Bits 8:0 **DR[8:0]: Data value.**

Contains the Received or Transmitted data character, depending on whether it is read from or written to.

The Data register performs a double function (read and write) since it is composed of two registers, one for transmission (TDR) and one for reception (RDR)

The TDR register provides the parallel interface between the internal bus and the output shift register (see Figure 1).

The RDR register provides the parallel interface between the input shift register and the internal bus.

When transmitting with the parity enabled (PCE bit set to 1 in the USART_CR1 register), the value written in the MSB (bit 7 or bit 8 depending on the data length) has no effect because it is replaced by the parity.

When receiving with the parity enabled, the value read in the MSB bit is the received parity bit.

19.5.3 Baud Rate Register (USART_BRR)

Note: The baud counters stop counting if the TE or RE bits are disabled respectively.

Address offset: 0x08

Reset value: 0x0000

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

Bits 31:16 Reserved, forced by hardware to 0.

Bits 15:4 **DIV_Mantissa[11:0]: mantissa of USARTDIV.**

These 12 bits define the mantissa of the USART Divider (USARTDIV)

Bits 3:0 **DIV_Fraction[3:0]: fraction of USARTDIV.**

These 4 bits define the fraction of the USART Divider (USARTDIV)

19.5.4 Control register 1 (USART_CR1)

Address offset: 0x0C

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNE IE	IDLEIE	TE	RE	RWU	SBK	rw

Bits 31:14 Reserved, forced by hardware to 0.

Bit 13 **UE**: USART Enable.

When this bit is cleared the USART prescalers and outputs are stopped and the end of the current byte transfer in order to reduce power consumption. This bit is set and cleared by software.

0: USART prescaler and outputs disabled

1: USART enabled

Bit 12 **M**: word length.

This bit determines the word length. It is set or cleared by software.

0: 1 Start bit, 8 Data bits, n Stop bit

1: 1 Start bit, 9 Data bits, 1 Stop bit

Note:

The M bit must not be modified during a data transfer (both transmission and reception)

Bit 11 **WAKE**: Wakeup method.

This bit determines the USART wakeup method, it is set or cleared by software.

0: Idle Line

1: Address Mark

Bit 10 **PCE**: Parity Control Enable.

This bit selects the hardware parity control (generation and detection). When the parity control is enabled, the computed parity is inserted at the MSB position (9th bit if M=1; 8th bit if M=0) and parity is checked on the received data. This bit is set and cleared by software. Once it is set, PCE is active after the current byte (in reception and in transmission).

0: Parity control disabled

1: Parity control enabled

Bit 9 **PS**: Parity Selection.

This bit selects the odd or even parity when the parity generation/detection is enabled (PCE bit set). It is set and cleared by software. The parity will be selected after the current byte.

0: Even parity

1: Odd parity

Bit 8 **PEIE**: PE Interrupt Enable.

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated whenever PE=1 in the USART_SR register

Bit 7 TXEIE: TXE Interrupt Enable.

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated whenever TXE=1 in the USART_SR register

Bit 6 TCIE: Transmission Complete Interrupt Enable.

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated whenever TC=1 in the USART_SR register

Bit 5 RXNEIE: RXNE Interrupt Enable.

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated whenever ORE=1 or RXNE=1 in the USART_SR register

Bit 4 IDLEIE: IDLE Interrupt Enable.

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated whenever IDLE=1 in the USART_SR register

Bit 3 TE: Transmitter Enable.

This bit enables the transmitter. It is set and cleared by software.

0: Transmitter is disabled

1: Transmitter is enabled

Notes:

1: During transmission, a “0” pulse on the TE bit (“0” followed by “1”) sends a preamble (idle line) after the current word, except in smartcard mode.

2: When TE is set there is a 1 bit-time delay before the transmission starts.

Bit 2 RE: Receiver Enable.

This bit enables the receiver. It is set and cleared by software.

0: Receiver is disabled

1: Receiver is enabled and begins searching for a start bit

Bit 1 RWU: Receiver wakeup.

This bit determines if the USART is in mute mode or not. It is set and cleared by software and can be cleared by hardware when a wakeup sequence is recognized.

0: Receiver in active mode

1: Receiver in mute mode

Notes:

1: Before selecting Mute mode (by setting the RWU bit) the USART must first receive a data byte, otherwise it cannot function in Mute mode with wakeup by Idle line detection.

2: In Address Mark Detection wakeup configuration (WAKE bit=1) the RWU bit cannot be modified by software while the RXNE bit is set.

Bit 0 SBK: Send Break.

This bit set is used to send break characters. It can be set and cleared by software. It should be set by software, and will be reset by hardware during the stop bit of break.

0: No break character is transmitted

1: Break character will be transmitted

19.5.5 Control register 2 (USART_CR2)

Address offset: 0x10

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LINEN	STOP[1:0]	CLK EN	COPOL	CPHA	LBCL	Res.	LBDIE	LBDL	Res.					ADD[3:0]
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:15 Reserved, forced by hardware to 0.

Bit 14 **LINEN**: LIN mode enable

This bit is set and cleared by software.

0: LIN mode disabled

1: LIN mode enabled

The LIN mode enables the capability to send LIN Sync Breaks (13 low bits) using the SBK bit in the USART_CR1 register, and to detect LIN Sync breaks.

Bits 13:12 **STOP**: STOP bits.

These bits are used for programming the stop bits.

00: 1 Stop bit

01: 0.5 Stop bit

10: 2 Stop bits

11: 1.5 Stop bit

Bit 11 **CLKEN**: Clock Enable.

This bit allows the user to enable the SCLK pin.

0: SCLK pin disabled

1: SCLK pin enabled

Bit 10 **COPOL**: Clock Polarity.

This bit allows the user to select the polarity of the clock output on the SCLK pin in synchronous mode. It works in conjunction with the CPHA bit to produce the desired clock/data relationship

0: Steady low value on SCLK pin outside transmission window.

1: Steady high value on SCLK pin outside transmission window.

Bit 9 **CPHA**: Clock Phase

This bit allows the user to select the phase of the clock output on the SCLK pin in synchronous mode. It works in conjunction with the COPOL bit to produce the desired clock/data relationship (see figures 174 to 175)

0: The first clock transition is the first data capture edge.

1: The second clock transition is the first data capture edge.

Bit 8 **LBCL**: *Last Bit Clock pulse.*

This bit allows the user to select whether the clock pulse associated with the last data bit transmitted (MSB) has to be output on the SCLK pin in synchronous mode.

0: The clock pulse of the last data bit is not output to the SCLK pin.

1: The clock pulse of the last data bit is output to the SCLK pin.

Note: The last bit is the 8th or 9th data bit transmitted depending on the 8 or 9 bit format selected by the M bit in the USART_CR1 register.

Bit 7 Reserved, forced by hardware to 0.

Bit 6 **LBDIE**: *LIN Break Detection Interrupt Enable.*

Break interrupt mask (break detection using break delimiter).

0: Interrupt is inhibited

1: An interrupt is generated whenever LBD=1 in the USART_SR register

Bit 5 **LBDL**: *LIN Break Detection Length.*

This bit is for selection between 11 bit or 10 bit break detection.

0: 10 bit break detection

1: 11 bit break detection

Bit 4 Reserved, forced by hardware to 0.

Bits 3:0 **ADD[3:0]**: *Address of the USART node.*

This bit-field gives the address of the USART node.

This is used in multiprocessor communication during mute mode, for wake up with address mark detection.

Note: These 3 bits (CPOL, CPHA, LBCL) should not be written while the transmitter is enabled.

19.5.6 Control register 3 (USART_CR3)

Address offset: 0x14

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HD SEL	IRLP	IREN	EIE	
rw				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:11 Reserved, forced by hardware to 0.

Bit 10 **CTSIE**: *CTS Interrupt Enable*.

0: Interrupt is inhibited

1: An interrupt is generated whenever CTS=1 in the USART_SR register

Bit 9 **CTSE**: *CTS Enable*.

0: CTS hardware flow control disabled

1: CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0). If the nCTS input is deasserted while a data is being transmitted, then the transmission is completed before stopping. If a data is written into the data register while nCTS is asserted, the transmission is postponed until nCTS is asserted.

Bit 8 **RTSE**: *RTS Enable*.

0: RTS hardware flow control disabled

1: RTS interrupt enabled, data is only requested when there is space in the receive buffer. The transmission of data is expected to cease after the current character has been transmitted. The nRTS output is asserted (tied to 0) when a data can be received.

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

Bit 4 **NACK**: *Smartcard NACK enable.*

- 0: NACK transmission in case of parity error is disabled
- 1: NACK transmission during parity error is enabled.

Bit 3 **HDSEL**: *Half-Duplex Selection.*

- Selection of Single-wire Half-duplex mode
- 0: Half duplex mode is not selected
- 1: Half duplex mode is selected

Bit 2 **IRLP**: *IrDA Low-Power.*

- This bit is used for selecting between normal and low-power IrDA modes
- 0: Normal mode
- 1: Low-power mode

Bit 1 **IREN**: *IrDA mode Enable.*

- This bit is set and cleared by software.
- 0: IrDA disabled
- 1: IrDA enabled

Bit 0 **EIE**: *Error Interrupt Enable.*

Error Interrupt Enable Bit is required to enable interrupt generation in case of a framing error, overrun error or noise error (FE=1 or ORE=1 or NE=1 in the USART_SR register) in case of Multi Buffer Communication (DMAR=1 in the USART_CR3 register).

- 0: Interrupt is inhibited
- 1: An interrupt is generated whenever DMAR=1 in the USART_CR3 register and FE=1 or ORE=1 or NE=1 in the USART_SR register.

19.5.7 Guard time and prescaler register (USART_GTPR)

Address offset: 0x18

Reset value: 0x0000

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

Bits 31:16 Reserved, forced by hardware to 0.

Bits 15:8 **GT[7:0]: Guard time value.**

This bit-field gives the Guard time value in terms of number of baud clocks.

This is used in Smartcard mode. The Transmission Complete flag is set after this guard time value.

Bits 7:0 **PSC[7:0]: Prescaler value.**

– **In IrDA Low-power mode:**

PSC[7:0] = IrDA Low-Power Baud Rate

Used for programming the prescaler for dividing the system clock to achieve the low-power frequency:

The source clock is divided by the value given in the register (8 significant bits):

00000000: Reserved - do not program this value

00000001: divides the source clock by 1

00000010: divides the source clock by 2

...

– **In normal IrDA mode:** PSC must be set to 00000001.

– **In smartcard mode:**

PSC[4:0]: Prescaler value.

Used for programming the prescaler for dividing the system clock to provide the smartcard clock.

The value given in the register (5 significant bits) is multiplied by 2 to give the division factor of the source clock frequency:

00000: Reserved - do not program this value

00001: divides the source clock by 2

00010: divides the source clock by 4

00011: divides the source clock by 6

...

Note: Bits [7:5] have no effect if Smartcard mode is used.

19.6 USART register map

Table 71. USART register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
0x00	USART_SR																							CTS	0	LBD	8	IDLE	4	ORE	3	NE	2	FE	1	PE	0
	Reset value																																				
0x04	USART_DR																																				
	Reset value																																				
0x08	USART_BRR																																				
	Reset value																																				
0x0C	USART_CR1																																				
	Reset value																																				
0x10	USART_CR2																																				
	Reset value																																				
0x14	USART_CR3																																				
	Reset value																																				
0x18	USART_GTPR																																				
	Reset value																																				

Refer to [Table 1 on page 27](#) for the register boundary addresses.

20 Debug support (DBG)

20.1 Overview

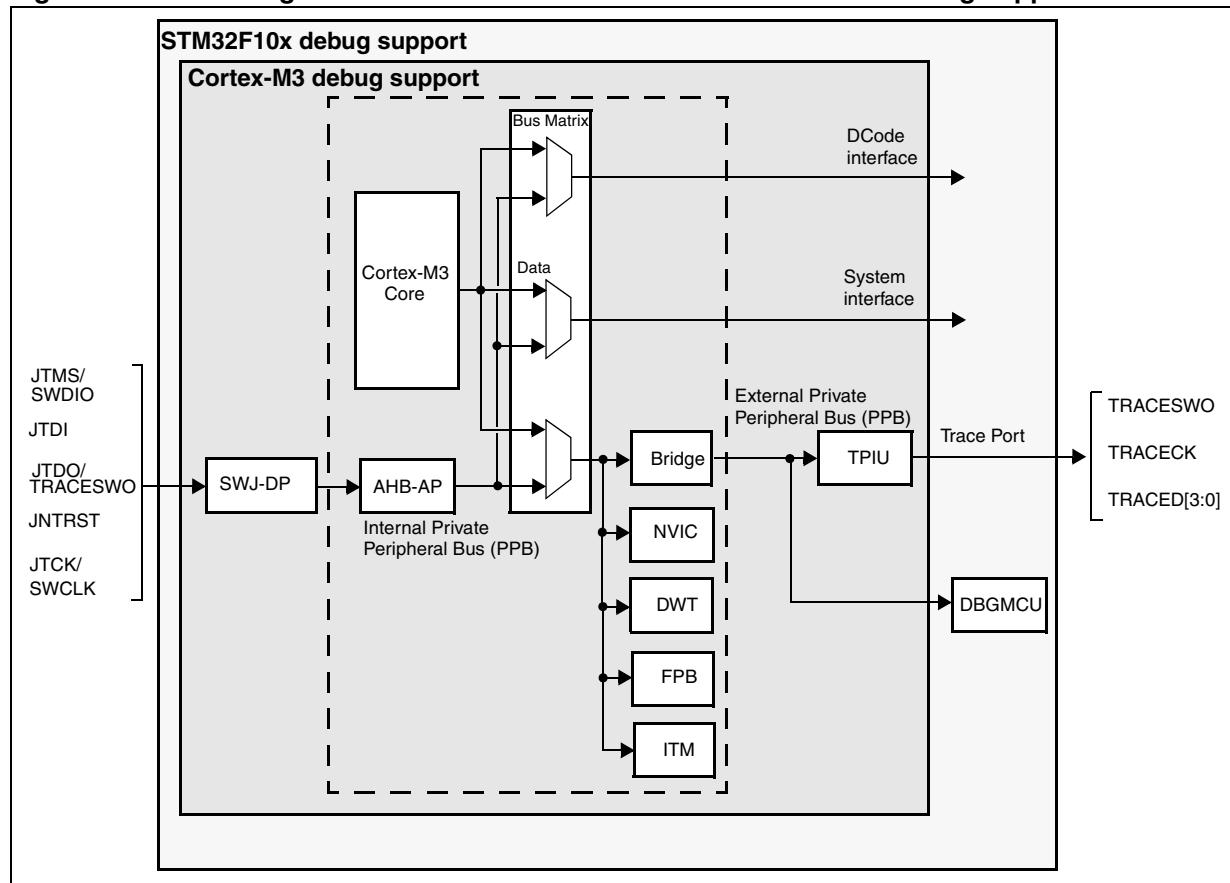
The STM32F10xxx is built around a Cortex-M3 core which contains hardware extensions for advanced debugging features. The debug extensions allow the core to be stopped either on a given instruction fetch (breakpoint) or data access (watchpoint). When stopped, the core's internal state and the system's external state may be examined. Once examination is complete, the core and the system may be restored and program execution resumed.

The debug features are used by the debugger host when connecting to and debugging the STM32F10x MCU.

Two interfaces for debug are available:

- Serial wire
- JTAG debug port

Figure 185. Block diagram of STM32F10xxx-level and Cortex-M3-level debug support



Note: The debug features embedded in the Cortex-M3 core are a subset of the ARM CoreSight Design Kit.

The ARM Cortex-M3 core provides integrated on-chip debug support. It is comprised of:

- SWJ-DP: Serial wire / JTAG debug port
- AHP-AP: AHB access port
- ITM: Instrumentation trace macrocell
- FPB: Flash patch breakpoint
- DWT: Data watchpoint trigger
- TPUI: Trace port unit interface (available on larger packages, where the corresponding pins are mapped)

It also includes debug features dedicated to STM32F10xxx:

- Flexible debug pinout assignment
- MCU debug box (support for low-power modes, control over peripheral clocks, etc.)

Note: For further information on debug functionality supported by the ARM Cortex-M3 core, refer to the Cortex-M3 r1p1 Technical Reference Manual (TRM) and to the CoreSight Design Kit r1p0 TRM.

20.2 Referenced ARM documentation

- Cortex™-M3 r1p1 Technical Reference Manual (TRM)
- ARM Debug Interface V5
- ARM CoreSight Design Kit revision r1p0 Technical Reference Manual

20.3 SWJ debug port (serial wire and JTAG)

The STM32F10xxx core integrates the Serial Wire / JTAG Debug Port (SWJ-DP). It is an ARM standard CoreSight debug port that combines a JTAG-DP (5-pin) interface and a SW-DP (2-pin) interface.

- The JTAG Debug Port (JTAG-DP) provides a 5-pin standard JTAG interface to the AHP-AP port.
- The Serial Wire Debug Port (SW-DP) provides a 2-pin (clock + data) interface to the AHP-AP port.

In the SWJ-DP, the two JTAG pins of the SW-DP are multiplexed with some of the five JTAG pins of the JTAG-DP.

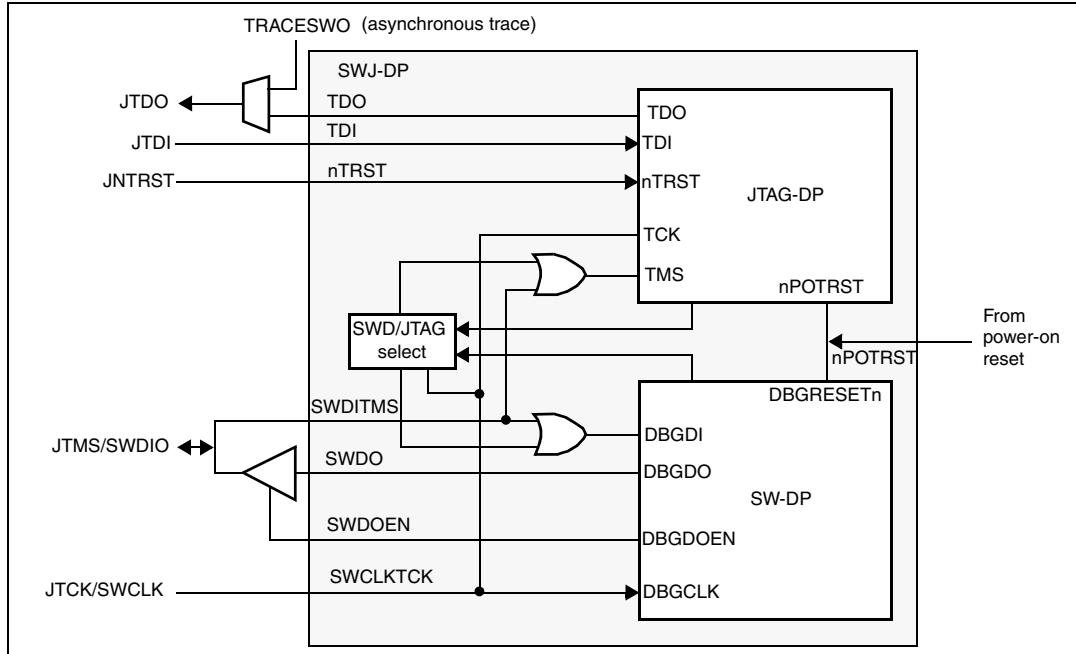
Figure 186. SWJ debug port

Figure 186 shows that the asynchronous TRACE output (TRACESWO) is multiplexed with TDO. This means that the asynchronous trace can only be used with SW-DP, not JTAG-DP.

20.3.1 Mechanism to select the JTAG-DP or the SW-DP

By default, the JTAG-Debug Port is active.

If the debugger host wants to switch to the SW-DP, it must provide a dedicated JTAG sequence on TMS/TCK (respectively mapped to SWDIO and SWCLK) which disables the JTAG-DP and enables the SW-DP. This way it is possible to activate the SWDP using only the SWCLK and SWDIO pins.

This sequence is:

1. Send more than 50 TCK cycles with TMS (SWDIO) =1
2. Send the 16-bit sequence on TMS (SWDIO) = 0111100111100111 (MSB transmitted first)
3. Send more than 50 TCK cycles with TMS (SWDIO) =1

20.4 Pinout and debug port pins

The STM32F10xxx MCU is available in various packages with different numbers of available pins. As a result, some functionality related to pin availability may differ between packages.

20.4.1 SWJ debug port pins

Five pins are used as outputs from the STM32F10xxx for the SWJ-DP as *alternate functions* of General Purpose I/Os. These pins are available on all packages.

Table 72. SWJ debug port pins

SWJ-DP pin name	JTAG debug port		SW debug port		Pin assignment
	Type	Description	Type	Debug assignment	
JTMS/SWDIO	I	JTAG Test Mode Selection	I/O	Serial Wire Data Input/Output	PA13
JTCK/SWCLK	I	JTAG Test Clock	I	Serial Wire Clock	PA14
JTDI	I	JTAG Test Data Input	-	-	PA15
JTDO/TRACESWO	O	JTAG Test Data Output	-	TRACESWO if async trace is enabled	PB3
JNTRST	I	JTAG Test nReset	-	-	PB4

20.4.2 Flexible SWJ-DP pin assignment

After RESET (SYSRESETn or PORESETn), all five pins used for the SWJ-DP are assigned as dedicated pins immediately usable by the debugger host (note that the trace outputs are not assigned except if explicitly programmed by the debugger host).

However, the STM32F10xxx MCU implements the REMAP_DBGAFR register to disable some part or all of the SWJ-DP port and so releases the associated pins for General Purpose I/Os usage. This register is mapped on an APB bridge connected to the Cortex-M3 System Bus. Programming of this register is done by the user software program and not the debugger host.

Three control bits allow the configuration of the SWJ-DP pin assignments. These bits are reset by the System Reset.

- REMAP_AF_REG (@ 0x4001 0004 in STM32F10xxx MCU)
 - READ: APB - No Wait State
 - WRITE: APB - 1 Wait State if the write buffer of the AHB-APB bridge is full.

Bit 26:24= **SWJ_CFG[2:0]**

Set and cleared by software.

These bits are used to configure the number of pins assigned to the SWJ debug port. The goal is to release as much as possible the number of pins to be used as General Purpose I/Os if using a small size for the debug port.

The default state after reset is “000” (whole pins assigned for a full JTAG-DP connection). Only one of the 3 bits can be set (it is forbidden to set more than one bit).

Table 73. Flexible SWJ-DP pin assignment

SWJ_CFG [2:0]	Available debug ports	SWJ I/O pin assigned				
		PA13 / JTMS/SWDIO	PA14 / JTCK/SWCLK	PA15 / JTDI	PB3 / JTDO	PB4/JNTRST
000	Full SWJ (JTAG-DP + SW-DP) - Reset State	X	X	X	X	X
001	Full SWJ (JTAG-DP + SW-DP) but without JNTRST	X	X	X	X	
010	JTAG-DP Disabled and SW-DP Enabled	X	X			
100	JTAG-DP Disabled and SW-DP Disabled					Released
other	Forbidden					

Note:

When the APB bridge write buffer is full, it takes one extra APB cycle when writing the REMAP_AF register. This is because the deactivation of the JTAGSW pins is done in two cycles to guarantee a clean level on the nTRST and TCK input signals of the core.

- Cycle 1: the JTAGSW input signals to the core are tied to 1 or 0 (to 1 for nTRST, TDI and TMS, to 0 for TCK)
- Cycle 2: the GPIO controller takes the control signals of the SWJTAG I/O pins (like controls of direction, pull-up/down, Schmitt trigger activation, etc.).

20.4.3 Internal pull-up and pull-down on JTAG pins

It is necessary to ensure that the JTAG input pins are not floating since they are directly connected to flip-flops to control the debug mode features. Special care must be taken with the SWCLK/TCK pin which is directly connected to the clock of some of these flip-flops.

To avoid any uncontrolled I/O levels, the STM32F10xxx embeds internal pull-ups and pull-downs on JTAG input pins:

- JNTRST: Internal pull-up
- JTDI: Internal pull-up
- JTMS/SWDIO: Internal pull-up
- TCK/SWCLK: Internal pull-down

Once a JTAG I/O is released by the user software, the GPIO controller takes control again. The reset states of the GPIO control registers put the I/Os in the equivalent state:

- JNTRST: Input pull-up
- JTDI: Input pull-up
- JTMS/SWDIO: Input pull-up
- JTCK/SWCLK: Input pull-down
- JTDO: Input floating

The software can then use these I/Os as standard GPIOs.

Note:

The JTAG IEEE standard recommends to add pull-ups on TDI, TMS and nTRST but there is no special recommendation for TCK. However, for STM32F10xxx, an integrated pull-down is used for JTCK.

Having embedded pull-ups and pull-downs removes the need to add external resistors.

20.4.4 Using serial wire and releasing the unused debug pins as GPIOs

To use the serial wire DP to release some GPIOs, the user software must set SWJ_CFG=010 just after reset. This releases PA15, PB3 and PB4 which now become available as GPIOs.

When debugging, the host performs the following actions:

- Under system RESET, all SWJ pins are assigned (JTAG-DP + SW-DP)
- Under system RESET, the debugger host sends the JTAG sequence to switch from the JTAG-DP to the SW-DP.
- Still under system RESET, the debugger sets a breakpoint on vector reset
- The System Reset is released and the Core halts.
- All the debug communications from this point are done using the SW-DP. The other JTAG pins can then be reassigned as GPIOs by the user software.

Note: For user software designs, note that:

To release the debug pins, remember that they will be first configured either in input-pull-up ($nTRST$, TMS , TDI) or pull-down (TCK) or output tristate (TDO) for a certain duration after reset until the instant when the user software releases the pins.

When debug pins (JTAG or SW or TRACE) are mapped, changing the corresponding I/O pin configuration in the IOPORT controller has no effect.

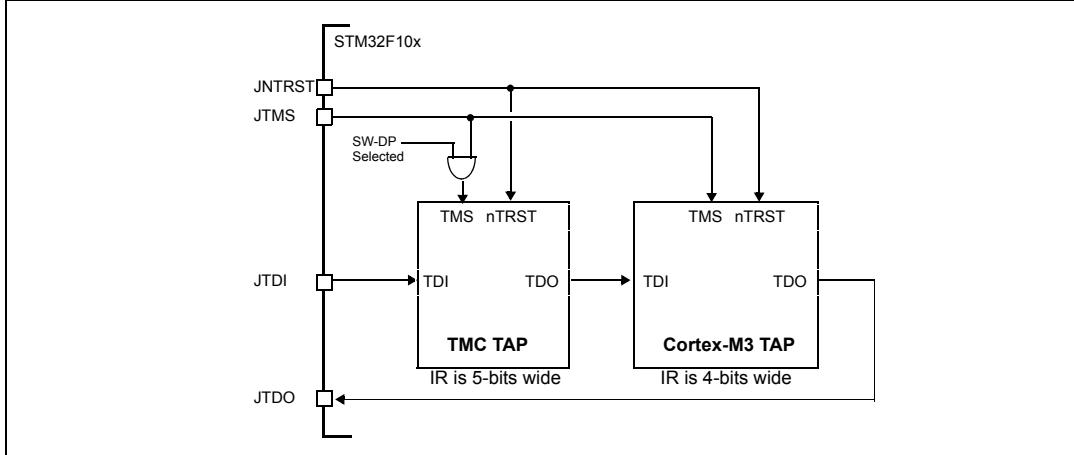
20.5 STM32F10xxx JTAG TAP connection

The STM32F10xxx MCU integrates two serially-connected JTAG TAPs, the TMC TAP dedicated for Test (IR is 5-bit wide) and the Cortex-M3 TAP (IR is 4-bits wide).

To access the TAP of the Cortex-M3 for debug purposes:

1. First, it is necessary to shift the BYPASS instruction of the TMC TAP.
2. Then, for each IR shift, the scan chain contains 9 bits (=5+4) and the unused TAP instruction must be shifted in using the BYPASS instruction.
3. For each data shift, the unused TAP, which is in BYPASS mode, adds 1 extra data bit in the data scan chain.

Note: **Important:** Once Serial-Wire is selected using the dedicated ARM JTAG sequence, the TMC TAP is automatically disabled (JTMS forced high).

Figure 187. JTAG TAP connections

20.6 ID codes and locking mechanism

There are several ID codes inside the STM32F10xxx MCU. ST strongly recommends tools designers to lock their debuggers using the MCU DEVICE ID code located in the external PPB memory map at address 0xE0042000.

20.6.1 MCU device ID code

The MCU STM32F10xxx integrates an MCU ID code. This ID identifies the ST MCU part-number and the die revision. It is part of the DBG_MCU Component and is mapped on the external PPB bus (see [Section 20.15 on page 487](#)). This code is accessible using the JTAG debug port (4 to 5 pins) or the SW debug port (two pins) or by the user software. It is even accessible while the MCU is under system reset.

DBGMCU_IDCODE

Address: 0xE004 2000

Only 32-bits access supported

Read only = 0xXXXXX410 where X is undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
REV_ID															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		DEV_ID													
Res.		r	r	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:

- 0x0000 = Revision A
- 0x2000 = Revision B
- 0x2001 = Revision Z

Bits 27:12 Reserved

Bits 11:0 **DEV_ID(11:0)**: *Device Identifier*

This field indicates the device ID. For STM32F10xxx MCU, the device ID is 0x410.

20.6.2 TMC TAP

JTAG ID code

The TAP of the STM32F10xxx TMC (Test Mode Controller) integrates a JTAG ID code which is equal to **0x16410041**.

20.6.3 Cortex-M3 TAP

The TAP of the ARM Cortex-M3 integrates a JTAG ID code. This ID code is the ARM default one and has not been modified. This code is only accessible by the JTAG Debug Port. This code is **0x3BA00477** (corresponds to Cortex-M3 r1p1)

Only the DEV_ID(15:0) should be used for identification by the debugger/programmer tools.

20.6.4 Cortex-M3 JEDEC-106 ID code

The ARM Cortex-M3 integrates a JEDEC-106 ID code. It is located in the 4KB ROM table mapped on the internal PPB bus at address 0xE00FF000_0xE00FFFF.

This code is accessible by the JTAG Debug Port (4 to 5 pins) or by the SW Debug Port (two pins) or by the user software.

20.7 JTAG debug port

A standard JTAG state machine is implemented with a 4-bit Instruction Register (IR) and five Data Registers (for full details, refer to the *Cortex-M3 r1p1 Technical Reference Manual (TRM)*):

Table 74. JTAG debug port data registers

IR(3:0)	Data register	Details
1111	BYPASS [1 bit]	
1110	IDCODE [32 bits]	<i>ID CODE</i> 0x3BA00477 (ARM Cortex-M3 r1p1 ID Code)

Table 74. JTAG debug port data registers

IR(3:0)	Data register	Details
1010	DPACC [35 bits]	<p><i>Debug Port Access Register</i> This initiates a debug port and allows access to a debug port register.</p> <ul style="list-style-type: none"> – When transferring data IN: <ul style="list-style-type: none"> Bits 34:3 = DATA[31:0] = 32-bit data to transfer for a write request Bits 2:1 = A[3:2] = 2-bit address of a debug port register. Bit 0 = RnW = Read request (1) or write request (0). – When transferring data OUT: <ul style="list-style-type: none"> Bits 34:3 = DATA[31:0] = 32-bit data which is read following a read request Bits 2:0 = ACK[2:0] = 3-bit Acknowledge: <ul style="list-style-type: none"> 010 = OK/FAULT 001 = WAIT OTHER = reserved <p>Refer to Table 75 for a description of the A(3:2) bits</p>
1011	APACC [35 bits]	<p><i>Access Port Access Register</i> Initiates an access port and allows access to an access port register.</p> <ul style="list-style-type: none"> – When transferring data IN: <ul style="list-style-type: none"> Bits 34:3 = DATA[31:0] = 32-bit data to shift in for a write request Bits 2:1 = A[3:2] = 2-bit address (sub-address AP registers). Bit 0 = RnW= Read request (1) or write request (0). – When transferring data OUT: <ul style="list-style-type: none"> Bits 34:3 = DATA[31:0] = 32-bit data which is read following a read request Bits 2:0 = ACK[2:0] = 3-bit Acknowledge: <ul style="list-style-type: none"> 010 = OK/FAULT 001 = WAIT OTHER = reserved <p>There are many AP Registers (see AHB-AP) addressed as the combination of:</p> <ul style="list-style-type: none"> – The shifted value A[3:2] – The current value of the DP SELECT register
1000	ABORT [35 bits]	<p><i>Abort Register</i></p> <ul style="list-style-type: none"> – Bits 31:1 = Reserved – Bit 0 = DAPABORT: write 1 to generate a DAP abort.

Table 75. 32-bit debug port registers addressed through the shifted value A[3:2]

Address	A(3:2) value	Description
0x0	00	Reserved
0x4	01	<p>DP CTRL/STAT register. Used to:</p> <ul style="list-style-type: none"> – 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)

Table 75. 32-bit debug port registers addressed through the shifted value A[3:2]

Address	A(3:2) value	Description
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)

20.8 SW debug port

20.8.1 SW protocol introduction

This synchronous serial protocol uses two pins:

- SWCLK: clock from host to target
- SWDIO: bidirectional

The protocol allows two banks of registers (DPACC registers and APACC registers) to be read and written to.

Bits are transferred LSB-first on the wire.

For SWDIO bidirectional management, the line must be pulled-up on the board (100 KΩ recommended by ARM).

Each time the direction of SWDIO changes in the protocol, a turnaround time is inserted where the line is not driven by the host nor the target. By default, this turnaround time is one bit time, however this can be adjusted by configuring the SWCLK frequency.

20.8.2 SW protocol sequence

Each sequence consist of three phases:

1. Packet request (8 bits) transmitted by the host
2. Acknowledge response (3 bits) transmitted by the target
3. Data transfer phase (33 bits) transmitted by the host or the target

Table 76. 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

Table 76. Packet request (8-bits) (continued)

Bit	Name	Description
4:3	A(3:2)	Address field of the DP or AP registers (refer to Table 75)
5	Parity	Single bit parity of preceding bits
6	Stop	0
7	Park	Not driven by the host. Must be read as “1” by the target because of the pull-up

Refer to the *Cortex-M3 r1p1 TRM* for a detailed description of DPACC and APACC registers.

The packet request is always followed by the turnaround time (default 1 bit) where neither the host nor target drive the line.

Table 77. 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 78. 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.

20.8.3 SW-DP state machine (Reset, idle states, ID code)

The State Machine of the SW-DP has an internal ID code which identifies the SW-DP. It follows the JEP-106 standard. This ID code is the default ARM one and is set at **0x1BA01477** (corresponding to Cortex-M3 r1p1).

Note:

Note that the SW-DP state machine is inactive until the target reads this ID code.

- The SW-DP state machine is in RESET STATE either after power-on reset, or after the DP has switched from JTAG to SWD or after the line is high for more than 50 cycles
- The SW-DP state machine is in IDLE STATE if the line is low for at least two cycles after RESET state.
- After RESET state, it is **mandatory** to first enter into an IDLE state AND to perform a READ access of the DP-SW ID CODE register. Otherwise, the target will issue a FAULT acknowledge response on another transactions.

Further details of the SW-DP state machine can be found in the *Cortex-M3 r1p1 TRM* and the *CoreSight Design Kit r1p0 TRM*.

20.8.4 DP and AP read/write accesses

- Read accesses to the DP are not posted: the target response can be immediate (if ACK=OK) or can be delayed (if ACK=WAIT).
- Read accesses to the AP are posted. This means that the result of the access is returned on the next transfer. If the next access to be done is NOT an AP access, then the DP-RDBUFF register must be read to obtain the result.
The READOK flag of the DP-CTRL/STAT register is updated on every AP read access or RDBUFF read request to know if the AP read access was successful.
- The SW-DP implements a write buffer (for both DP or AP writes), that enables it to accept a write operation even when other transactions are still outstanding. If the write buffer is full, the target acknowledge response is “WAIT”. With the exception of IDCODE read or CTRL/STAT read or ABORT write which are accepted even if the write buffer is full.
- Because of the asynchronous clock domains SWCLK and HCLK, two extra SWCLK cycles are needed after a write transaction (after the parity bit) to make the write effective internally. These cycles should be applied while driving the line low (IDLE state)
This is particularly important when writing the CTRL/STAT for a power-up request. If the next transaction (requiring a power-up) occurs immediately, it will fail.

20.8.5 SW-DP registers

Access to these registers are initiated when APnDP=0

Table 79. SW-DP registers

A(3:2)	R/W	CTRLSEL bit of SELECT register	Register	Notes
00	Read		IDCODE	The manufacturer code is not set to ST code. 0x1BA01477 (identifies the SW-DP)
00	Write		ABORT	
01	Read/Write	0	DP-CTRL/STAT	Purpose is to: – request a system or debug power-up – configure the transfer operation for AP accesses – control the pushed compare and pushed verify operations. – read some status flags (overrun, power-up acknowledges)
01	Read/Write	1	WIRE CONTROL	Purpose is to configure the physical serial port protocol (like the duration of the turnaround time)
10	Read		READ RESEND	Enables recovery of the read data from a corrupted debugger transfer, without repeating the original AP transfer.

Table 79. SW-DP registers (continued)

A(3:2)	R/W	CTRLSEL bit of SELECT register	Register	Notes
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

20.8.6 SW-AP registers

Access to these registers are initiated when APnDP=1

There are many AP Registers (see AHB-AP) addressed as the combination of:

- The shifted value A[3:2]
- The current value of the DP SELECT register

20.9 AHB-AP (AHB Access Port) - valid for both JTAG-DP or SW-DP

Features:

- System access is independent of the processor status.
- Either SW-DP or JTAG-DP accesses AHB-AP.
- The AHB-AP is an AHB master into the Bus Matrix. Consequently, it can access all the data buses (Dcode Bus, System Bus, internal and external PPB bus) but the ICode bus.
- Bitband transactions are supported.
- AHB-AP transactions bypass the FPB.

The address of the 32-bits AHP-AP registers are 6-bits wide (up to 64 words or 256 bytes) and consists of:

- a) Bits [8:4] = the bits[7:4] APBANKSEL of the DP SELECT register
- b) Bits [3:2] = the 2 address bits of A(3:2) of the 35-bit packet request for SW-DP.

The AHB-AP of the Cortex-M3 includes 9 x 32-bits registers:

Table 80. Cortex-M3 AHB-AP registers

Address offset	Register name	Notes
0x00	AHB-AP Control and Status Word	Configures and controls transfers through the AHB interface (size, hprot, status on current transfer, address increment type)
0x04	AHB-AP Transfer Address	

Table 80. Cortex-M3 AHB-AP registers (continued)

Address offset	Register name	Notes
0x0C	AHB-AP Data Read/Write	
0x10	AHB-AP Banked Data 0	
0x14	AHB-AP Banked Data 1	
0x18	AHB-AP Banked Data 2	
0x1C	AHB-AP Banked Data 3	
0xF8	AHB-AP Debug ROM Address	Base Address of the debug interface
0xFC	AHB-AP ID Register	

Refer to the *Cortex-M3 r1p1 TRM* for further details.

20.10 Core debug

Core debug is accessed through the core debug registers. Debug access to these registers is by means of the *Advanced High-performance Bus* (AHB-AP) port. The processor can access these registers directly over the internal *Private Peripheral Bus* (PPB).

It consists of 4 registers:

Table 81. 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. This register contains a bit named TRCENA which enable the use of a TRACE.

Note: *Important: these registers are not reset by a system reset. They are only reset by a power-on reset.*

Refer to the *Cortex-M3 r1p1 TRM* for further details.

To Halt on reset, it is necessary to:

- enable the bit0 (VC_CORRESET) of the Debug and Exception Monitor Control Register
- enable the bit0 (C_DEBUGEN) of the Debug Halting Control and Status Register.

20.11 Capability of the debugger host to connect under system reset

The STM32F10xxx MCU reset system comprises the following reset sources:

- POR (Power On Reset) which asserts a RESET at each power-up.
- Internal Watchdog Reset
- Software Reset
- External Reset

The Cortex-M3 differentiates the reset of the debug part (generally PORRESETn) and the other one (SYSRESETn)

This way, it is possible for the debugger to connect under System Reset, programming the Core Debug Registers to halt the core when fetching the reset vector. Then the host can release the system reset and the core will immediately halt without having executed any instructions. In addition, it is possible to program any debug features under System Reset.

Note: *It is highly recommended for the debugger host to connect (set a breakpoint in the reset vector) under system reset.*

20.12 FPB (Flash patch breakpoint)

The FPB unit:

- implements hardware breakpoints
- patches code and data from code space to system space. This feature gives the possibility to correct software bugs located in the Code Memory Space.

The use of a Software Patch or a Hardware Breakpoint is exclusive.

The FPB consists of:

- 2 literal comparators for matching against literal loads from Code Space and remapping to a corresponding area in the System Space.
- 6 instruction comparators for matching against instruction fetches from Code Space. They can be used either to remap to a corresponding area in the System Space or to generate a Breakpoint Instruction to the core.

20.13 DWT (data watchpoint trigger)

The DWT unit consists of four comparators. They are configurable as:

- a hardware watchpoint or
- a trigger to an ETM or
- a PC sampler or
- a data address sampler.

The DWT also provides some means to give some profiling informations. For this, some counters are accessible to give the number of:

- Clock cycle
- Folded instructions
- Load store unit (LSU) operations
- Sleep cycles
- CPI (clock per instructions)
- Interrupt overhead

20.14 ITM (instrumentation trace macrocell)

20.14.1 General description

The ITM is an application-driven trace source that supports *printf* style debugging to trace *Operating System* (OS) and application events, and emits diagnostic system information. The ITM emits trace information as packets which can be generated as:

- **Software trace.** Software can write directly to the ITM stimulus registers to emit packets.
- **Hardware trace.** The DWT generates these packets, and the ITM emits them.
- **Time stamping.** Timestamps are emitted relative to packets. The ITM contains a 21-bit counter to generate the timestamp. The Cortex-M3 clock or the bit clock rate of the *Serial Wire Viewer* (SWV) output clocks the counter.

The packets emitted by the ITM are output to the TPIU (Trace Port Interface Unit). The formatter of the TPIU adds some extra packets (refer to TPIU) and then output the complete packets sequence to the debugger host.

The bit TRCEN of the Debug Exception and Monitor Control Register must be enabled before you program or use the ITM.

20.14.2 Timestamp packets, synchronization and overflow packets

Timestamp packets encode timestamp information, generic control and synchronization. It uses a 21-bit timestamp counter (with possible prescalers) which is reset at each time stamp packet emission. This counter can be either clocked by the CPU clock or the SWV clock.

A synchronization packet consists of 6 bytes equal to 0x80_00_00_00_00_00 which is emitted to the TPIU as 00 00 00 00 00 80 (LSB emitted first).

A synchronization packet is a timestamp packet control. It is emitted at each DWT trigger.

For this, the DWT must be configured to trigger the ITM: the bit CYCCNTENA (bit0) of the DWT Control Register must be set. In addition, the bit2 (SYNCENA) of the ITM Trace Control Register must be set.

Note: If the SYNCENA bit is not set, the DWT generates Synchronization triggers to the TPIU which will send only TPIU synchronization packets and not ITM synchronization packets.

An overflow packet consists is a special timestamp packets which indicates that data has been written but the FIFO was full.

Table 82. Main ITM registers

Address	Register	Details
@E0000FB0	ITM Lock Access	Write 0xC5ACCE55 to unlock Write Access to the other ITM registers
@E0000E80	ITM Trace Control	Bits 31-24 = Always 0
		Bits 23 = Busy
		Bits 22-16 = 7-bits ATB ID which identifies the source of the trace data.
		Bits 15-10 = Always 0
		Bits 9:8 = TSPrescale = Time Stamp Prescaler
		Bits 7-5 = Reserved
		Bit 4 = SWOENA = Enable SWV behavior (to clock the timestamp counter by the SWV clock).
		Bit 3 = DWTENA: Enable the DWT Stimulus
		Bit 2 = SYNCENA: this bit must be to 1 to enable the DWT to generate synchronization triggers so that the TPIU can then emit the synchronization packets.
		Bit 1 = TSENA (Timestamp Enable)
@E0000E40	ITM Trace Privilege	Bit 0 = ITMENA: Global Enable Bit of the ITM
		Bit 3: mask to enable tracing ports31:24
		Bit 2: mask to enable tracing ports23:16
		Bit 1: mask to enable tracing ports15:8
@E0000E00	ITM Trace Enable	Bit 0: mask to enable tracing ports7:0
		Each bit enables the corresponding Stimulus port to generate trace.
@E0000000-E000007C	Stimulus Port Registers 0-31	Write the 32-bits data on the selected Stimulus Port (32 available) to be traced out.

Example of configuration

To output a simple value to the TPIU:

- Configure the TPIU and assign TRACE I/Os by configuring the DBGMCU_CR (refer to [Section 20.16.2: TRACE pin assignment](#) and [Section 20.15.3: Debug MCU configuration register](#))
- Write 0xC5ACCE55 to the ITM Lock Access Register to unlock the write access to the ITM registers
- Write 0x00010005 to the ITM Trace Control Register to enable the ITM with Sync enabled and an ATB ID different from 0x00
- Write 0x1 to the ITM Trace Enable Register to enable the Stimulus Port 0
- Write 0x1 to the ITM Trace Privilege Register to unmask stimulus ports 7:0
- Write the value to output in the Stimulus Port Register 0: this can be done by software (using a printf function)

20.15 MCU debug component (MCUDBG)

The MCU debug component helps the debugger provide support for:

- Low-power modes
- Clock control for timers, watchdog and bxCAN during a breakpoint
- Control of the trace pins assignment

20.15.1 Debug support for low-power modes

To enter low-power mode, the instruction WFI or WFE must be executed.

The MCU implements several low-power modes which can either deactivate the CPU clock or reduce the power of the CPU.

The core does not allow FCLK or HCLK to be turned off during a debug session. As these are required for the debugger connection, during a debug, they must remain active. The MCU integrates special means to allow the user to debug software in low-power modes.

For this, the debugger host must first set some debug configuration registers to change the low-power mode behavior:

- In Sleep mode, DBG_SLEEP bit of DBGMCU_CR register must be previously set by the debugger. This will feed HCLK with the same clock that is provided to FCLK (system clock previously configured by the software).
- In STOP mode, the bit DBG_STOP must be previously set by the debugger. This will enable the internal RC oscillator clock to feed FCLK and HCLK in STOP mode.

20.15.2 Debug support for timers, watchdog, bxCAN and I²C

During a breakpoint, it is necessary to choose how the counter of timers and watchdog should behave:

- they can continue to count inside a breakpoint. This is usually required when a PWM is controlling a motor, for example.
- they can stop to count inside a breakpoint. This is required for watchdog purposes.

For the bxCAN, the user can choose to block the update of the receive register during a breakpoint.

For the I²C, the user can choose to block the SMBUS timeout during a breakpoint.

20.15.3 Debug MCU configuration register

This register allows the configuration of the MCU under DEBUG. This concerns:

- Low-power mode support
- Timer and Watchdog counters support
- bxCAN communication support
- Trace pin assignment

This DBGMCU_CR is mapped on the External PPB bus at address 0xE0042000

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.

DBGMCU_CR

Address: 0xE0042004

Only 32-bit access supported

POR Reset: 0x00000000 (not reset by system reset)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reserved															DBG_I2 C2_SMB US_TIM EOUT	
Res.															rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
DBG_I2 C1_SM BUS_TI MEOUT	DBG_ CAN_ STOP	DBG_ TIM4_ STOP	DBG_ TIM3_ STOP	DBG_ TIM2_ STOP	DBG_ TIM1_ STOP	DBG_ WWDG_ STOP	DBG_ IWDG_ STOP	TRACE_ MODE [1:0]	TRACE_ IOEN	Reserved	DBG_ STANDB Y	DBG_ STOP	DBG_ SLEEP			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bits 31:17 Reserved, must be kept cleared.

Bit 16 **DBG_I2C2_SMBUS_TIMEOUT** *SMBUS timeout mode stopped when Core is halted*

- 0: Same behavior as in normal mode.
- 1: The SMBUS timeout is frozen

Bit 15 **DBG_I2C1_SMBUS_TIMEOUT** *SMBUS timeout mode stopped when Core is halted*

- 0: Same behavior as in normal mode.
- 1: The SMBUS timeout is frozen.

Bit 14 **DBG_CAN_STOP:** *Debug CAN stopped when Core is halted*

- 0: Same behavior as in normal mode.
- 1: The CAN receive registers are frozen.

Bits 13:10 **DBG_TIMx_STOP:** *Regular data. x=4..1*

- 0: The clock of the involved Timer Counter is fed even if the core is halted.
- 1: The clock of the involved Timer counter is stopped when the core is halted.

Bit 9 **DBG_WWDG_STOP:** *Debug Window Watchdog stopped when Core is halted*

- 0: The Window Watchdog Counter clock continues even if the core is halted.
- 1: The Window Watchdog Counter clock is stopped when the core is halted.

Bit 8 **DBG_IWDG_STOP:** *Debug Independent Watchdog stopped when Core is halted*

- 0: The Watchdog counter clock continues even if the core is halted.
- 1: The Watchdog counter clock is stopped when the core is halted.

Bits 7:5 **TRACE_MODE[1:0] and TRACE_IOEN:** *Trace Pin Assignment Control*

- With *TRACE_IOEN=0*:
TRACE_MODE=xx: TRACE pins not assigned (default state)
- With *TRACE_IOEN=1*:
TRACE_MODE=00: TRACE pin assignment for Asynchronous Mode
TRACE_MODE=01: TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 1
TRACE_MODE=10: TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 2
TRACE_MODE=11: TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 4

Bit 4:3 Reserved, must be kept cleared.

Bit 2 `DBG_STANDBY`: Debug Standby mode

0: (FCLK=Off, HCLK=Off) The whole digital part is unpowered.

From software point of view, exiting from Standby is identical than fetching reset vector (except a few status bit indicated that the MCU is resuming from Standby)

1: (FCLK=On, HCLK=On) In this case, the digital part is not unpowered and FCLK and HCLK are provided by the internal RC oscillator which remains active. In addition, the MCU generate a system reset during Standby mode so that exiting from Standby is identical than fetching from reset

Bit 1 `DBG_STOP`: Debug Stop Mode

0: (FCLK=Off, HCLK=Off) In STOP mode, the clock controller disables all clocks (including HCLK and FCLK). When exiting from STOP mode, the clock configuration is identical to the one after RESET (CPU clocked by the 8 MHz internal RC oscillator (HSI)). Consequently, the software must reprogram the clock controller to enable the PLL, the Xtal, etc.

1: (FCLK=On, HCLK=On) In this case, when entering STOP mode, FCLK and HCLK are provided by the internal RC oscillator which remains active in STOP mode. When exiting STOP mode, the software must reprogram the clock controller to enable the PLL, the Xtal, etc. (in the same way it would do in case of `DBG_STOP`=0)

Bit 0 `DBG_SLEEP`: Debug Sleep Mode

0: (FCLK=On, HCLK=Off) In Sleep mode, FCLK is clocked by the system clock as previously configured by the software while HCLK is disabled.

In Sleep mode, the clock controller configuration is not reset and remains in the previously programmed state. Consequently, when exiting from Sleep mode, the software does not need to reconfigure the clock controller.

1: (FCLK=On, HCLK=On) In this case, when entering Sleep mode, HCLK is fed by the same clock that is provided to FCLK (system clock as previously configured by the software).

20.16 TPIU (trace port interface unit)

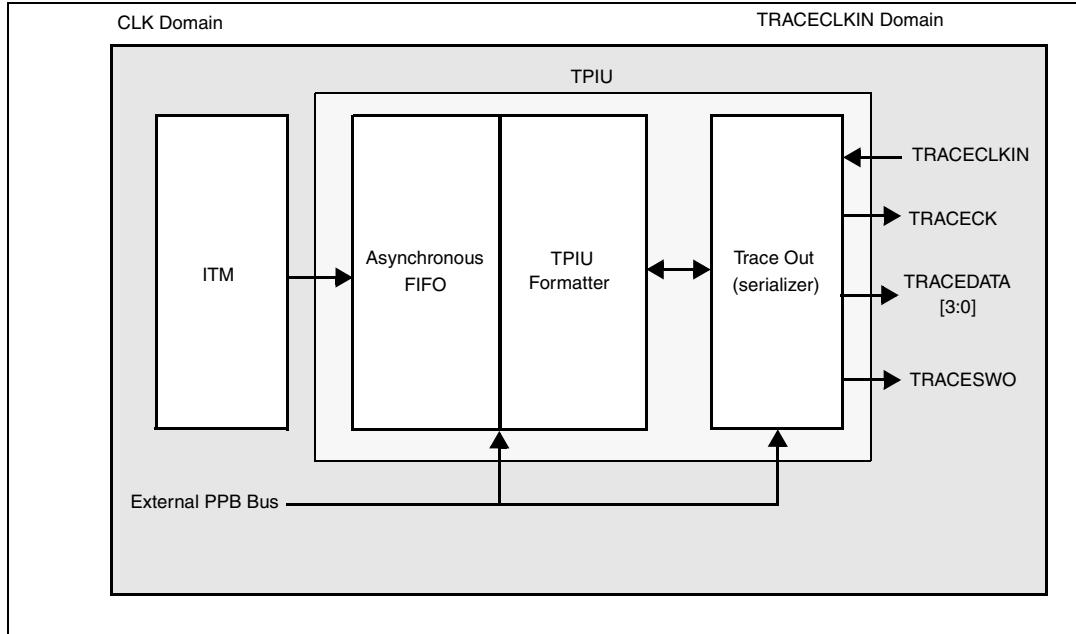
20.16.1 Introduction

The TPIU acts as a bridge between the on-chip trace data from the ITM.

The output data stream encapsulates the trace source ID, that is then captured by a *Trace Port Analyzer* (TPA).

The core embeds a simple TPIU, especially designed for low-cost debug (consisting of a special version of the CoreSight TPIU).

The TPIU only supports ITM debug trace which is a *limited trace* as it only outputs information coming from the ITM.

Figure 188. TPIU block diagram

20.16.2 TRACE pin assignment

- Asynchronous mode

The asynchronous mode requires 1 extra pin and is available on all packages. It is only available if using Serial Wire mode (not in JTAG mode).

Table 83. Asynchronous TRACE pin assignment

TPUI pin name	Trace synchronous mode		STM32F10xxx pin assignment
	Type	Description	
TRACESWO	O	TRACE Async Data Output	PB3

- Synchronous mode

The synchronous mode requires from 2 to 6 extra pins depending on the data trace size and is only available in the larger packages. In addition it is available in JTAG mode and in Serial Wire mode and provides better bandwidth output capabilities than asynchronous trace.

Table 84. Synchronous TRACE pin assignment

TPUI pin name	Trace synchronous mode		STM32F10xxx pin assignment
	Type	Description	
TRACECK	O	TRACE Clock	PE2
TRACED[3:0]	O	TRACE Sync Data Outputs Can be 1, 2 or 4.	PE[6:3]

TPUI TRACE pin assignment

By default, these pins are NOT assigned. They can be assigned by setting the IOTRACEN and IOTRACEMODE bits of the **MCU Debug Component Configuration Register**. This configuration has to be done by the debugger host.

In addition, the number of pins to assign depends on the trace configuration (asynchronous or synchronous).

- **Asynchronous mode:** 1 extra pin is needed
- **Synchronous mode:** from 2 to 5 extra pins are needed depending on the size of the data trace port register (1, 2 or 4):
 - TRACECK
 - TRACED(0) if port size is configured to 1, 2 or 4
 - TRACED(1) if port size is configured to 2 or 4
 - TRACED(2) if port size is configured to 4
 - TRACED(3) if port size is configured to 4

To assign the TRACE pin, the debugger host must program the bits TRACE_IOEN and TRACE_MODE[1:0] of the Debug MCU configuration Register (DBGMCU_CR). By default the TRACE pins are not assigned.

This register is mapped on the external PPB and is reset by the PORESET (and not by the SYSTEM reset). It can be written by the debugger under SYSTEM reset.

Table 85. Flexible TRACE pin assignment

DBGMCU_CR register		Pins assigned for:	TRACE I/O pin assigned					
			PB3 / JTDO/ TRACES WO	PE2 / TRACE CK	PE3 / TRACE D[0]	PE4 / TRACE D[1]	PE5 / TRACE D[2]	PE6 / TRACE D[3]
0	XX	No Trace (default state)	Released (1)	TRACES WO	Released (usable as GPIO)			
1	00	Asynchronous Trace						
1	01	Synchronous Trace 1 bit	TRACE CK	TRACE D[0]	TRACE D[1]			
1	10	Synchronous Trace 2 bit	TRACE CK	TRACE D[0]	TRACE D[2] TRACE D[3]			
1	11	Synchronous Trace 4 bit	TRACE CK	TRACE D[0]	TRACE D[1]	TRACE D[2] TRACE D[3]		

(1) When Serial Wire mode is used, it is released. But when JTAG is used, it is assigned to JTDO.

Note:

By default, the TRACECLKIN input clock of the TPIU is tied to GND. It is assigned to HCLK two clock cycles after the bit TRACE_IOEN has been set.

The debugger must then program the Trace Mode by writing the PROTOCOL[1:0] bits in the SPP_R (Selected Pin Protocol) register of the TPIU.

- PROTOCOL=00: Trace Port Mode (synchronous)
- PROTOCOL=01 or 10: Serial Wire (Manchester or NRZ) Mode (asynchronous mode). Default state is 01

It then also configures the TRACE port size by writing the bits [3:0] in the CPSPS_R (Current Sync Port Size Register) of the TPIU:

- 0x1 for 1 pin (default state)
- 0x2 for 2 pins
- 0x8 for 4 pins

20.16.3 TPUI formatter

The formatter protocol outputs data in 16-byte frames:

- seven bytes of data
- eight bytes of mixed-use bytes consisting of:
 - 1 bit (LSB) to indicate it is a DATA byte ('0') or an ID byte ('1').
 - 7 bits (MSB) which can be data or change of source ID trace.
- one byte of auxiliary bits where each bit corresponds to one of the eight mixed-use bytes:
 - if the corresponding byte was a data, this bit gives bit0 of the data.
 - if the corresponding byte was an ID change, this bit indicates when that ID change takes effect.

Note: Refer to the ARM CoreSight Architecture Specification v1.0 (ARM IHI 0029B) for further information

Use of the formatter for STM32F10xxx MCU

For STM32F10xxx MCU, there is only one TRACE source (the ITM). But the formatter can not be disabled and must be used in bypass mode because the TRACECTL pin is not assigned. This way, the Trace Port Analyzer can decode part of the formatter protocol to determine the position of the trigger.

20.16.4 TPUI frame synchronization packets

The TPUI can generate two types of synchronization packets:

- The Frame Synchronization packet (or Full Word Synchronization packet)

It consists of the word: 0x7F_FF_FF_FF (LSB emitted first). This sequence can not occur at any other time provided that the ID source code 0x7F has not been used.

It is output periodically **between** frames.

In continuous mode, the TPA must discard all these frames once a synchronization frame has been found.

- The Half-Word Synchronization packet

It consists of the half word: 0x7F_FF (LSB emitted first).

It is output periodically **between or within** frames.

These packets are only generated in continuous mode and enable the TPA to detect that the TRACE port is in IDLE mode (no TRACE to be captured). When detected by the TPA, it must be discarded.

20.16.5 Emission of synchronization frame packet

There is no Synchronization Counter register implemented in the TPUI of the core.

Consequently, the synchronization trigger can only be generated by the **DWT**. Refer to the registers DWT Control Register (bits SYNCTAP[11:10]) and the DWT Current PC Sampler Cycle Count Register.

The TPUI Frame synchronization packet (0x7F_FF_FF_FF) is emitted:

- after each TPUI reset release. This reset is synchronously released with the rising edge of TRACECLKIN clock. This means that this packet is emitted once the bit IO_TRACEN of the DBGMCU_CFG register has been set. In this case, the word 0x7F_FF_FF_FF is not followed by any formatted packet.
- at each DWT trigger (assuming DWT has been previously configured). Two cases occur:
 - If the bit SYNENA of the ITM is reset, only the word 0x7F_FF_FF_FF is emitted without any formatted stream which follows.
 - If the bit SYNENA of the ITM is set, then the ITM synchronization packets will follow (0x80_00_00_00_00_00), formatted by the TPUI (trace source ID added).

20.16.6 Synchronous mode

The trace data output size can be configured to 4, 2 or 1 pin: TRACED(3:0)

The output clock is output to the debugger (TRACECK)

Here, TRACECLKIN is driven internally and is connected to HCLK only when TRACE is used.

Note: *In this synchronous mode, it is not required to provide a stable clock frequency.*

The TRACE I/Os (including TRACECK) are driven by the rising edge of TRACLKIN (equal to HCLK). Consequently, the output frequency of TRACECK is equal to HCLK/2.

20.16.7 Asynchronous mode

This is a low cost alternative to output the trace using only 1 pin: this is the asynchronous output pin TRACESWO. Obviously there is a limited bandwidth.

TRACESWO is multiplexed with JTDO when using the SW-DP pin. This way, this functionality is available in all STM32F10xxx packages.

This asynchronous mode requires a constant frequency for TRACECLKIN. For the standard UART (NRZ) capture mechanism, 5% accuracy is needed. The Manchester encoded version is tolerant up to 10%.

20.16.8 TRACECLKIN connection inside STM32F10xxx

In STM32F10xxx, this TRACECLKIN input is internally connected to HCLK. This means that when in asynchronous trace mode, the application is restricted to use time frames where the CPU frequency is stable.

Note: **Important:** *when using asynchronous trace: it is important to be aware that:*

The default clock of the STM32F10xxx MCU is the internal RC oscillator. Its frequency under reset is different from the one after reset release. This is because the RC calibration is the default one under system reset and is updated at each system reset release.

Consequently, the Trace Port Analyzer (TPA) should not enable the trace (with the bit IOTRACEN) under system reset, because a Synchronization Frame Packet will be issued with a different bit time than trace packets which will be transmitted after reset release.

20.16.9 TPIU registers

The TPIU APB registers can be read and written only if the bit TRCENA of the Debug Exception and Monitor Control Register (DEMCR) is set. Otherwise, the registers are read as zero (the output of this bit enables the PCLK of the TPIU).

Table 86. Important TPIU registers

Address	Register	Description
0xE0040004	Current port size	<p>Allows the trace port size to be selected:</p> <ul style="list-style-type: none"> Bit 0: Port size = 1 Bit 1: Port size = 2 Bit 2: Port size = 3, not supported Bit 3: Port Size = 4 <p>Only 1 bit must be set. By default, the port size is one bit. (0x00000001)</p>
0xE00400F0	Selected pin protocol	<p>Allows the Trace Port Protocol to be selected:</p> <ul style="list-style-type: none"> Bit1:0= 00: Sync Trace Port Mode 01: Serial Wire Output - manchester (default value) 10: Serial Wire Output - NRZ 11: reserved
0xE0040304	Formatter and flush control	<p>Bit 31-9 = always '0'</p> <p>Bit 8 = TrigIn = always '1' to indicate that triggers are indicated</p> <p>Bit 7-4 = always 0</p> <p>Bit 3-2 = always 0</p> <p>Bit 1 = EnFCont. In Sync Trace mode (Select_Pin_Protocol register bit1:0=00), this bit is forced to '1': the formatter is automatically enabled in continuous mode. In asynchronous mode (Select_Pin_Protocol register bit1:0 <> 00), this bit can be written to activate or not the formatter.</p> <p>Bit 0 = always 0</p> <p>The resulting default value is 0x102</p> <p>Note: In synchronous mode, because the TRACECTL pin is not mapped outside the chip, the formatter is always enabled in continuous mode -this way the formatter inserts some control packets to identify the source of the trace packets).</p>
0xE0040300	Formatter and flush status	Not used in Cortex-M3, always read as 0x00000008

20.16.10 Example of configuration

- Set the bit TRCENA in the Debug Exception and Monitor Control Register (DEMCR)
- Write the TPIU Current Port Size Register to the desired value (default is 0x1 for a 1-bit port size)
- Write TPIU Formatter and Flush Control Register to 0x102 (default value)
- Write the TPIU Select Pin Protocol to select the sync or async mode. Example: 0x2 for async NRZ mode (UART like)
- Write the DBGMCU Control Register to 0x20 (bit IO_TRACEN) to assign TRACE I/Os for async mode. A TPIU Sync packet is emitted at this time (FF_FF_FF_7F)
- Configure the ITM and write the ITM Stimulus register to output a value

20.17 DBG register map

The following table summarizes the Debug registers.

Table 87. DBG - register map and reset values

Addr.	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0xE0042000	DBGMCU_IDCODE	REV_ID																Reserved		DEV_ID															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
0xE0042004	DBGMCU_CR	Reserved																DBG_I2C2_SMBUS_TIMEOUT	DBG_I2C1_SMBUS_TIMEOUT	DBG_CAN_STOP	DBG_TIM4_STOP	DBG_TIM3_STOP	DBG_TIM2_STOP	DBG_TIM1_STOP	DBG_WWDG_STOP	DBG_WWDG_STOP	TRACE_MODE [1:0]	TRACE_JDEN	Reserved	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				

Revision history

Table 88. Document revision history

Date	Revision	Changes
19-Oct-2007	1	<p>Document reference number changed from UM0306 to RM008. The changes below were made with reference to revision 1 of 01-Jun-2007 of UM0306.</p> <p>EXTSEL[2:0] and JEXTSEL[2:0] removed from Table 49: ADC pins on page 350 and V_{REF+} range modified in Remarks column.</p> <p>Notes added to Section 16.4.9 on page 354, Section 16.10.2 on page 361, Section 16.10.7 on page 364 and Section 16.10.9 on page 365.</p> <p>SPI_CR2 corrected to SPI_CR1 in 1 clock and 1 bidirectional data wire on page 420.</p> <p>f_{CPU} frequency changed to f_{PCLK} in Section 18.2: Main features on page 414.</p> <p>Section 18.3.6: CRC calculation on page 421 and Section 18.3.7: SPI communication using DMA (direct memory addressing) on page 422 modified.</p> <p>Note added to bit 13 description changed in Section 18.4.1: SPI Control Register 1 (SPI_CR1) on page 424. Note for bit 4 modified in Section 18.4.3: SPI status register (SPI_SR) on page 428.</p> <p>On 64-pin packages on page 34 modified.</p> <p>Section 5.3.2: Using OSC_IN/OSC_OUT pins as GPIO ports PD0/PD1 on page 87 updated.</p> <p>Description of SRAM at address 0x4000 6000 modified in Figure 2: Memory map on page 26 and Table 1: Register boundary addresses.</p> <p>Note added to Section 17.2: Main features on page 383 and Section 14.2: Main features on page 274.</p> <p>Figure 3: Power supply overview and On 100-pin packages modified.</p> <p>Formula added to Bits 25:24 description in CAN bit timing register (CAN_BTR) on page 303.</p> <p>Section 7.3: Functional description on page 108 modified.</p> <p>Example of configuration on page 486 modified.</p> <p>MODEx[1:0] bit definitions corrected in Section 5.2.2: Port configuration register high (GPIOx_CRH) (x=A..E) on page 83.</p> <p>Downcounting mode on page 155 modified.</p> <p>Figure 54: Output stage of capture/compare channel (channel 4) on page 166 and Figure 56: Output compare mode, toggle on OC1 modified.</p> <p>OCx output enable conditions modified in Section 12.4.10: PWM mode on page 170.</p> <p>Section 12.4.19: TIM1 and external trigger synchronization on page 185 title changed.</p> <p>CC1S, CC2S, CC3S and CC4S definitions modified for (1, 1) bit setting modified in Section 12.5.7: Capture/compare mode register 1 (TIM1_CCMR1) and Section 12.5.8: Capture/compare mode register 2 (TIM1_CCMR2).</p> <p>CC1S, CC2S, CC3S and CC4S definitions for (1, 1) bit setting modified in Section 13.5.7: Capture/compare mode register 1 (TIMx_CCMR1) and Section 13.5.8: Capture/compare mode register 2 (TIMx_CCMR2).</p> <p>AFIO_EVCR pins modified in Table 26: AFIO register map and reset values on page 96.</p> <p>Section 12.4.6: Input capture mode on page 166 modified.</p>

Table 88. Document revision history (continued)

Date	Revision	Changes
19-Oct-2007 continued	1 continued	<p><i>Figure 87: Counter timing diagram, internal clock divided by 1, TIMx_ARR=0x6 and Figure 102: Output compare mode, toggle on OC1.</i> modified.</p> <p>CKD definition modified in <i>Section 13.5.1: Control register 1 (TIMx_CR1)</i>. Bit 8 and Bit 9 added to <i>Section 9.5.2: RTC clock calibration register (BKP_RTCCR)</i>.</p> <p>Bit 15 and Bit 16 added to <i>DBGMCU_CR</i> on page 488. <i>Section 15.6: I2C debug mode</i> on page 335 added.</p> <p>Stop and Standby modified in <i>Table 4: Low-power mode summary</i>. <i>Table 6: Sleep-on-exit</i> modified. <i>Debug mode</i> on page 41 modified.</p> <p>HSITRIM[4:0] bit description modified in <i>Section 4.3.1: Clock control register (RCC_CR)</i>. Note modified in MCO description in <i>Section 4.3.2: Clock configuration register (RCC_CFGR)</i>.</p> <p>RCC_CR row modified in <i>Table 10: RCC - register map and reset values</i>. Bits 15:0 description modified in <i>Section 5.2.6: Port bit reset register (GPIOx_BRR) (x=A..E)</i>. <i>Embedded boot loader</i> on page 32 added.</p> <p><i>Figure 9, Figure 11, Figure 12, Figure 13 and Figure 14</i> modified.</p> <p><i>Section 2.3.4: Embedded Flash memory</i> on page 29 modified.</p> <p>REV_ID bit description added to <i>DBGMCU_IDCODE</i> on page 476.</p> <p>Reset value modified in <i>Section 4.3.1: Clock control register (RCC_CR)</i> and HSITRIM[4:0] description modified.</p> <p><i>Section 5.1.1</i> on page 77 modified. Bit definitions modified in <i>Section 5.2: GPIO register description</i> on page 82. Wakeup latency description modified in <i>Table 7: Stop mode</i>.</p> <p><i>Clock control register (RCC_CR)</i> reset value modified.</p> <p>Note added in ASOS and ASOE bit descriptions in <i>9.5.2</i> on page 134.</p> <p><i>Section 20.15.2: Debug support for timers, watchdog, bxCAN and I2C</i> modified. <i>Table 87: DBG - register map and reset values</i> updated.</p> <p><i>Section 17.6.3: Buffer descriptor table</i> clarified.</p> <p><i>Center-aligned mode (up/down counting)</i> on page 157 and <i>Center-aligned mode (up/down counting)</i> on page 223 updated.</p> <p><i>Figure 58: Center-aligned PWM waveforms (ARR=8)</i> on page 172 and <i>Figure 104: Center-aligned PWM waveforms (ARR=8)</i> on page 236 modified.</p> <p>RSTCAL description modified in <i>Section 16.13.3: ADC control register 2 (ADC_CR2)</i>.</p> <p>Note changed below <i>Table 34: Watchdog timeout period (with 40 kHz input clock)</i>. Note added below <i>Figure 7: Clock tree</i>.</p> <p>ADC conversion time modified in <i>Section 16.2: Main features</i>.</p> <p><i>Auto-injection</i> on page 354 updated.</p> <p>Note added in <i>Section 16.10.9: Combined injected simultaneous + interleaved</i>. Note added to <i>Section 5.3.2: Using OSC_IN/OSC_OUT pins as GPIO ports PD0/PD1</i>. Small text changes. Internal LSI RC frequency changed from 32 to 40 kHz. <i>Table 34: Watchdog timeout period (with 40 kHz input clock)</i> updated. Option byte addresses corrected in <i>Figure 2: Memory map</i> and <i>Table 2: Flash module organization</i>. Information block organization modified in <i>Section 2.3.4: Embedded Flash memory</i>.</p> <p>External event that trigger ADC conversion is EXTI line instead of external interrupt (see <i>Section 16: Analog-to-digital converter (ADC)</i>).</p> <p><i>Appendix A: Important notes</i> on page 500 added.</p>

Table 88. Document revision history (continued)

Date	Revision	Changes
20-Nov-2007	2	<p><i>Figure 165: USART block diagram</i> modified. Procedure modified in <i>Character reception on page 439</i>. In <i>Section 19.3.5: Fractional baud rate generation</i>:</p> <ul style="list-style-type: none"> – Equation legend modified – <i>Table 68: Error calculation for programmed baud rates</i> modified – Note added <p>Small text changes. In <i>CAN bit timing register (CAN_BTR) on page 303</i>, bit 15 is reserved. Flash memory organization corrected, <i>Table 2: Flash module organization</i> modified in <i>Section 2.3.4: Embedded Flash memory</i>. Note added below <i>Figure 3: Power supply overview</i> in <i>Section 3.1: Power supplies</i>. RTCSEL[1:0] bit description modified in <i>Section 4.3.9: Backup domain control register (RCC_BDCR)</i>. Names of bits [0:2] corrected for RCC_APB1RSTR and RCC_APB1ENR in <i>Table 10: RCC - register map and reset values</i>. Impedance value specified in <i>A.4: Voltage glitch on ADC input 0 on page 500</i>. In <i>Section 18.4.1: SPI Control Register 1 (SPI_CR1)</i>, BR[2:0] description corrected. Prescaler buffer behavior specified when an update event occurs (see <i>upcounting mode on page 218</i>, <i>Downcounting mode on page 221</i> and <i>Center-aligned mode (up/down counting) on page 223</i>). AWDCH[4:0] modified in <i>Section 16.13.2: ADC control register 1 (ADC_CR1)</i> and bits [26:24] are reserved in <i>Section 16.13.4: ADC sample time register 1 (ADC_SMPR1)</i>. CAN_BTR bit 8 is reserved in <i>Table 45: bxCAN - register map and reset values</i>. <i>CAN master control register (CAN_MCR) on page 293</i> corrected. V_{REF+} range corrected in <i>Table 49: ADC pins</i> and in <i>On 100-pin packages on page 34</i>. <i>Start condition on page 324</i> updated. Note removed in <i>Table 13: BXCAN alternate function remapping</i>. Note added in <i>Table 16: Timer 4 alternate function remapping</i>. In <i>Section 5.4.2: AF remap and debug I/O configuration register (AFIO_MAPR)</i>, bit definition modified for USART2_REMAP = 0. In <i>Section 5.4.3: External interrupt configuration register 1 (AFIO_EXTICR1)</i>, bit definition modified for SPI1_REMAP = 0. In <i>Table 86: Important TPIU registers</i>, at 0xE0040004, bit2 set is not supported. TRACE port size setting corrected in <i>TPUI TRACE pin assignment on page 491</i>. <i>Figure 9, Figure 11, Figure 12, Figure 13</i> and <i>Figure 14</i> modified. <i>Figure 10: Basic structure of a five-volt tolerant I/O port bit added</i>. <i>Table 5.3.1: Using OSC32_IN/OSC32_OUT pins as GPIO ports PC14/PC15 on page 87</i> added. Bit descriptions modified in <i>Section 8.4.5</i> and <i>Section 8.4.6</i>. JTAG ID code corrected in <i>Section 20.6.2: TMC TAP on page 477</i>. Modified: <i>Section 11.2: Main features</i>, <i>Section 9.2: Features</i>, <i>Section 9.3: Tamper detection</i>, <i>Section 9.4: RTC calibration</i>, <i>Section 17.4: Functional description</i>, <i>Controlling the downcounter: on page 145</i>, <i>Section 3.1.2: Battery backup domain</i>, <i>Section 8.1: Introduction</i>. ASOE bit description modified in <i>Section 9.5.2: RTC clock calibration register (BKP_RTCCR)</i>.</p>

Table 88. Document revision history (continued)

Date	Revision	Changes
08-Feb-2008	3	<p><i>Figure 3: Power supply overview on page 33</i> modified.</p> <p><i>Section 4.1.2: Power reset on page 47</i> modified.</p> <p><i>Section 4.2: Clocks on page 47</i> modified.</p> <p>Definition of Bits 26:24 modified in <i>Section 5.4.2: AF remap and debug I/O configuration register (AFIO_MAPR) on page 92</i>.</p> <p>AFIO_EVCR bits corrected in <i>Table 26: AFIO register map and reset values on page 96</i>.</p> <p>Number of maskable interrupt channels modified in <i>Section 6.1: Nested vectored interrupt controller (NVIC) on page 97</i>.</p> <p><i>Section 7.3.5: Interrupts on page 111</i> added. Small text changes.</p> <p>Examples modified in <i>Figure 64: 6-step generation, COM example (OSSR=1) on page 178</i>.</p> <p><i>Table 38: Output control bits for complementary OCx and OCxN channels with break feature on page 206</i> modified.</p> <p>Register names modified in <i>Section 14.8.3: CAN filter registers on page 310</i>.</p> <p>Small text change in <i>Section 15.4.2: I2C master mode on page 324</i>.</p> <p>Bits 5:0 frequency description modified in <i>Section 15.7.2: Control register 2 (I2C_CR2) on page 338</i>.</p> <p><i>Section 17.4.1: Description of USB blocks on page 385</i> modified.</p> <p><i>Section 18.3.4: Simplex communication on page 420</i> modified.</p> <p><i>Section 18.3.6: CRC calculation on page 421</i> modified.</p> <p>Note added in <i>BUSY flag on page 421</i>.</p> <p><i>Section 18.3.9: Disabling the SPI on page 423</i> added.</p> <p>Appendix A: Important notes, removed.</p>

Index

A

ADC_CR1	369
ADC_CR2	371
ADC_DR	380
ADC_HTR	376
ADC_JDRx	380
ADC_JOFRx	376
ADC_JSQR	379
ADC_LTR	377
ADC_SMPR1	374
ADC_SMPR2	375
ADC_SQR1	377
ADC_SQR2	378
ADC_SQR3	378
ADC_SR	368
AFIO_EVCR	91
AFIO_EXTICR1	94
AFIO_EXTICR2	94
AFIO_EXTICR3	95
AFIO_EXTICR4	95
AFIO_MAPR	92

B

BKP_CR	135
BKP_CSR	136
BKP_DRx	134
BKP_RTCCR	134

C

CAN_BTR	303
CAN_ESR	302
CAN_FA1R	312
CAN_FFA1R	311
CAN_FiRx	313
CAN_FM1R	310
CAN_FMR	310
CAN_FS1R	311
CAN_IER	300
CAN_MCR	293
CAN_MSR	295
CAN_RDHxR	309
CAN_RDLxR	308
CAN_RDTxR	308
CAN_RF0R	298
CAN_RF1R	299
CAN_RIxR	307

CAN_TDHzR	306
CAN_TDLxR	306
CAN_TDTxR	305
CAN_TIxR	304
CAN_TSRI	296

D

DBGMCU_CR	488
DBGMCU_IDCODE	476
DMA_CCRx	116
DMA_CMARx	118
DMA_CNDTRx	117
DMA_CPARx	118
DMA_IFCR	115
DMA_ISR	113

E

EXTI_EMR	103
EXTI_FTSR	104
EXTI_IMR	103
EXTI_PR	105
EXTI_RTSR	104
EXTI_SWIER	105

G

GPIOx_BRR	85
GPIOx_BSRR	85
GPIOx_CRH	83
GPIOx_CRL	82
GPIOx_IDR	84
GPIOx_LCKR	86
GPIOx_ODR	84

I

I2C_CCR	345
I2C_CR1	335
I2C_CR2	338
I2C_DR	340
I2C_OAR1	339
I2C_OAR2	339
I2C_SR1	341
I2C_SR2	344
I2C_TRISE	346
IWDG_KR	140
IWDG_PR	140

IWDG_RLR 141
 IWDG_SR 142

P

PWR_CR 43
 PWR_CSR 44

R

RCC_AHBENR 65
 RCC_APB1ENR 68
 RCC_APB1RSTR 63
 RCC_APB2ENR 66
 RCC_APB2RSTR 61
 RCC_BDCR 70
 RCC_CFGR 55
 RCC_CIR 58
 RCC_CR 53
 RCC_CSR 71
 RTC_ALRH 131
 RTC_ALRL 131
 RTC_CNTH 130
 RTC_CNTL 130
 RTC_CRH 125
 RTC_CRL 126
 RTC_DIVH 129
 RTC_DIVL 129
 RTC_PRLH 128
 RTC_PRLL 128

S

SPI_CR1 424
 SPI_CR2 427
 SPI_CRCPR 429
 SPI_DR 429
 SPI_RXCRCR 429
 SPI_SR 428
 SPI_TXCRCR 430

T

TIM1_ARR 207
 TIM1_BDTR 210
 TIM1_CCER 204
 TIM1_CCMR1 200
 TIM1_CCMR2 203
 TIM1_CCR1 208
 TIM1_CCR2 209
 TIM1_CCR3 209
 TIM1_CCR4 210
 TIM1_CNT 207

TIM1_CR1 189
 TIM1_CR2 191
 TIM1_DCR 212
 TIM1_DIER 196
 TIM1_DMAR 213
 TIM1_EGR 199
 TIM1_PSC 207
 TIM1_RCR 208
 TIM1_SMCR 193
 TIM1_SR 197
 TIMx_ARR 268
 TIMx_CCER 267
 TIMx_CCMR1 262
 TIMx_CCMR2 265
 TIMx_CCR1 269
 TIMx_CCR2 269
 TIMx_CCR3 270
 TIMx_CCR4 270
 TIMx_CNT 268
 TIMx_CR1 252
 TIMx_CR2 254
 TIMx_DCR 271
 TIMx_DIER 258
 TIMx_DMAR 271
 TIMx_EGR 261
 TIMx_PSC 268
 TIMx_SMCR 255
 TIMx_SR 259

U

USART_BRR 461
 USART_CR1 462
 USART_CR2 464
 USART_CR3 466
 USART_DR 461
 USART_GTPR 468
 USART_SR 459
 USB_ADDRn_RX 410
 USB_ADDRn_TX 409
 USB_BTABLE 404
 USB_CNTR 398
 USB_COUNTn_RX 411
 USB_COUNTn_TX 410
 USB_DADDR 404
 USB_EPnR 405
 USB_FNR 403
 USB_ISTR 400

W

WWDG_CFR 147
 WWDG_CR 147

WWDG_SR 148

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST's terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST'S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER'S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2008 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com