

Introduction

This reference manual targets application developers. It provides complete information on how to use the STM32F72xxx and STM32F73xxx microcontroller memory and peripherals.

The STM32F72xxx and STM32F73xxx is a family of microcontrollers with different memory sizes, packages and peripherals.

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

For information on the Arm® Cortex®-M7 with FPU core, refer to the *Cortex®-M7 with FPU technical reference manual*.

Related documents

Available from STMicroelectronics web site www.st.com:

- STM32F72xxx and STM32F732xx/F733xx datasheets
- STM32F730x8 datasheet
- *STM32F7 Series Cortex®-M7 processor programming manual* (PM0253)

Contents

1	Documentation conventions	53
1.1	General information	53
1.2	List of abbreviations for registers	53
1.3	Glossary	54
1.4	Availability of peripherals	54
1.5	Memory organization	55
1.5.1	Introduction	55
1.5.2	Memory map and register boundary addresses	56
1.6	Embedded SRAM	59
1.7	Flash memory overview	60
1.8	Boot configuration	60
2	System and memory overview	62
2.1	System architecture	62
2.1.1	Multi AHB BusMatrix	63
2.1.2	AHB/APB bridges (APB)	63
2.1.3	CPU AXIM bus	64
2.1.4	ITCM bus	64
2.1.5	DTCM bus	64
2.1.6	CPU AHBS bus	64
2.1.7	AHB peripheral bus	64
2.1.8	DMA memory bus	64
2.1.9	DMA peripheral bus	65
2.1.10	USB OTG HS DMA bus	65
3	Embedded Flash memory (FLASH)	66
3.1	Introduction	66
3.2	Flash main features	66
3.3	Flash functional description	67
3.3.1	Flash memory organization	67
3.3.2	Read access latency	68
3.3.3	Flash program and erase operations	71
3.3.4	Unlocking the Flash control register	71

3.3.5	Program/erase parallelism	71
3.3.6	Flash erase sequences	72
3.3.7	Flash programming sequences	72
3.3.8	Flash Interrupts	74
3.4	FLASH Option bytes	74
3.4.1	Option bytes description	74
3.4.2	Option bytes programming	79
3.5	FLASH memory protection	79
3.5.1	Read protection (RDP)	79
3.5.2	Write protections	81
3.5.3	Proprietary code readout protection (PCROP)	82
3.6	One-time programmable bytes	83
3.7	FLASH registers	84
3.7.1	Flash access control register (FLASH_ACR)	84
3.7.2	Flash key register (FLASH_KEYR)	85
3.7.3	Flash option key register (FLASH_OPTKEYR)	85
3.7.4	Flash status register (FLASH_SR)	86
3.7.5	Flash control register (FLASH_CR)	87
3.7.6	Flash option control register (FLASH_OPTCR)	88
3.7.7	Flash option control register (FLASH_OPTCR1)	91
3.7.8	Flash option control register (FLASH_OPTCR2)	92
3.7.9	Flash interface register map	93
4	Power controller (PWR)	94
4.1	Power supplies	94
4.1.1	Independent A/D converter supply and reference voltage	96
4.1.2	Independent USB transceivers supply	96
4.1.3	Battery backup domain	98
4.1.4	Voltage regulator	100
4.2	Power supply supervisor	103
4.2.1	Power-on reset (POR)/power-down reset (PDR)	103
4.2.2	Brownout reset (BOR)	104
4.2.3	Programmable voltage detector (PWD)	104
4.3	Low-power modes	105
4.3.1	Debug mode	108
4.3.2	Run mode	109

4.3.3	Low-power mode	109
4.3.4	Sleep mode	110
4.3.5	Stop mode	111
4.3.6	Standby mode	114
4.3.7	Programming the RTC alternate functions to wake up the device from the Stop and Standby modes	116
4.4	Power control registers	119
4.4.1	PWR power control register (PWR_CR1)	119
4.4.2	PWR power control/status register (PWR_CSR1)	121
4.4.3	PWR power control/status register 2 (PWR_CR2)	123
4.4.4	PWR power control register 2 (PWR_CSR2)	125
4.5	PWR register map	127
5	Reset and clock control (RCC)	128
5.1	Reset	128
5.1.1	System reset	128
5.1.2	Power reset	128
5.1.3	Backup domain reset	129
5.2	Clocks	129
5.2.1	HSE clock	132
5.2.2	HSI clock	133
5.2.3	PLL	134
5.2.4	LSE clock	135
5.2.5	LSI clock	135
5.2.6	System clock (SYSCLK) selection	135
5.2.7	Clock security system (CSS)	135
5.2.8	RTC/AWU clock	136
5.2.9	Watchdog clock	137
5.2.10	Clock-out capability	137
5.2.11	Internal/external clock measurement using TIM5/TIM11	137
5.2.12	Peripheral clock enable register (RCC_AHBxENR, RCC_APBxENRy)	139
5.3	RCC registers	139
5.3.1	RCC clock control register (RCC_CR)	139
5.3.2	RCC PLL configuration register (RCC_PLLCFGR)	142
5.3.3	RCC clock configuration register (RCC_CFGR)	144
5.3.4	RCC clock interrupt register (RCC_CIR)	146

5.3.5	RCC AHB1 peripheral reset register (RCC_AHB1RSTR)	149
5.3.6	RCC AHB2 peripheral reset register (RCC_AHB2RSTR)	152
5.3.7	RCC AHB3 peripheral reset register (RCC_AHB3RSTR)	153
5.3.8	RCC APB1 peripheral reset register (RCC_APB1RSTR)	153
5.3.9	RCC APB2 peripheral reset register (RCC_APB2RSTR)	157
5.3.10	RCC AHB1 peripheral clock register (RCC_AHB1ENR)	159
5.3.11	RCC AHB2 peripheral clock enable register (RCC_AHB2ENR)	160
5.3.12	RCC AHB3 peripheral clock enable register (RCC_AHB3ENR)	161
5.3.13	RCC APB1 peripheral clock enable register (RCC_APB1ENR)	162
5.3.14	RCC APB2 peripheral clock enable register (RCC_APB2ENR)	165
5.3.15	RCC AHB1 peripheral clock enable in low-power mode register (RCC_AHB1LPENR)	168
5.3.16	RCC AHB2 peripheral clock enable in low-power mode register (RCC_AHB2LPENR)	170
5.3.17	RCC AHB3 peripheral clock enable in low-power mode register (RCC_AHB3LPENR)	171
5.3.18	RCC APB1 peripheral clock enable in low-power mode register (RCC_APB1LPENR)	171
5.3.19	RCC APB2 peripheral clock enabled in low-power mode register (RCC_APB2LPENR)	175
5.3.20	RCC backup domain control register (RCC_BDCR)	177
5.3.21	RCC clock control & status register (RCC_CSR)	178
5.3.22	RCC spread spectrum clock generation register (RCC_SSCGR)	180
5.3.23	RCC PLLI2S configuration register (RCC_PLLI2SCFGR)	181
5.3.24	RCC PLLSAI configuration register (RCC_PLLSAICFGR)	184
5.3.25	RCC dedicated clocks configuration register (RCC_DKCFGR1)	185
5.3.26	RCC dedicated clocks configuration register (DCKCFGR2)	187
5.3.27	RCC register map	190
6	General-purpose I/Os (GPIO)	193
6.1	Introduction	193
6.2	GPIO main features	193
6.3	GPIO functional description	193
6.3.1	General-purpose I/O (GPIO)	196
6.3.2	I/O pin alternate function multiplexer and mapping	196
6.3.3	I/O port control registers	197
6.3.4	I/O port data registers	197
6.3.5	I/O data bitwise handling	197

6.3.6	GPIO locking mechanism	198
6.3.7	I/O alternate function input/output	198
6.3.8	External interrupt/wakeup lines	198
6.3.9	Input configuration	199
6.3.10	Output configuration	199
6.3.11	Alternate function configuration	200
6.3.12	Analog configuration	201
6.3.13	Using the HSE or LSE oscillator pins as GPIOs	201
6.3.14	Using the GPIO pins in the backup supply domain	201
6.4	GPIO registers	202
6.4.1	GPIO port mode register (GPIOx_MODER) (x = A to K)	202
6.4.2	GPIO port output type register (GPIOx_OTYPER) (x = A to K)	202
6.4.3	GPIO port output speed register (GPIOx_OSPEEDR) (x = A to K)	203
6.4.4	GPIO port pull-up/pull-down register (GPIOx_PUPDR) (x = A to K)	203
6.4.5	GPIO port input data register (GPIOx_IDR) (x = A to K)	204
6.4.6	GPIO port output data register (GPIOx_ODR) (x = A to K)	204
6.4.7	GPIO port bit set/reset register (GPIOx_BSRR) (x = A to K)	204
6.4.8	GPIO port configuration lock register (GPIOx_LCKR) (x = A to K)	205
6.4.9	GPIO alternate function low register (GPIOx_AFRL) (x = A to K)	206
6.4.10	GPIO alternate function high register (GPIOx_AFRH) (x = A to J)	207
6.4.11	GPIO register map	208
7	System configuration controller (SYSCFG)	210
7.1	I/O compensation cell	210
7.2	SYSCFG registers	210
7.2.1	SYSCFG memory remap register (SYSCFG_MEMRMP)	210
7.2.2	SYSCFG peripheral mode configuration register (SYSCFG_PMC)	211
7.2.3	SYSCFG external interrupt configuration register 1 (SYSCFG_EXTICR1)	213
7.2.4	SYSCFG external interrupt configuration register 2 (SYSCFG_EXTICR2)	213
7.2.5	SYSCFG external interrupt configuration register 3 (SYSCFG_EXTICR3)	214

7.2.6	SYSCFG external interrupt configuration register 4 (SYSCFG_EXTICR4)	215
7.2.7	Compensation cell control register (SYSCFG_CMPCR)	215
7.2.8	SYSCFG register maps	216
8	Direct memory access controller (DMA)	217
8.1	DMA introduction	217
8.2	DMA main features	217
8.3	DMA functional description	219
8.3.1	DMA block diagram	219
8.3.2	DMA overview	219
8.3.3	DMA transactions	220
8.3.4	Channel selection	220
8.3.5	Arbiter	222
8.3.6	DMA streams	222
8.3.7	Source, destination and transfer modes	222
8.3.8	Pointer incrementation	225
8.3.9	Circular mode	226
8.3.10	Double-buffer mode	226
8.3.11	Programmable data width, packing/unpacking, endianness	227
8.3.12	Single and burst transfers	229
8.3.13	FIFO	229
8.3.14	DMA transfer completion	232
8.3.15	DMA transfer suspension	233
8.3.16	Flow controller	233
8.3.17	Summary of the possible DMA configurations	234
8.3.18	Stream configuration procedure	235
8.3.19	Error management	236
8.4	DMA interrupts	237
8.5	DMA registers	238
8.5.1	DMA low interrupt status register (DMA_LISR)	238
8.5.2	DMA high interrupt status register (DMA_HISR)	239
8.5.3	DMA low interrupt flag clear register (DMA_LIFCR)	240
8.5.4	DMA high interrupt flag clear register (DMA_HIFCR)	240
8.5.5	DMA stream x configuration register (DMA_SxCR)	241
8.5.6	DMA stream x number of data register (DMA_SxNDTR)	244
8.5.7	DMA stream x peripheral address register (DMA_SxPAR)	245

8.5.8	DMA stream x memory 0 address register (DMA_SxM0AR)	245
8.5.9	DMA stream x memory 1 address register (DMA_SxM1AR)	245
8.5.10	DMA stream x FIFO control register (DMA_SxFCR)	246
8.5.11	DMA register map	248
9	Nested vectored interrupt controller (NVIC)	252
9.1	NVIC features	252
9.1.1	SysTick calibration value register	252
9.1.2	Interrupt and exception vectors	252
10	Extended interrupts and events controller (EXTI)	257
10.1	EXTI main features	257
10.2	EXTI block diagram	257
10.3	Wakeup event management	258
10.4	Functional description	258
10.5	Hardware interrupt selection	258
10.6	Hardware event selection	258
10.7	Software interrupt/event selection	259
10.8	External interrupt/event line mapping	259
10.9	EXTI registers	260
10.9.1	Interrupt mask register (EXTI_IMR)	260
10.9.2	Event mask register (EXTI_EMR)	260
10.9.3	Rising trigger selection register (EXTI_RTSR)	261
10.9.4	Falling trigger selection register (EXTI_FTSR)	261
10.9.5	Software interrupt event register (EXTI_SWIER)	262
10.9.6	Pending register (EXTI_PR)	262
10.9.7	EXTI register map	263
11	Cyclic redundancy check calculation unit (CRC)	264
11.1	Introduction	264
11.2	CRC main features	264
11.3	CRC functional description	265
11.3.1	CRC block diagram	265
11.3.2	CRC internal signals	265
11.3.3	CRC operation	265
11.4	CRC registers	267

11.4.1	Data register (CRC_DR)	267
11.4.2	Independent data register (CRC_IDR)	267
11.4.3	Control register (CRC_CR)	268
11.4.4	Initial CRC value (CRC_INIT)	268
11.4.5	CRC polynomial (CRC_POL)	269
11.4.6	CRC register map	269
12	Flexible memory controller (FMC)	270
12.1	FMC main features	270
12.2	FMC block diagram	271
12.3	AHB interface	272
12.3.1	Supported memories and transactions	272
12.4	External device address mapping	274
12.4.1	NOR/PSRAM address mapping	274
12.4.2	NAND Flash memory address mapping	275
12.4.3	SDRAM address mapping	276
12.5	NOR Flash/PSRAM controller	279
12.5.1	External memory interface signals	281
12.5.2	Supported memories and transactions	282
12.5.3	General timing rules	284
12.5.4	NOR Flash/PSRAM controller asynchronous transactions	284
12.5.5	Synchronous transactions	301
12.5.6	NOR/PSRAM controller registers	308
12.6	NAND Flash controller	315
12.6.1	External memory interface signals	315
12.6.2	NAND Flash supported memories and transactions	317
12.6.3	Timing diagrams for NAND Flash memory	317
12.6.4	NAND Flash operations	318
12.6.5	NAND Flash prewait functionality	319
12.6.6	Computation of the error correction code (ECC) in NAND Flash memory	320
12.6.7	NAND Flash controller registers	321
12.7	SDRAM controller	327
12.7.1	SDRAM controller main features	327
12.7.2	SDRAM External memory interface signals	327
12.7.3	SDRAM controller functional description	328
12.7.4	Low-power modes	334

12.7.5	SDRAM controller registers	337
12.8	FMC register map	345
13	Quad-SPI interface (QUADSPI)	347
13.1	Introduction	347
13.2	QUADSPI main features	347
13.3	QUADSPI functional description	347
13.3.1	QUADSPI block diagram	347
13.3.2	QUADSPI pins	348
13.3.3	QUADSPI command sequence	349
13.3.4	QUADSPI signal interface protocol modes	351
13.3.5	QUADSPI indirect mode	353
13.3.6	QUADSPI status flag polling mode	355
13.3.7	QUADSPI memory-mapped mode	355
13.3.8	QUADSPI Flash memory configuration	356
13.3.9	QUADSPI delayed data sampling	356
13.3.10	QUADSPI configuration	356
13.3.11	QUADSPI usage	357
13.3.12	Sending the instruction only once	359
13.3.13	QUADSPI error management	359
13.3.14	QUADSPI busy bit and abort functionality	360
13.3.15	nCS behavior	360
13.4	QUADSPI interrupts	362
13.5	QUADSPI registers	363
13.5.1	QUADSPI control register (QUADSPI_CR)	363
13.5.2	QUADSPI device configuration register (QUADSPI_DCR)	366
13.5.3	QUADSPI status register (QUADSPI_SR)	367
13.5.4	QUADSPI flag clear register (QUADSPI_FCR)	368
13.5.5	QUADSPI data length register (QUADSPI_DLR)	368
13.5.6	QUADSPI communication configuration register (QUADSPI_CCR)	369
13.5.7	QUADSPI address register (QUADSPI_AR)	371
13.5.8	QUADSPI alternate bytes registers (QUADSPI_ABR)	372
13.5.9	QUADSPI data register (QUADSPI_DR)	372
13.5.10	QUADSPI polling status mask register (QUADSPI_PSMKR)	373
13.5.11	QUADSPI polling status match register (QUADSPI_PSMAR)	373
13.5.12	QUADSPI polling interval register (QUADSPI_PIR)	374

13.5.13	QUADSPI low-power timeout register (QUADSPI_LPTR)	374
13.5.14	QUADSPI register map	375
14	Analog-to-digital converter (ADC)	376
14.1	ADC introduction	376
14.2	ADC main features	376
14.3	ADC functional description	376
14.3.1	ADC on-off control	378
14.3.2	ADC1/2 and ADC3 connectivity	379
14.3.3	ADC clock	382
14.3.4	Channel selection	382
14.3.5	Single conversion mode	383
14.3.6	Continuous conversion mode	383
14.3.7	Timing diagram	383
14.3.8	Analog watchdog	384
14.3.9	Scan mode	385
14.3.10	Injected channel management	385
14.3.11	Discontinuous mode	386
14.4	Data alignment	387
14.5	Channel-wise programmable sampling time	388
14.6	Conversion on external trigger and trigger polarity	389
14.7	Fast conversion mode	390
14.8	Data management	391
14.8.1	Using the DMA	391
14.8.2	Managing a sequence of conversions without using the DMA	391
14.8.3	Conversions without DMA and without overrun detection	392
14.9	Multi ADC mode	392
14.9.1	Injected simultaneous mode	395
14.9.2	Regular simultaneous mode	396
14.9.3	Interleaved mode	397
14.9.4	Alternate trigger mode	399
14.9.5	Combined regular/injected simultaneous mode	401
14.9.6	Combined regular simultaneous + alternate trigger mode	401
14.10	Temperature sensor	402
14.11	Battery charge monitoring	404
14.12	ADC interrupts	404

14.13	ADC registers	405
14.13.1	ADC status register (ADC_SR)	405
14.13.2	ADC control register 1 (ADC_CR1)	406
14.13.3	ADC control register 2 (ADC_CR2)	408
14.13.4	ADC sample time register 1 (ADC_SMPR1)	410
14.13.5	ADC sample time register 2 (ADC_SMPR2)	411
14.13.6	ADC injected channel data offset register x (ADC_JOFRx) (x=1..4)	411
14.13.7	ADC watchdog higher threshold register (ADC_HTR)	411
14.13.8	ADC watchdog lower threshold register (ADC_LTR)	412
14.13.9	ADC regular sequence register 1 (ADC_SQR1)	412
14.13.10	ADC regular sequence register 2 (ADC_SQR2)	413
14.13.11	ADC regular sequence register 3 (ADC_SQR3)	414
14.13.12	ADC injected sequence register (ADC_JSQR)	415
14.13.13	ADC injected data register x (ADC_JDRx) (x= 1..4)	415
14.13.14	ADC regular data register (ADC_DR)	416
14.13.15	ADC Common status register (ADC_CSR)	416
14.13.16	ADC common control register (ADC_CCR)	417
14.13.17	ADC common regular data register for dual and triple modes (ADC_CDR)	420
14.13.18	ADC register map	420
15	Digital-to-analog converter (DAC)	423
15.1	DAC introduction	423
15.2	DAC main features	423
15.3	DAC functional description	424
15.3.1	DAC channel enable	424
15.3.2	DAC output buffer enable	425
15.3.3	DAC data format	425
15.3.4	DAC conversion	426
15.3.5	DAC output voltage	427
15.3.6	DAC trigger selection	427
15.3.7	DMA request	428
15.3.8	Noise generation	428
15.3.9	Triangle-wave generation	429
15.4	Dual DAC channel conversion	430
15.4.1	Independent trigger without wave generation	431
15.4.2	Independent trigger with single LFSR generation	431

15.4.3	Independent trigger with different LFSR generation	431
15.4.4	Independent trigger with single triangle generation	432
15.4.5	Independent trigger with different triangle generation	432
15.4.6	Simultaneous software start	432
15.4.7	Simultaneous trigger without wave generation	433
15.4.8	Simultaneous trigger with single LFSR generation	433
15.4.9	Simultaneous trigger with different LFSR generation	433
15.4.10	Simultaneous trigger with single triangle generation	434
15.4.11	Simultaneous trigger with different triangle generation	434
15.5	DAC registers	435
15.5.1	DAC control register (DAC_CR)	435
15.5.2	DAC software trigger register (DAC_SWTRIGR)	438
15.5.3	DAC channel1 12-bit right-aligned data holding register (DAC_DHR12R1)	438
15.5.4	DAC channel1 12-bit left aligned data holding register (DAC_DHR12L1)	439
15.5.5	DAC channel1 8-bit right aligned data holding register (DAC_DHR8R1)	439
15.5.6	DAC channel2 12-bit right aligned data holding register (DAC_DHR12R2)	440
15.5.7	DAC channel2 12-bit left aligned data holding register (DAC_DHR12L2)	440
15.5.8	DAC channel2 8-bit right-aligned data holding register (DAC_DHR8R2)	440
15.5.9	Dual DAC 12-bit right-aligned data holding register (DAC_DHR12RD)	441
15.5.10	DUAL DAC 12-bit left aligned data holding register (DAC_DHR12LD)	441
15.5.11	DUAL DAC 8-bit right aligned data holding register (DAC_DHR8RD)	442
15.5.12	DAC channel1 data output register (DAC_DOR1)	442
15.5.13	DAC channel2 data output register (DAC_DOR2)	442
15.5.14	DAC status register (DAC_SR)	443
15.5.15	DAC register map	444
16	True random number generator (RNG)	445
16.1	Introduction	445
16.2	RNG main features	445
16.3	RNG functional description	446

16.3.1	RNG block diagram	446
16.3.2	RNG internal signals	446
16.3.3	Random number generation	447
16.3.4	RNG initialization	449
16.3.5	RNG operation	449
16.3.6	RNG clocking	450
16.3.7	Error management	450
16.4	RNG low-power usage	451
16.5	RNG interrupts	451
16.6	RNG processing time	451
16.7	Entropy source validation	452
16.7.1	Introduction	452
16.7.2	Validation conditions	452
16.7.3	Data collection	452
16.8	RNG registers	453
16.8.1	RNG control register (RNG_CR)	453
16.8.2	RNG status register (RNG_SR)	454
16.8.3	RNG data register (RNG_DR)	455
16.8.4	RNG register map	456
17	AES hardware accelerator (AES)	457
17.1	Introduction	457
17.2	AES main features	457
17.3	AES implementation	458
17.4	AES functional description	458
17.4.1	AES block diagram	458
17.4.2	AES internal signals	458
17.4.3	AES cryptographic core	459
17.4.4	AES procedure to perform a cipher operation	464
17.4.5	AES decryption key preparation	468
17.4.6	AES ciphertext stealing and data padding	469
17.4.7	AES task suspend and resume	470
17.4.8	AES basic chaining modes (ECB, CBC)	471
17.4.9	AES counter (CTR) mode	476
17.4.10	AES Galois/counter mode (GCM)	478
17.4.11	AES Galois message authentication code (GMAC)	483

17.4.12	AES counter with CBC-MAC (CCM)	485
17.4.13	.AES data registers and data swapping	490
17.4.14	AES key registers	492
17.4.15	AES initialization vector registers	492
17.4.16	AES DMA interface	492
17.4.17	AES error management	495
17.5	AES interrupts	495
17.6	AES processing latency	496
17.7	AES registers	497
17.7.1	AES control register (AES_CR)	497
17.7.2	AES status register (AES_SR)	500
17.7.3	AES data input register (AES_DINR)	501
17.7.4	AES data output register (AES_DOUTR)	502
17.7.5	AES key register 0 (AES_KEYR0)	502
17.7.6	AES key register 1 (AES_KEYR1)	503
17.7.7	AES key register 2 (AES_KEYR2)	503
17.7.8	AES key register 3 (AES_KEYR3)	504
17.7.9	AES initialization vector register 0 (AES_IVR0)	504
17.7.10	AES initialization vector register 1 (AES_IVR1)	504
17.7.11	AES initialization vector register 2 (AES_IVR2)	505
17.7.12	AES initialization vector register 3 (AES_IVR3)	505
17.7.13	AES key register 4 (AES_KEYR4)	506
17.7.14	AES key register 5 (AES_KEYR5)	506
17.7.15	AES key register 6 (AES_KEYR6)	506
17.7.16	AES key register 7 (AES_KEYR7)	507
17.7.17	AES suspend registers (AES_SUSPxR)	507
17.7.18	AES register map	508
18	Advanced-control timers (TIM1/TIM8)	510
18.1	TIM1/TIM8 introduction	510
18.2	TIM1/TIM8 main features	510
18.3	TIM1/TIM8 functional description	512
18.3.1	Time-base unit	512
18.3.2	Counter modes	514
18.3.3	Repetition counter	525
18.3.4	External trigger input	527

18.3.5	Clock selection	528
18.3.6	Capture/compare channels	532
18.3.7	Input capture mode	534
18.3.8	PWM input mode	535
18.3.9	Forced output mode	536
18.3.10	Output compare mode	537
18.3.11	PWM mode	538
18.3.12	Asymmetric PWM mode	541
18.3.13	Combined PWM mode	542
18.3.14	Combined 3-phase PWM mode	543
18.3.15	Complementary outputs and dead-time insertion	544
18.3.16	Using the break function	546
18.3.17	Clearing the OC _x REF signal on an external event	552
18.3.18	6-step PWM generation	554
18.3.19	One-pulse mode	555
18.3.20	Retriggerable one pulse mode (OPM)	556
18.3.21	Encoder interface mode	557
18.3.22	UIF bit remapping	559
18.3.23	Timer input XOR function	560
18.3.24	Interfacing with Hall sensors	560
18.3.25	Timer synchronization	563
18.3.26	ADC synchronization	567
18.3.27	DMA burst mode	567
18.3.28	Debug mode	568
18.4	TIM1/TIM8 registers	569
18.4.1	TIMx control register 1 (TIM _x _CR1)(x = 1, 8)	569
18.4.2	TIMx control register 2 (TIM _x _CR2)(x = 1, 8)	570
18.4.3	TIMx slave mode control register (TIM _x _SMCR)(x = 1, 8)	573
18.4.4	TIMx DMA/interrupt enable register (TIM _x _DIER)(x = 1, 8)	575
18.4.5	TIMx status register (TIM _x _SR)(x = 1, 8)	577
18.4.6	TIMx event generation register (TIM _x _EGR)(x = 1, 8)	579
18.4.7	TIMx capture/compare mode register 1 (TIM _x _CCMR1)(x = 1, 8)	580
18.4.8	TIMx capture/compare mode register 2 (TIM _x _CCMR2)(x = 1, 8)	584
18.4.9	TIMx capture/compare enable register (TIM _x _CCER)(x = 1, 8)	586
18.4.10	TIMx counter (TIM _x _CNT)(x = 1, 8)	590
18.4.11	TIMx prescaler (TIM _x _PSC)(x = 1, 8)	590
18.4.12	TIMx auto-reload register (TIM _x _ARR)(x = 1, 8)	590

18.4.13	TIMx repetition counter register (TIMx_RCR)(x = 1, 8)	591
18.4.14	TIMx capture/compare register 1 (TIMx_CCR1)(x = 1, 8)	591
18.4.15	TIMx capture/compare register 2 (TIMx_CCR2)(x = 1, 8)	592
18.4.16	TIMx capture/compare register 3 (TIMx_CCR3)(x = 1, 8)	592
18.4.17	TIMx capture/compare register 4 (TIMx_CCR4)(x = 1, 8)	593
18.4.18	TIMx break and dead-time register (TIMx_BDTR)(x = 1, 8)	593
18.4.19	TIMx DMA control register (TIMx_DCR)(x = 1, 8)	596
18.4.20	TIMx DMA address for full transfer (TIMx_DMAR)(x = 1, 8)	597
18.4.21	TIMx capture/compare mode register 3 (TIMx_CCMR3)(x = 1, 8)	598
18.4.22	TIMx capture/compare register 5 (TIMx_CCR5)(x = 1, 8)	599
18.4.23	TIMx capture/compare register 6 (TIMx_CCR6)(x = 1, 8)	600
18.4.24	TIM1 register map	601
18.4.25	TIM8 register map	603
19	General-purpose timers (TIM2/TIM3/TIM4/TIM5)	605
19.1	TIM2/TIM3/TIM4/TIM5 introduction	605
19.2	TIM2/TIM3/TIM4/TIM5 main features	605
19.3	TIM2/TIM3/TIM4/TIM5 functional description	607
19.3.1	Time-base unit	607
19.3.2	Counter modes	609
19.3.3	Clock selection	619
19.3.4	Capture/Compare channels	623
19.3.5	Input capture mode	625
19.3.6	PWM input mode	626
19.3.7	Forced output mode	627
19.3.8	Output compare mode	628
19.3.9	PWM mode	629
19.3.10	Asymmetric PWM mode	632
19.3.11	Combined PWM mode	633
19.3.12	Clearing the OC _x REF signal on an external event	634
19.3.13	One-pulse mode	636
19.3.14	Retriggerable one pulse mode (OPM)	637
19.3.15	Encoder interface mode	638
19.3.16	UIF bit remapping	640
19.3.17	Timer input XOR function	640
19.3.18	Timers and external trigger synchronization	641
19.3.19	Timer synchronization	644

19.3.20	DMA burst mode	648
19.3.21	Debug mode	649
19.4	TIM2/TIM3/TIM4/TIM5 registers	650
19.4.1	TIMx control register 1 (TIMx_CR1)(x = 2 to 5)	650
19.4.2	TIMx control register 2 (TIMx_CR2)(x = 2 to 5)	651
19.4.3	TIMx slave mode control register (TIMx_SMCR)(x = 2 to 5)	653
19.4.4	TIMx DMA/Interrupt enable register (TIMx_DIER)(x = 2 to 5)	656
19.4.5	TIMx status register (TIMx_SR)(x = 2 to 5)	657
19.4.6	TIMx event generation register (TIMx_EGR)(x = 2 to 5)	658
19.4.7	TIMx capture/compare mode register 1 (TIMx_CCMR1)(x = 2 to 5)	659
19.4.8	TIMx capture/compare mode register 2 (TIMx_CCMR2)(x = 2 to 5)	663
19.4.9	TIMx capture/compare enable register (TIMx_CCER)(x = 2 to 5)	665
19.4.10	TIMx counter (TIMx_CNT)(x = 2 to 5)	666
19.4.11	TIMx prescaler (TIMx_PSC)(x = 2 to 5)	667
19.4.12	TIMx auto-reload register (TIMx_ARR)(x = 2 to 5)	667
19.4.13	TIMx capture/compare register 1 (TIMx_CCR1)(x = 2 to 5)	668
19.4.14	TIMx capture/compare register 2 (TIMx_CCR2)(x = 2 to 5)	668
19.4.15	TIMx capture/compare register 3 (TIMx_CCR3)(x = 2 to 5)	669
19.4.16	TIMx capture/compare register 4 (TIMx_CCR4)(x = 2 to 5)	669
19.4.17	TIMx DMA control register (TIMx_DCR)(x = 2 to 5)	670
19.4.18	TIMx DMA address for full transfer (TIMx_DMAR)(x = 2 to 5)	670
19.4.19	TIM2 option register (TIM2_OR)	671
19.4.20	TIM5 option register (TIM5_OR)	671
19.4.21	TIMx register map	672
20	General-purpose timers (TIM9/TIM10/TIM11/TIM12/TIM13/TIM14)	675
20.1	TIM9/TIM10/TIM11/TIM12/TIM13/TIM14 introduction	675
20.2	TIM9/TIM10/TIM11/TIM12/TIM13/TIM14 main features	675
20.2.1	TIM9/TIM12 main features	675
20.2.2	TIM10/TIM11/TIM13/TIM14 main features	676
20.3	TIM9/TIM10/TIM11/TIM12/TIM13/TIM14 functional description	678
20.3.1	Time-base unit	678
20.3.2	Counter modes	680
20.3.3	Clock selection	683
20.3.4	Capture/compare channels	685
20.3.5	Input capture mode	687
20.3.6	PWM input mode (only for TIM9/TIM12)	688

20.3.7	Forced output mode	689
20.3.8	Output compare mode	689
20.3.9	PWM mode	690
20.3.10	Combined PWM mode (TIM9/TIM12 only)	691
20.3.11	One-pulse mode	693
20.3.12	Retriggerable one pulse mode (OPM) (TIM12 only)	694
20.3.13	UIF bit remapping	695
20.3.14	TIM9/TIM12 external trigger synchronization	695
20.3.15	Slave mode – combined reset + trigger mode	698
20.3.16	Timer synchronization (TIM9/TIM12)	699
20.3.17	Debug mode	699
20.4	TIM9/TIM12 registers	699
20.4.1	TIMx control register 1 (TIMx_CR1)(x = 9, 12)	699
20.4.2	TIMx slave mode control register (TIMx_SMCR)(x = 9, 12)	700
20.4.3	TIMx Interrupt enable register (TIMx_DIER)(x = 9, 12)	702
20.4.4	TIMx status register (TIMx_SR)(x = 9, 12)	702
20.4.5	TIMx event generation register (TIMx_EGR)(x = 9, 12)	703
20.4.6	TIMx capture/compare mode register 1 (TIMx_CCMR1)(x = 9, 12)	704
20.4.7	TIMx capture/compare enable register (TIMx_CCER)(x = 9, 12)	708
20.4.8	TIMx counter (TIMx_CNT)(x = 9, 12)	709
20.4.9	TIMx prescaler (TIMx_PSC)(x = 9, 12)	709
20.4.10	TIMx auto-reload register (TIMx_ARR)(x = 9, 12)	710
20.4.11	TIMx capture/compare register 1 (TIMx_CCR1)(x = 9, 12)	710
20.4.12	TIMx capture/compare register 2 (TIMx_CCR2)(x = 9, 12)	710
20.4.13	TIM9/TIM12 register map	712
20.5	TIM10/TIM11/TIM13/TIM14 registers	714
20.5.1	TIMx control register 1 (TIMx_CR1)(x = 10, 11, 13, 14)	714
20.5.2	TIMx Interrupt enable register (TIMx_DIER)(x = 10, 11, 13, 14)	715
20.5.3	TIMx status register (TIMx_SR)(x = 10, 11, 13, 14)	715
20.5.4	TIMx event generation register (TIMx_EGR)(x = 10, 11, 13, 14)	716
20.5.5	TIMx capture/compare mode register 1 (TIMx_CCMR1)(x = 10, 11, 13, 14)	717
20.5.6	TIMx capture/compare enable register (TIMx_CCER)(x = 10, 11, 13, 14)	719
20.5.7	TIMx counter (TIMx_CNT)(x = 10, 11, 13, 14)	720
20.5.8	TIMx prescaler (TIMx_PSC)(x = 10, 11, 13, 14)	721
20.5.9	TIMx auto-reload register (TIMx_ARR)(x = 10, 11, 13, 14)	721

20.5.10	TIMx capture/compare register 1 (TIMx_CCR1)(x = 10, 11, 13, 14)	721
20.5.11	TIM11 option register 1 (TIM11_OR)	722
20.5.12	TIM10/TIM11/TIM13/TIM14 register map	722
21	Basic timers (TIM6/TIM7)	724
21.1	TIM6/TIM7 introduction	724
21.2	TIM6/TIM7 main features	724
21.3	TIM6/TIM7 functional description	725
21.3.1	Time-base unit	725
21.3.2	Counting mode	727
21.3.3	UIF bit remapping	730
21.3.4	Clock source	730
21.3.5	Debug mode	731
21.4	TIM6/TIM7 registers	731
21.4.1	TIM6/TIM7 control register 1 (TIMx_CR1)	731
21.4.2	TIM6/TIM7 control register 2 (TIMx_CR2)	733
21.4.3	TIM6/TIM7 DMA/Interrupt enable register (TIMx_DIER)	733
21.4.4	TIM6/TIM7 status register (TIMx_SR)	734
21.4.5	TIM6/TIM7 event generation register (TIMx_EGR)	734
21.4.6	TIM6/TIM7 counter (TIMx_CNT)	734
21.4.7	TIM6/TIM7 prescaler (TIMx_PSC)	735
21.4.8	TIM6/TIM7 auto-reload register (TIMx_ARR)	735
21.4.9	TIM6/TIM7 register map	736
22	Low-power timer (LPTIM)	737
22.1	Introduction	737
22.2	LPTIM main features	737
22.3	LPTIM implementation	737
22.4	LPTIM functional description	738
22.4.1	LPTIM block diagram	738
22.4.2	LPTIM trigger mapping	738
22.4.3	LPTIM reset and clocks	739
22.4.4	Glitch filter	739
22.4.5	Prescaler	740
22.4.6	Trigger multiplexer	740
22.4.7	Operating mode	741

22.4.8	Timeout function	743
22.4.9	Waveform generation	743
22.4.10	Register update	744
22.4.11	Counter mode	745
22.4.12	Timer enable	745
22.4.13	Encoder mode	746
22.4.14	Debug mode	747
22.5	LPTIM low-power modes	747
22.6	LPTIM interrupts	748
22.7	LPTIM registers	748
22.7.1	LPTIM interrupt and status register (LPTIM_ISR)	748
22.7.2	LPTIM interrupt clear register (LPTIM_ICR)	750
22.7.3	LPTIM interrupt enable register (LPTIM_IER)	751
22.7.4	LPTIM configuration register (LPTIM_CFGR)	752
22.7.5	LPTIM control register (LPTIM_CR)	755
22.7.6	LPTIM compare register (LPTIM_CMP)	756
22.7.7	LPTIM autoreload register (LPTIM_ARR)	757
22.7.8	LPTIM counter register (LPTIM_CNT)	757
22.7.9	LPTIM register map	758
23	Independent watchdog (IWDG)	759
23.1	Introduction	759
23.2	IWDG main features	759
23.3	IWDG functional description	759
23.3.1	IWDG block diagram	759
23.3.2	Window option	760
23.3.3	Hardware watchdog	761
23.3.4	Low-power freeze	761
23.3.5	Behavior in Stop and Standby modes	761
23.3.6	Register access protection	761
23.3.7	Debug mode	761
23.4	IWDG registers	762
23.4.1	Key register (IWDG_KR)	762
23.4.2	Prescaler register (IWDG_PR)	763
23.4.3	Reload register (IWDG_RLR)	764
23.4.4	Status register (IWDG_SR)	765

23.4.5	Window register (IWDG_WINR)	766
23.4.6	IWDG register map	767
24	System window watchdog (WWDG)	768
24.1	Introduction	768
24.2	WWDG main features	768
24.3	WWDG functional description	768
24.3.1	WWDG block diagram	769
24.3.2	Enabling the watchdog	769
24.3.3	Controlling the downcounter	769
24.3.4	Advanced watchdog interrupt feature	769
24.3.5	How to program the watchdog timeout	770
24.3.6	Debug mode	771
24.4	WWDG registers	772
24.4.1	Control register (WWDG_CR)	772
24.4.2	Configuration register (WWDG_CFR)	772
24.4.3	Status register (WWDG_SR)	773
24.4.4	WWDG register map	774
25	Real-time clock (RTC)	775
25.1	Introduction	775
25.2	RTC main features	776
25.3	RTC functional description	777
25.3.1	RTC block diagram	777
25.3.2	GPIOs controlled by the RTC	778
25.3.3	Clock and prescalers	780
25.3.4	Real-time clock and calendar	781
25.3.5	Programmable alarms	781
25.3.6	Periodic auto-wakeup	782
25.3.7	RTC initialization and configuration	783
25.3.8	Reading the calendar	784
25.3.9	Resetting the RTC	785
25.3.10	RTC synchronization	786
25.3.11	RTC reference clock detection	786
25.3.12	RTC smooth digital calibration	787
25.3.13	Time-stamp function	789

25.3.14	Tamper detection	790
25.3.15	Calibration clock output	792
25.3.16	Alarm output	792
25.4	RTC low-power modes	793
25.5	RTC interrupts	793
25.6	RTC registers	794
25.6.1	RTC time register (RTC_TR)	794
25.6.2	RTC date register (RTC_DR)	795
25.6.3	RTC control register (RTC_CR)	796
25.6.4	RTC initialization and status register (RTC_ISR)	799
25.6.5	RTC prescaler register (RTC_PRER)	802
25.6.6	RTC wakeup timer register (RTC_WUTR)	803
25.6.7	RTC alarm A register (RTC_ALRMAR)	804
25.6.8	RTC alarm B register (RTC_ALRMBR)	805
25.6.9	RTC write protection register (RTC_WPR)	806
25.6.10	RTC sub second register (RTC_SSR)	806
25.6.11	RTC shift control register (RTC_SHIFTR)	807
25.6.12	RTC timestamp time register (RTC_TSTR)	808
25.6.13	RTC timestamp date register (RTC_TSDDR)	809
25.6.14	RTC time-stamp sub second register (RTC_TSSSR)	810
25.6.15	RTC calibration register (RTC_CALR)	811
25.6.16	RTC tamper configuration register (RTC_TAMPCCR)	812
25.6.17	RTC alarm A sub second register (RTC_ALRMASSR)	815
25.6.18	RTC alarm B sub second register (RTC_ALRMBSSR)	816
25.6.19	RTC option register (RTC_OR)	817
25.6.20	RTC backup registers (RTC_BKPxR)	817
25.6.21	RTC register map	818
26	Inter-integrated circuit (I2C) interface	820
26.1	Introduction	820
26.2	I2C main features	820
26.3	I2C implementation	821
26.4	I2C functional description	821
26.4.1	I2C block diagram	822
26.4.2	I2C clock requirements	822
26.4.3	Mode selection	823

26.4.4	I2C initialization	824
26.4.5	Software reset	828
26.4.6	Data transfer	829
26.4.7	I2C slave mode	831
26.4.8	I2C master mode	840
26.4.9	I2C_TIMINGR register configuration examples	852
26.4.10	SMBus specific features	853
26.4.11	SMBus initialization	856
26.4.12	SMBus: I2C_TIMEOUTR register configuration examples	858
26.4.13	SMBus slave mode	858
26.4.14	Error conditions	865
26.4.15	DMA requests	867
26.4.16	Debug mode	868
26.5	I2C low-power modes	868
26.6	I2C interrupts	869
26.7	I2C registers	869
26.7.1	I2C2 control register 1 (I2C_CR1)	869
26.7.2	I2C2 control register 2 (I2C_CR2)	872
26.7.3	I2C2 own address 1 register (I2C_OAR1)	875
26.7.4	I2C2 own address 2 register (I2C_OAR2)	876
26.7.5	I2C2 timing register (I2C_TIMINGR)	877
26.7.6	I2C2 timeout register (I2C_TIMEOUTR)	878
26.7.7	I2C2 interrupt and status register (I2C_ISR)	879
26.7.8	I2C2 interrupt clear register (I2C_ICR)(.	881
26.7.9	I2C2 PEC register (I2C_PECR)	882
26.7.10	I2C2 receive data register (I2C_RXDR)	883
26.7.11	I2C2 transmit data register (I2C_TXDR)	883
26.7.12	I2C register map	884
27	Universal synchronous receiver transmitter (USART) /universal asynchronous receiver transmitter (UART)	886
27.1	Introduction	886
27.2	USART main features	886
27.3	USART extended features	887
27.4	USART implementation	888
27.5	USART functional description	888

27.5.1	USART character description	891
27.5.2	USART transmitter	893
27.5.3	USART receiver	895
27.5.4	USART baud rate generation	901
27.5.5	Tolerance of the USART receiver to clock deviation	904
27.5.6	USART auto baud rate detection	905
27.5.7	Multiprocessor communication using USART	906
27.5.8	Modbus communication using USART	908
27.5.9	USART parity control	909
27.5.10	USART LIN (local interconnection network) mode	910
27.5.11	USART synchronous mode	912
27.5.12	USART Single-wire Half-duplex communication	915
27.5.13	USART Smartcard mode	915
27.5.14	USART IrDA SIR ENDEC block	920
27.5.15	USART continuous communication in DMA mode	922
27.5.16	RS232 hardware flow control and RS485 driver enable using USART	924
27.6	USART low-power modes	926
27.7	USART interrupts	927
27.8	USART registers	929
27.8.1	Control register 1 (USART_CR1)	929
27.8.2	Control register 2 (USART_CR2)	932
27.8.3	Control register 3 (USART_CR3)	935
27.8.4	Baud rate register (USART_BRR)	938
27.8.5	Guard time and prescaler register (USART_GTPR)	939
27.8.6	Receiver timeout register (USART_RTOR)	940
27.8.7	Request register (USART_RQR)	941
27.8.8	Interrupt and status register (USART_ISR)	942
27.8.9	Interrupt flag clear register (USART_ICR)	947
27.8.10	Receive data register (USART_RDR)	948
27.8.11	Transmit data register (USART_TDR)	948
27.8.12	USART register map	949
28	Serial peripheral interface / inter-IC sound (SPI/I2S)	951
28.1	Introduction	951
28.2	SPI main features	951
28.3	I2S main features	952

28.4	SPI/I ² S implementation	952
28.5	SPI functional description	953
28.5.1	General description	953
28.5.2	Communications between one master and one slave	954
28.5.3	Standard multi-slave communication	956
28.5.4	Multi-master communication	957
28.5.5	Slave select (NSS) pin management	958
28.5.6	Communication formats	959
28.5.7	Configuration of SPI	961
28.5.8	Procedure for enabling SPI	962
28.5.9	Data transmission and reception procedures	962
28.5.10	SPI status flags	972
28.5.11	SPI error flags	973
28.5.12	NSS pulse mode	974
28.5.13	TI mode	974
28.5.14	CRC calculation	975
28.6	SPI interrupts	977
28.7	I ² S functional description	978
28.7.1	I ² S general description	978
28.7.2	I ² S full duplex	979
28.7.3	Supported audio protocols	980
28.7.4	Start-up description	987
28.7.5	Clock generator	989
28.7.6	I ² S master mode	992
28.7.7	I ² S slave mode	993
28.7.8	I ² S status flags	995
28.7.9	I ² S error flags	996
28.7.10	DMA features	997
28.8	I ² S interrupts	997
28.9	SPI and I ² S registers	998
28.9.1	SPI control register 1 (SPIx_CR1)	998
28.9.2	SPI control register 2 (SPIx_CR2)	1000
28.9.3	SPI status register (SPIx_SR)	1002
28.9.4	SPI data register (SPIx_DR)	1004
28.9.5	SPI CRC polynomial register (SPIx_CRCPR)	1004
28.9.6	SPI Rx CRC register (SPIx_RXCRCR)	1004

28.9.7	SPI Tx CRC register (SPIx_TXCRCR)	1005
28.9.8	SPIx_I2S configuration register (SPIx_I2SCFGR)	1005
28.9.9	SPIx_I2S prescaler register (SPIx_I2SPR)	1007
28.9.10	SPI/I2S register map	1009
29	Serial audio interface (SAI)	1010
29.1	Introduction	1010
29.2	SAI main features	1011
29.3	SAI functional description	1012
29.3.1	SAI block diagram	1012
29.3.2	SAI pins and internal signals	1013
29.3.3	Main SAI modes	1013
29.3.4	SAI synchronization mode	1014
29.3.5	Audio data size	1015
29.3.6	Frame synchronization	1016
29.3.7	Slot configuration	1019
29.3.8	SAI clock generator	1021
29.3.9	Internal FIFOs	1023
29.3.10	AC'97 link controller	1025
29.3.11	SPDIF output	1027
29.3.12	Specific features	1030
29.3.13	Error flags	1034
29.3.14	Disabling the SAI	1037
29.3.15	SAI DMA interface	1037
29.4	SAI interrupts	1038
29.5	SAI registers	1039
29.5.1	Global configuration register (SAI_GCR)	1039
29.5.2	Configuration register 1 (SAI_ACR1)	1039
29.5.3	Configuration register 1 (SAI_BCR1)	1042
29.5.4	Configuration register 2 (SAI_ACR2)	1044
29.5.5	Configuration register 2 (SAI_BCR2)	1046
29.5.6	Frame configuration register (SAI_AFRCR)	1048
29.5.7	Frame configuration register (SAI_BFRCR)	1049
29.5.8	Slot register (SAI_ASLOTR)	1051
29.5.9	Slot register (SAI_BSLOTR)	1052
29.5.10	Interrupt mask register (SAI_AIM)	1053
29.5.11	Interrupt mask register (SAI_BIM)	1054

29.5.12	Status register (SAI_ASR)	1055
29.5.13	Status register (SAI_BSR)	1057
29.5.14	Clear flag register (SAI_ACLRFR)	1059
29.5.15	Clear flag register (SAI_BCLRFR)	1060
29.5.16	Data register (SAI_ADR)	1061
29.5.17	Data register (SAI_BDR)	1062
29.5.18	SAI register map	1063
30	SD/SDIO/MMC card host interface (SDMMC)	1064
30.1	SDMMC main features	1064
30.2	SDMMC bus topology	1064
30.3	SDMMC functional description	1066
30.3.1	SDMMC adapter	1068
30.3.2	SDMMC APB2 interface	1079
30.4	Card functional description	1080
30.4.1	Card identification mode	1080
30.4.2	Card reset	1081
30.4.3	Operating voltage range validation	1081
30.4.4	Card identification process	1081
30.4.5	Block write	1082
30.4.6	Block read	1083
30.4.7	Stream access, stream write and stream read (MultiMediaCard only)	1083
30.4.8	Erase: group erase and sector erase	1085
30.4.9	Wide bus selection or deselection	1085
30.4.10	Protection management	1085
30.4.11	Card status register	1089
30.4.12	SD status register	1092
30.4.13	SD I/O mode	1096
30.4.14	Commands and responses	1097
30.5	Response formats	1100
30.5.1	R1 (normal response command)	1101
30.5.2	R1b	1101
30.5.3	R2 (CID, CSD register)	1101
30.5.4	R3 (OCR register)	1102
30.5.5	R4 (Fast I/O)	1102
30.5.6	R4b	1102

30.5.7	R5 (interrupt request)	1103
30.5.8	R6	1103
30.6	SDIO I/O card-specific operations	1104
30.6.1	SDIO I/O read wait operation by SDMMC_D2 signalling	1104
30.6.2	SDIO read wait operation by stopping SDMMC_CK	1105
30.6.3	SDIO suspend/resume operation	1105
30.6.4	SDIO interrupts	1105
30.7	HW flow control	1105
30.8	SDMMC registers	1106
30.8.1	SDMMC power control register (SDMMC_POWER)	1106
30.8.2	SDMMC clock control register (SDMMC_CLKCR)	1106
30.8.3	SDMMC argument register (SDMMC_ARG)	1108
30.8.4	SDMMC command register (SDMMC_CMD)	1108
30.8.5	SDMMC command response register (SDMMC_RESPCMD)	1109
30.8.6	SDMMC response 1..4 register (SDMMC_RESPx)	1109
30.8.7	SDMMC data timer register (SDMMC_DTIMER)	1110
30.8.8	SDMMC data length register (SDMMC_DLEN)	1111
30.8.9	SDMMC data control register (SDMMC_DCTRL)	1111
30.8.10	SDMMC data counter register (SDMMC_DCOUNT)	1114
30.8.11	SDMMC status register (SDMMC_STA)	1114
30.8.12	SDMMC interrupt clear register (SDMMC_ICR)	1115
30.8.13	SDMMC mask register (SDMMC_MASK)	1117
30.8.14	SDMMC FIFO counter register (SDMMC_FIFOCNT)	1119
30.8.15	SDMMC data FIFO register (SDMMC_FIFO)	1120
30.8.16	SDMMC register map	1121
31	Controller area network (bxCAN)	1123
31.1	Introduction	1123
31.2	bxCAN main features	1123
31.3	bxCAN general description	1123
31.3.1	CAN 2.0B active core	1124
31.3.2	Control, status and configuration registers	1124
31.3.3	Tx mailboxes	1124
31.3.4	Acceptance filters	1124
31.4	bxCAN operating modes	1125
31.4.1	Initialization mode	1125

31.4.2	Normal mode	1126
31.4.3	Sleep mode (low-power)	1126
31.5	Test mode	1127
31.5.1	Silent mode	1127
31.5.2	Loop back mode	1128
31.5.3	Loop back combined with silent mode	1128
31.6	Behavior in debug mode	1129
31.7	bxCAN functional description	1129
31.7.1	Transmission handling	1129
31.7.2	Time triggered communication mode	1131
31.7.3	Reception handling	1131
31.7.4	Identifier filtering	1132
31.7.5	Message storage	1136
31.7.6	Error management	1138
31.7.7	Bit timing	1138
31.8	bxCAN interrupts	1141
31.9	CAN registers	1142
31.9.1	Register access protection	1142
31.9.2	CAN control and status registers	1142
31.9.3	CAN mailbox registers	1152
31.9.4	CAN filter registers	1159
31.9.5	bxCAN register map	1163
32	USB on-the-go full-speed/high-speed (OTG_FS/OTG_HS)	1167
32.1	Introduction	1167
32.2	OTG main features	1168
32.2.1	General features	1169
32.2.2	Host-mode features	1170
32.2.3	Peripheral-mode features	1170
32.3	OTG implementation	1171
32.4	OTG functional description	1172
32.4.1	OTG block diagram	1172
32.4.2	USB OTG pin and internal signals	1174
32.4.3	OTG core	1175
32.4.4	Full-speed OTG PHY	1176
32.4.5	Embedded full speed OTG PHY	1176

32.4.6	High-speed OTG PHY	1177
32.5	OTG dual role device (DRD)	1177
32.5.1	ID line detection	1178
32.5.2	HNP dual role device	1178
32.5.3	SRP dual role device	1178
32.6	USB peripheral	1179
32.6.1	SRP-capable peripheral	1179
32.6.2	Peripheral states	1180
32.6.3	Peripheral endpoints	1181
32.7	USB host	1183
32.7.1	SRP-capable host	1184
32.7.2	USB host states	1184
32.7.3	Host channels	1186
32.7.4	Host scheduler	1187
32.8	SOF trigger	1188
32.8.1	Host SOFs	1188
32.8.2	Peripheral SOFs	1188
32.9	OTG low-power modes	1189
32.10	Dynamic update of the OTG_HFIR register	1190
32.11	USB data FIFOs	1190
32.11.1	Peripheral FIFO architecture	1191
32.11.2	Host FIFO architecture	1192
32.11.3	FIFO RAM allocation	1193
32.12	OTG_FS system performance	1195
32.13	OTG_FS/OTG_HS interrupts	1195
32.14	OTG_FS/OTG_HS control and status registers	1197
32.14.1	CSR memory map	1197
32.15	OTG_FS/OTG_HS registers	1203
32.15.1	OTG control and status register (OTG_GOTGCTL)	1203
32.15.2	OTG interrupt register (OTG_GOTGINT)	1206
32.15.3	OTG AHB configuration register (OTG_GAHBCFG)	1208
32.15.4	OTG USB configuration register (OTG_GUSBCFG)	1210
32.15.5	OTG reset register (OTG_GRSTCTL)	1214
32.15.6	OTG core interrupt register (OTG_GINTSTS)	1217
32.15.7	OTG interrupt mask register (OTG_GINTMSK)	1222

32.15.8	OTG receive status debug read/OTG status read and pop registers (OTG_GRXSTSR/OTG_GRXSTSP)	1225
32.15.9	OTG receive FIFO size register (OTG_GRXFSIZ)	1228
32.15.10	OTG host non-periodic transmit FIFO size register (OTG_HNPTXFSIZ)/Endpoint 0 Transmit FIFO size (OTG_DIEPTXF0)	1228
32.15.11	OTG non-periodic transmit FIFO/queue status register (OTG_HNPTXSTS)	1229
32.15.12	OTG general core configuration register (OTG_GCCFG)	1230
32.15.13	OTG core ID register (OTG_CID)	1232
32.15.14	OTG core LPM configuration register (OTG_GLPMCFG)	1232
32.15.15	OTG host periodic transmit FIFO size register (OTG_HPTXFSIZ)	1237
32.15.16	OTG device IN endpoint transmit FIFO size register (OTG_DIEPTXF _x) ($x = 1..5[\text{FS}] / 8[\text{HS}]$, where x is the FIFO number)	1237
32.15.17	Host-mode registers	1238
32.15.18	OTG host configuration register (OTG_HCFG)	1238
32.15.19	OTG host frame interval register (OTG_HFIR)	1239
32.15.20	OTG host frame number/frame time remaining register (OTG_HFNUM)	1240
32.15.21	OTG_Host periodic transmit FIFO/queue status register (OTG_HPTXSTS)	1240
32.15.22	OTG host all channels interrupt register (OTG_HAINT)	1241
32.15.23	OTG host all channels interrupt mask register (OTG_HAINTMSK)	1242
32.15.24	OTG host port control and status register (OTG_HPRT)	1243
32.15.25	OTG host channel x characteristics register (OTG_HCCHAR x) ($x = 0..15[\text{HS}] / 11[\text{FS}]$, where x = Channel number)	1245
32.15.26	OTG host channel x split control register (OTG_HCSPLTx) ($x = 0..15$, where x = Channel number)	1246
32.15.27	OTG host channel x interrupt register (OTG_HCINT x) ($x = 0..15[\text{HS}] / 11[\text{FS}]$, where x = Channel number)	1247
32.15.28	OTG host channel x interrupt mask register (OTG_HCINTMSK x) ($x = 0..15[\text{HS}] / 11[\text{FS}]$, where x = Channel number)	1249
32.15.29	OTG host channel x transfer size register (OTG_HCTSIZ x) ($x = 0..15[\text{HS}] / 11[\text{FS}]$, where x = Channel number)	1250
32.15.30	OTG host channel x DMA address register (OTG_HCDMAX) ($x = 0..15$, where x = Channel number)	1251
32.15.31	Device-mode registers	1251
32.15.32	OTG device configuration register (OTG_DCFG)	1252
32.15.33	OTG device control register (OTG_DCTL)	1254

32.15.34 OTG device status register (OTG_DSTS)	1256
32.15.35 OTG device IN endpoint common interrupt mask register (OTG_DIEPMSK)	1257
32.15.36 OTG device OUT endpoint common interrupt mask register (OTG_DOEPMSK)	1258
32.15.37 OTG device all endpoints interrupt register (OTG_DAINT)	1260
32.15.38 OTG all endpoints interrupt mask register (OTG_DAINTMSK)	1260
32.15.39 OTG device V _{BUS} discharge time register (OTG_DVBUSDIS)	1261
32.15.40 OTG device V _{BUS} pulsing time register (OTG_DVBUSPULSE)	1261
32.15.41 OTG device threshold control register (OTG_DTHRCTL)	1262
32.15.42 OTG device IN endpoint FIFO empty interrupt mask register (OTG_DIEPEMPMSK)	1263
32.15.43 OTG device each endpoint interrupt register (OTG_DEACHINT) ...	1264
32.15.44 OTG device each endpoint interrupt mask register (OTG_DEACHINTMSK)	1264
32.15.45 OTG device each IN endpoint-1 interrupt mask register (OTG_HS_DIEPEACHMSK1)	1265
32.15.46 OTG device each OUT endpoint-1 interrupt mask register (OTG_HS_DOEPEACHMSK1)	1266
32.15.47 OTG device control IN endpoint 0 control register (OTG_DIEPCTL0)	1267
32.15.48 OTG device IN endpoint x control register (OTG_DIEPCTLx) (x = 1..5[FS] / 0..8[HS], where x = endpoint number)	1269
32.15.49 OTG device IN endpoint x interrupt register (OTG_DIEPINTx) (x = 0..5[FS] /8[HS], where x = Endpoint number)	1271
32.15.50 OTG device IN endpoint 0 transfer size register (OTG_DIEPTSIZ0)	1273
32.15.51 OTG device IN endpoint x DMA address register (OTG_DIEPDMAx) (x = 0..8, where x = endpoint number)	1274
32.15.52 OTG device IN endpoint transmit FIFO status register (OTG_DTXFSTSx) (x = 0..5[FS] /8[HS], where x = endpoint number)	1274
32.15.53 OTG device IN endpoint x transfer size register (OTG_DIEPTSIZx) (x = 1..5[FS] /8[HS], where x = endpoint number)	1275
32.15.54 OTG device control OUT endpoint 0 control register (OTG_DOEPCTL0)	1276
32.15.55 OTG device OUT endpoint x interrupt register (OTG_DOEPINTx) (x = 0..5[FS] /8[HS], where x = Endpoint number)	1277
32.15.56 OTG device OUT endpoint 0 transfer size register (OTG_DOEPTSIZ0)	1279

32.15.57	OTG device OUT endpoint x DMA address register (OTG_DOEPDMA x) ($x = 0..8$, where x = endpoint number)	1280
32.15.58	OTG device OUT endpoint x control register (OTG_DOEPCTL x) ($x = 1..5$ [FS] /8[HS], where x = endpoint number)	1281
32.15.59	OTG device OUT endpoint x transfer size register (OTG_DOEPTSIZ x) ($x = 1..5$ [FS] /8[HS], where x = Endpoint number)	1283
32.15.60	OTG power and clock gating control register (OTG_PCGCCTL)	1284
32.15.61	OTG_FS/OTG_HS register map	1285
32.16	OTG_FS/OTG_HS programming model	1297
32.16.1	Core initialization	1297
32.16.2	Host initialization	1297
32.16.3	Device initialization	1298
32.16.4	DMA mode	1299
32.16.5	Host programming model	1299
32.16.6	Device programming model	1332
32.16.7	Worst case response time	1350
32.16.8	OTG programming model	1352
33	USB PHY controller (USBPHYC) available on the STM32F7x3xx and STM32F730xx devices only	1359
33.1	USBPHYC introduction	1359
33.2	USBPHYC main features	1359
33.3	USBPHYC functional description	1359
33.3.1	USBPHYC block diagram	1359
33.3.2	USBPHYC reset and clocks	1360
33.4	USBPHYC register interface	1360
33.4.1	USBPHYC PLL1 control register (USBPHYC_PLL1)	1360
33.4.2	USBPHYC tuning control register (USBPHYC_TUNE)	1361
33.4.3	USBPHYC LDO control and status register (USBPHYC_LDO)	1363
33.4.4	USBPHYC register map	1364
34	Debug support (DBG)	1365
34.1	Overview	1365
34.2	Reference ARM® documentation	1366
34.3	SWJ debug port (serial wire and JTAG)	1366
34.3.1	Mechanism to select the JTAG-DP or the SW-DP	1367
34.4	Pinout and debug port pins	1367

34.4.1	SWJ debug port pins	1368
34.4.2	Flexible SWJ-DP pin assignment	1368
34.4.3	Internal pull-up and pull-down on JTAG pins	1369
34.4.4	Using serial wire and releasing the unused debug pins as GPIOs	1370
34.5	STM32F72xxx and STM32F73xxx JTAG Debug Port connection	1370
34.6	ID codes and locking mechanism	1372
34.6.1	MCU device ID code	1372
34.6.2	Boundary scan Debug Port	1372
34.6.3	Cortex®-M7 with FPU Debug Port	1372
34.6.4	Cortex®-M7 with FPU JEDEC-106 ID code	1373
34.7	JTAG debug port	1373
34.8	SW debug port	1375
34.8.1	SW protocol introduction	1375
34.8.2	SW protocol sequence	1375
34.8.3	SW-DP state machine (reset, idle states, ID code)	1376
34.8.4	DP and AP read/write accesses	1377
34.8.5	SW-DP registers	1377
34.8.6	SW-AP registers	1378
34.9	AHB-AP (AHB access port) - valid for both JTAG-DP and SW-DP	1379
34.10	Core debug	1380
34.11	Capability of the debugger host to connect under system reset	1381
34.12	FPB (Flash patch breakpoint)	1381
34.13	DWT (data watchpoint trigger)	1382
34.14	ITM (instrumentation trace macrocell)	1382
34.14.1	General description	1382
34.14.2	Time stamp packets, synchronization and overflow packets	1382
34.15	ETM (Embedded trace macrocell)	1384
34.15.1	General description	1384
34.15.2	Signal protocol, packet types	1384
34.15.3	Main ETM registers	1385
34.15.4	Configuration example	1385
34.16	MCU debug component (DBGMCU)	1385
34.16.1	Debug support for low-power modes	1385
34.16.2	Debug support for timers, watchdog, bxCAN and I ² C	1386
34.16.3	Debug MCU configuration register	1386

34.16.4	DBGMCU_CR register	1386
34.16.5	Debug MCU APB1 freeze register (DBGMCU_APB1_FZ)	1387
34.16.6	Debug MCU APB2 Freeze register (DBGMCU_APB2_FZ)	1389
34.17	Pelican TPIU (trace port interface unit)	1390
34.17.1	Introduction	1390
34.17.2	TRACE pin assignment	1391
34.17.3	TPIU formatter	1393
34.17.4	TPIU frame synchronization packets	1393
34.17.5	Transmission of the synchronization frame packet	1393
34.17.6	Synchronous mode	1394
34.17.7	Asynchronous mode	1394
34.17.8	TRACECLKIN connection inside the STM32F72xxx and STM32F73xxx	1394
34.17.9	TPIU registers	1395
34.17.10	Example of configuration	1395
34.18	DBG register map	1396
35	Device electronic signature	1397
35.1	Unique device ID register (96 bits)	1397
35.2	Flash size	1398
35.3	Package data register	1399
36	Revision history	1400

List of tables

Table 1.	STM32F72xxx and STM32F73xxx register boundary addresses	57
Table 2.	Boot modes.....	61
Table 3.	STM32F72xxx and STM32F732xx/F733xx Flash memory organization	68
Table 4.	STM32F730xx Flash memory organization	68
Table 5.	Number of wait states according to CPU clock (HCLK) frequency.....	69
Table 6.	Program/erase parallelism	71
Table 7.	Flash interrupt request	74
Table 8.	Option byte organization.....	74
Table 9.	Access versus read protection level	80
Table 10.	Protections on sector i	83
Table 11.	OTP area organization	83
Table 12.	Flash register map and reset values.....	93
Table 13.	Voltage regulator configuration mode versus device operating mode	101
Table 14.	Low-power mode summary	106
Table 15.	Features over all modes	107
Table 16.	Sleep-now entry and exit	111
Table 17.	Sleep-on-exit entry and exit	111
Table 18.	Stop operating modes.....	112
Table 19.	Stop mode entry and exit (STM32F72xxx and STM32F73xxx)	114
Table 20.	Standby mode entry and exit	115
Table 21.	PWR - register map and reset values.....	127
Table 22.	RCC register map and reset values	190
Table 23.	Port bit configuration table	195
Table 24.	GPIO register map and reset values	208
Table 25.	SYSCFG register map and reset values.....	216
Table 26.	DMA1 request mapping	221
Table 27.	DMA2 request mapping	221
Table 28.	Source and destination address	222
Table 29.	Source and destination address registers in double-buffer mode (DBM = 1).....	227
Table 30.	Packing/unpacking and endian behavior (bit PINC = MINC = 1)	228
Table 31.	Restriction on NDT versus PSIZE and MSIZE	228
Table 32.	FIFO threshold configurations	230
Table 33.	Possible DMA configurations	234
Table 34.	DMA interrupt requests	237
Table 35.	DMA register map and reset values	248
Table 36.	STM32F72xxx and STM32F73xxx vector table	252
Table 37.	External interrupt/event controller register map and reset values.....	263
Table 38.	CRC internal input/output signals	265
Table 39.	CRC register map and reset values	269
Table 40.	NOR/PSRAM bank selection	275
Table 41.	NOR/PSRAM External memory address	275
Table 42.	NAND memory mapping and timing registers.....	275
Table 43.	NAND bank selection	275
Table 44.	SDRAM bank selection.....	276
Table 45.	SDRAM address mapping	276
Table 46.	SDRAM address mapping with 8-bit data bus width.....	277
Table 47.	SDRAM address mapping with 16-bit data bus width.....	278
Table 48.	SDRAM address mapping with 32-bit data bus width.....	278

Table 49.	Programmable NOR/PSRAM access parameters	280
Table 50.	Non-multiplexed I/O NOR Flash memory	281
Table 51.	16-bit multiplexed I/O NOR Flash memory	281
Table 52.	Non-multiplexed I/Os PSRAM/SRAM	282
Table 53.	16-Bit multiplexed I/O PSRAM	282
Table 54.	NOR Flash/PSRAM: example of supported memories and transactions	283
Table 55.	FMC_BCRx bit fields	286
Table 56.	FMC_BTRx bit fields	286
Table 57.	FMC_BCRx bit fields	288
Table 58.	FMC_BTRx bit fields	288
Table 59.	FMC_BWTRx bit fields	289
Table 60.	FMC_BCRx bit fields	291
Table 61.	FMC_BTRx bit fields	291
Table 62.	FMC_BWTRx bit fields	292
Table 63.	FMC_BCRx bit fields	293
Table 64.	FMC_BTRx bit fields	294
Table 65.	FMC_BWTRx bit fields	294
Table 66.	FMC_BCRx bit fields	296
Table 67.	FMC_BTRx bit fields	296
Table 68.	FMC_BWTRx bit fields	297
Table 69.	FMC_BCRx bit fields	298
Table 70.	FMC_BTRx bit fields	299
Table 71.	FMC_BCRx bit fields	304
Table 72.	FMC_BTRx bit fields	305
Table 73.	FMC_BCRx bit fields	306
Table 74.	FMC_BTRx bit fields	307
Table 75.	Programmable NAND Flash access parameters	315
Table 76.	8-bit NAND Flash	315
Table 77.	16-bit NAND Flash	316
Table 78.	Supported memories and transactions	317
Table 79.	ECC result relevant bits	326
Table 80.	SDRAM signals	327
Table 81.	FMC register map	345
Table 82.	QUADSPI pins	348
Table 83.	QUADSPI interrupt requests	362
Table 84.	QUADSPI register map and reset values	375
Table 85.	ADC pins	378
Table 86.	Analog watchdog channel selection	384
Table 87.	Configuring the trigger polarity	389
Table 88.	External trigger for regular channels	389
Table 89.	External trigger for injected channels	390
Table 90.	ADC interrupts	404
Table 91.	ADC global register map	420
Table 92.	ADC register map and reset values for each ADC	420
Table 93.	ADC register map and reset values (common ADC registers)	422
Table 94.	DAC pins	424
Table 95.	External triggers	427
Table 96.	DAC register map	444
Table 97.	RNG internal input/output signals	446
Table 98.	RNG interrupt requests	451
Table 99.	RNG register map and reset map	456
Table 100.	AES internal input/output signals	458

Table 101. CTR mode initialization vector definition	477
Table 102. GCM last block definition	479
Table 103. GCM mode IVI bitfield initialization	480
Table 104. Initialization of AES_IVRx registers in CCM mode	487
Table 105. Key endianness in AES_KEYRx registers (128- or 256-bit key length)	492
Table 106. DMA channel configuration for memory-to-AES data transfer	493
Table 107. DMA channel configuration for AES-to-memory data transfer	494
Table 108. AES interrupt requests	496
Table 109. Processing latency (in clock cycle) for ECB, CBC and CTR.	496
Table 110. Processing latency for GCM and CCM (in clock cycle)	496
Table 111. AES register map and reset values	508
Table 112. Behavior of timer outputs versus BRK/BRK2 inputs	551
Table 113. Counting direction versus encoder signals	558
Table 114. TIMx internal trigger connection	575
Table 115. Output control bits for complementary OCx and OCxN channels with break feature	589
Table 116. TIM1 register map and reset values	601
Table 117. TIM8 register map and reset values	603
Table 118. Counting direction versus encoder signals	639
Table 119. TIMx internal trigger connection	656
Table 120. Output control bit for standard OCx channels	666
Table 121. TIM2/TIM3/TIM4/TIM5 register map and reset values	672
Table 122. TIMx internal trigger connection	701
Table 123. Output control bit for standard OCx channels	709
Table 124. TIM9/TIM12 register map and reset values	712
Table 125. Output control bit for standard OCx channels	720
Table 126. TIM10/TIM11/TIM13/TIM14 register map and reset values	722
Table 127. TIM6/TIM7 register map and reset values	736
Table 128. STM32F72xxx and STM32F73xxx LPTIM features	737
Table 129. LPTIM1 external trigger connection	738
Table 130. Prescaler division ratios	740
Table 131. Encoder counting scenarios	746
Table 132. Effect of low-power modes on the LPTIM	747
Table 133. Interrupt events	748
Table 134. LPTIM register map and reset values	758
Table 135. IWDG register map and reset values	767
Table 136. WWWDG register map and reset values	774
Table 137. RTC pin PC13 configuration	778
Table 138. RTC pin PI8 configuration	779
Table 139. RTC pin PC2 configuration	780
Table 140. RTC functions over modes	780
Table 141. Effect of low-power modes on RTC	793
Table 142. Interrupt control bits	794
Table 143. RTC register map and reset values	818
Table 144. STM32F72xxx and STM32F73xxx I2C implementation	821
Table 145. Comparison of analog vs. digital filters	824
Table 146. I2C-SMBUS specification data setup and hold times	827
Table 147. I2C configuration	831
Table 148. I2C-SMBUS specification clock timings	842
Table 149. Examples of timing settings for fI2CCLK = 8 MHz	852
Table 150. Examples of timing settings for fI2CCLK = 16 MHz	852
Table 151. Examples of timing settings for fI2CCLK = 48 MHz	853
Table 152. SMBus timeout specifications	855

Table 153. SMBUS with PEC configuration	857
Table 154. Examples of TIMEOUTA settings for various I2CCLK frequencies (max $t_{TIMEOUT} = 25$ ms)	858
Table 155. Examples of TIMEOUTB settings for various I2CCLK frequencies	858
Table 156. Examples of TIMEOUTA settings for various I2CCLK frequencies (max $t_{IDLE} = 50$ μ s)	858
Table 157. Effect of low-power modes on the I2C	868
Table 158. I2C Interrupt requests	869
Table 159. I2C register map and reset values	884
Table 160. STM32F72xxx and STM32F73xxx USART features	888
Table 161. Noise detection from sampled data	900
Table 162. Error calculation for programmed baud rates at $f_{CK} = 216$ MHz in both cases of oversampling by 8 (OVER8 = 1)	903
Table 163. Error calculation for programmed baud rates at $f_{CK} = 216$ MHz in both cases of oversampling by 16 (OVER8 = 0)	903
Table 164. Tolerance of the USART receiver when BRR [3:0] = 0000	904
Table 165. Tolerance of the USART receiver when BRR [3:0] is different from 0000	904
Table 166. Frame formats	909
Table 167. Effect of low-power modes on the USART	926
Table 168. USART interrupt requests	927
Table 169. USART register map and reset values	949
Table 170. STM32F72xxx and STM32F73xxx SPI implementation	952
Table 171. SPI interrupt requests	977
Table 172. Audio-frequency precision using standard 8 MHz HSE	991
Table 173. I2S interrupt requests	997
Table 174. SPI register map and reset values	1009
Table 175. SAI internal input/output signals	1013
Table 176. SAI input/output pins	1013
Table 177. External synchronization selection	1015
Table 178. Example of possible audio frequency sampling range	1022
Table 179. SOPD pattern	1028
Table 180. Parity bit calculation	1028
Table 181. Audio sampling frequency versus symbol rates	1029
Table 182. SAI interrupt sources	1038
Table 183. SAI register map and reset values	1063
Table 184. SDMMC I/O definitions	1067
Table 185. Command format	1072
Table 186. Short response format	1073
Table 187. Long response format	1073
Table 188. Command path status flags	1073
Table 189. Data token format	1076
Table 190. DPSM flags	1077
Table 191. Transmit FIFO status flags	1078
Table 192. Receive FIFO status flags	1078
Table 193. Card status	1089
Table 194. SD status	1092
Table 195. Speed class code field	1093
Table 196. Performance move field	1094
Table 197. AU_SIZE field	1094
Table 198. Maximum AU size	1094
Table 199. Erase size field	1095
Table 200. Erase timeout field	1095

Table 201. Erase offset field	1095
Table 202. Block-oriented write commands	1098
Table 203. Block-oriented write protection commands	1099
Table 204. Erase commands	1099
Table 205. I/O mode commands	1099
Table 206. Lock card	1100
Table 207. Application-specific commands	1100
Table 208. R1 response	1101
Table 209. R2 response	1101
Table 210. R3 response	1102
Table 211. R4 response	1102
Table 212. R4b response	1102
Table 213. R5 response	1103
Table 214. R6 response	1104
Table 215. Response type and SDMMC_RESPx registers	1110
Table 216. SDMMC register map	1121
Table 217. Transmit mailbox mapping	1137
Table 218. Receive mailbox mapping	1137
Table 219. bxCAN register map and reset values	1163
Table 220. OTG_HS speeds supported	1168
Table 221. OTG_FS speeds supported	1168
Table 222. OTG implementation	1171
Table 223. OTG_FS input/output pins	1174
Table 224. OTG_HS input/output pins	1175
Table 225. OTG_FS/OTG_HS input/output signals	1175
Table 226. Compatibility of STM32 low power modes with the OTG	1189
Table 227. Core global control and status registers (CSRs)	1197
Table 228. Host-mode control and status registers (CSRs)	1198
Table 229. Device-mode control and status registers	1200
Table 230. Data FIFO (DFIFO) access register map	1202
Table 231. Power and clock gating control and status registers	1203
Table 232. TRDT values (FS)	1213
Table 233. TRDT values (HS)	1213
Table 234. Minimum duration for soft disconnect	1255
Table 235. OTG_FS/OTG_HS register map and reset values	1285
Table 236. USBPHYC register map and reset values	1364
Table 237. SWJ debug port pins	1368
Table 238. Flexible SWJ-DP pin assignment	1368
Table 239. JTAG debug port data registers	1373
Table 240. 32-bit debug port registers addressed through the shifted value A[3:2]	1375
Table 241. Packet request (8-bits)	1376
Table 242. ACK response (3 bits)	1376
Table 243. DATA transfer (33 bits)	1376
Table 244. SW-DP registers	1377
Table 245. Cortex®-M7 with FPU AHB-AP registers	1379
Table 246. Core debug registers	1380
Table 247. Main ITM registers	1383
Table 248. Asynchronous TRACE pin assignment	1391
Table 249. Synchronous TRACE pin assignment	1391
Table 250. Flexible TRACE pin assignment	1392
Table 251. Important TPIU registers	1395
Table 252. DBG register map and reset values	1396

Table 253. Document revision history	1400
--	------

List of figures

Figure 1.	Memory map	56
Figure 2.	System architecture for STM32F72xxx and STM32F73xxx devices	62
Figure 3.	Flash memory interface connection inside system architecture (STM32F72xxx and STM32F73xxx)	66
Figure 4.	RDP levels	81
Figure 5.	STM32F7x2xx and STM32F730xx power supply overview	94
Figure 6.	STM32F7x3xx and STM32F730xx power supply overview	95
Figure 7.	VDDUSB connected to VDD power supply	97
Figure 8.	VDDUSB connected to external independent power supply	97
Figure 9.	Backup domain	100
Figure 10.	Power-on reset/power-down reset waveform	103
Figure 11.	BOR thresholds	104
Figure 12.	PVD thresholds	105
Figure 13.	Simplified diagram of the reset circuit	129
Figure 14.	Clock tree	130
Figure 15.	HSE/ LSE clock sources	133
Figure 16.	Frequency measurement with TIM5 in Input capture mode	138
Figure 17.	Frequency measurement with TIM11 in Input capture mode	139
Figure 18.	Basic structure of an I/O port bit	194
Figure 19.	Basic structure of a 5-Volt tolerant I/O port bit	194
Figure 20.	Input floating/pull up/pull down configurations	199
Figure 21.	Output configuration	200
Figure 22.	Alternate function configuration	200
Figure 23.	High impedance-analog configuration	201
Figure 24.	DMA block diagram	219
Figure 25.	Channel selection	220
Figure 26.	Peripheral-to-memory mode	223
Figure 27.	Memory-to-peripheral mode	224
Figure 28.	Memory-to-memory mode	225
Figure 29.	FIFO structure	230
Figure 30.	External interrupt/event controller block diagram	257
Figure 31.	External interrupt/event GPIO mapping	259
Figure 32.	CRC calculation unit block diagram	265
Figure 33.	FMC block diagram	271
Figure 34.	FMC memory banks	274
Figure 35.	Mode1 read access waveforms	285
Figure 36.	Mode1 write access waveforms	285
Figure 37.	ModeA read access waveforms	287
Figure 38.	ModeA write access waveforms	287
Figure 39.	Mode2 and mode B read access waveforms	289
Figure 40.	Mode2 write access waveforms	290
Figure 41.	ModeB write access waveforms	290
Figure 42.	ModeC read access waveforms	292
Figure 43.	ModeC write access waveforms	293
Figure 44.	ModeD read access waveforms	295
Figure 45.	ModeD write access waveforms	295
Figure 46.	Muxed read access waveforms	297
Figure 47.	Muxed write access waveforms	298

Figure 48.	Asynchronous wait during a read access waveforms	300
Figure 49.	Asynchronous wait during a write access waveforms	301
Figure 50.	Wait configuration waveforms	303
Figure 51.	Synchronous multiplexed read mode waveforms - NOR, PSRAM (CRAM)	304
Figure 52.	Synchronous multiplexed write mode waveforms - PSRAM (CRAM)	306
Figure 53.	NAND Flash controller waveforms for common memory access	318
Figure 54.	Access to non 'CE don't care' NAND-Flash	319
Figure 55.	Burst write SDRAM access waveforms	329
Figure 56.	Burst read SDRAM access	330
Figure 57.	Logic diagram of Read access with RBURST bit set (CAS=1, RPIPE=0)	331
Figure 58.	Read access crossing row boundary	333
Figure 59.	Write access crossing row boundary	333
Figure 60.	Self-refresh mode	336
Figure 61.	Power-down mode	337
Figure 62.	QUADSPI block diagram when dual-flash mode is disabled	347
Figure 63.	QUADSPI block diagram when dual-flash mode is enabled	348
Figure 64.	An example of a read command in quad mode	349
Figure 65.	An example of a DDR command in quad mode	352
Figure 66.	nCS when CKMODE = 0 (T = CLK period)	360
Figure 67.	nCS when CKMODE = 1 in SDR mode (T = CLK period)	360
Figure 68.	nCS when CKMODE = 1 in DDR mode (T = CLK period)	361
Figure 69.	nCS when CKMODE = 1 with an abort (T = CLK period)	361
Figure 70.	Single ADC block diagram	377
Figure 71.	ADC1 connectivity	379
Figure 72.	ADC2 connectivity	380
Figure 73.	ADC3 connectivity	381
Figure 74.	Timing diagram	384
Figure 75.	Analog watchdog's guarded area	384
Figure 76.	Injected conversion latency	386
Figure 77.	Right alignment of 12-bit data	388
Figure 78.	Left alignment of 12-bit data	388
Figure 79.	Left alignment of 6-bit data	388
Figure 80.	Multi ADC block diagram ⁽¹⁾	393
Figure 81.	Injected simultaneous mode on 4 channels: dual ADC mode	396
Figure 82.	Injected simultaneous mode on 4 channels: triple ADC mode	396
Figure 83.	Regular simultaneous mode on 16 channels: dual ADC mode	397
Figure 84.	Regular simultaneous mode on 16 channels: triple ADC mode	397
Figure 85.	Interleaved mode on 1 channel in continuous conversion mode: dual ADC mode	398
Figure 86.	Interleaved mode on 1 channel in continuous conversion mode: triple ADC mode	399
Figure 87.	Alternate trigger: injected group of each ADC	400
Figure 88.	Alternate trigger: 4 injected channels (each ADC) in discontinuous mode	400
Figure 89.	Alternate trigger: injected group of each ADC	401
Figure 90.	Alternate + regular simultaneous	402
Figure 91.	Case of trigger occurring during injected conversion	402
Figure 92.	Temperature sensor and VREFINT channel block diagram	403
Figure 93.	DAC channel block diagram	424
Figure 94.	DAC output buffer connection	425
Figure 95.	Data registers in single DAC channel mode	426
Figure 96.	Data registers in dual DAC channel mode	426
Figure 97.	Timing diagram for conversion with trigger disabled TEN = 0	427
Figure 98.	DAC LFSR register calculation algorithm	429
Figure 99.	DAC conversion (SW trigger enabled) with LFSR wave generation	429

Figure 100. DAC triangle wave generation	430
Figure 101. DAC conversion (SW trigger enabled) with triangle wave generation	430
Figure 102. RNG block diagram	446
Figure 103. Entropy source model	447
Figure 104. AES block diagram	458
Figure 105. ECB encryption and decryption principle	460
Figure 106. CBC encryption and decryption principle	461
Figure 107. CTR encryption and decryption principle	462
Figure 108. GCM encryption and authentication principle	463
Figure 109. GMAC authentication principle	463
Figure 110. CCM encryption and authentication principle	464
Figure 111. STM32 cryptolib AES flowchart examples	465
Figure 112. STM32 cryptolib AES flowchart examples (continued)	466
Figure 113. Encryption key derivation for ECB/CBC decryption (Mode 2)	469
Figure 114. Example of suspend mode management	470
Figure 115. ECB encryption	471
Figure 116. ECB decryption	471
Figure 117. CBC encryption	472
Figure 118. CBC decryption	472
Figure 119. ECB/CBC encryption (Mode 1)	473
Figure 120. ECB/CBC decryption (Mode 3)	474
Figure 121. Message construction in CTR mode	476
Figure 122. CTR encryption	477
Figure 123. CTR decryption	477
Figure 124. Message construction in GCM	479
Figure 125. GCM authenticated encryption	480
Figure 126. Message construction in GMAC mode	484
Figure 127. GMAC authentication mode	484
Figure 128. Message construction in CCM mode	485
Figure 129. CCM mode authenticated decryption	487
Figure 130. 128-bit block construction with respect to data swap	491
Figure 131. DMA transfer of a 128-bit data block during input phase	493
Figure 132. DMA transfer of a 128-bit data block during output phase	494
Figure 133. AES interrupt signal generation	495
Figure 134. Advanced-control timer block diagram	511
Figure 135. Counter timing diagram with prescaler division change from 1 to 2	513
Figure 136. Counter timing diagram with prescaler division change from 1 to 4	513
Figure 137. Counter timing diagram, internal clock divided by 1	515
Figure 138. Counter timing diagram, internal clock divided by 2	515
Figure 139. Counter timing diagram, internal clock divided by 4	516
Figure 140. Counter timing diagram, internal clock divided by N	516
Figure 141. Counter timing diagram, update event when ARPE=0 (TIMx_ARR not preloaded)	517
Figure 142. Counter timing diagram, update event when ARPE=1 (TIMx_ARR preloaded)	517
Figure 143. Counter timing diagram, internal clock divided by 1	519
Figure 144. Counter timing diagram, internal clock divided by 2	519
Figure 145. Counter timing diagram, internal clock divided by 4	520
Figure 146. Counter timing diagram, internal clock divided by N	520
Figure 147. Counter timing diagram, update event when repetition counter is not used	521
Figure 148. Counter timing diagram, internal clock divided by 1, TIMx_ARR = 0x6	522
Figure 149. Counter timing diagram, internal clock divided by 2	523
Figure 150. Counter timing diagram, internal clock divided by 4, TIMx_ARR=0x36	523
Figure 151. Counter timing diagram, internal clock divided by N	524

Figure 152. Counter timing diagram, update event with ARPE=1 (counter underflow)	524
Figure 153. Counter timing diagram, Update event with ARPE=1 (counter overflow)	525
Figure 154. Update rate examples depending on mode and TIMx_RCR register settings	526
Figure 155. External trigger input block	527
Figure 156. Control circuit in normal mode, internal clock divided by 1	528
Figure 157. TI2 external clock connection example	529
Figure 158. Control circuit in external clock mode 1	530
Figure 159. External trigger input block	530
Figure 160. Control circuit in external clock mode 2	531
Figure 161. Capture/compare channel (example: channel 1 input stage)	532
Figure 162. Capture/compare channel 1 main circuit	533
Figure 163. Output stage of capture/compare channel (channel 1, idem ch. 2 and 3)	533
Figure 164. Output stage of capture/compare channel (channel 4)	534
Figure 165. Output stage of capture/compare channel (channel 5, idem ch. 6)	534
Figure 166. PWM input mode timing	536
Figure 167. Output compare mode, toggle on OC1	538
Figure 168. Edge-aligned PWM waveforms (ARR=8)	539
Figure 169. Center-aligned PWM waveforms (ARR=8)	540
Figure 170. Generation of 2 phase-shifted PWM signals with 50% duty cycle	542
Figure 171. Combined PWM mode on channel 1 and 3	543
Figure 172. 3-phase combined PWM signals with multiple trigger pulses per period	544
Figure 173. Complementary output with dead-time insertion	545
Figure 174. Dead-time waveforms with delay greater than the negative pulse	545
Figure 175. Dead-time waveforms with delay greater than the positive pulse	546
Figure 176. Break and Break2 circuitry overview	548
Figure 177. Various output behavior in response to a break event on BRK (OSSI = 1)	550
Figure 178. PWM output state following BRK and BRK2 pins assertion (OSSI=1)	551
Figure 179. PWM output state following BRK assertion (OSSI=0)	552
Figure 180. Clearing TIMx OCxREF	553
Figure 181. 6-step generation, COM example (OSSR=1)	554
Figure 182. Example of one pulse mode	555
Figure 183. Retriggerable one pulse mode	557
Figure 184. Example of counter operation in encoder interface mode	558
Figure 185. Example of encoder interface mode with TI1FP1 polarity inverted	559
Figure 186. Measuring time interval between edges on 3 signals	560
Figure 187. Example of Hall sensor interface	562
Figure 188. Control circuit in reset mode	563
Figure 189. Control circuit in Gated mode	564
Figure 190. Control circuit in trigger mode	565
Figure 191. Control circuit in external clock mode 2 + trigger mode	566
Figure 192. General-purpose timer block diagram	606
Figure 193. Counter timing diagram with prescaler division change from 1 to 2	608
Figure 194. Counter timing diagram with prescaler division change from 1 to 4	608
Figure 195. Counter timing diagram, internal clock divided by 1	609
Figure 196. Counter timing diagram, internal clock divided by 2	610
Figure 197. Counter timing diagram, internal clock divided by 4	610
Figure 198. Counter timing diagram, internal clock divided by N	611
Figure 199. Counter timing diagram, Update event when ARPE=0 (TIMx_ARR not preloaded)	611
Figure 200. Counter timing diagram, Update event when ARPE=1 (TIMx_ARR preloaded)	612
Figure 201. Counter timing diagram, internal clock divided by 1	613
Figure 202. Counter timing diagram, internal clock divided by 2	613
Figure 203. Counter timing diagram, internal clock divided by 4	614

Figure 204. Counter timing diagram, internal clock divided by N.....	614
Figure 205. Counter timing diagram, Update event when repetition counter is not used	615
Figure 206. Counter timing diagram, internal clock divided by 1, TIMx_ARR=0x6	616
Figure 207. Counter timing diagram, internal clock divided by 2	617
Figure 208. Counter timing diagram, internal clock divided by 4, TIMx_ARR=0x36	617
Figure 209. Counter timing diagram, internal clock divided by N.....	618
Figure 210. Counter timing diagram, Update event with ARPE=1 (counter underflow).....	618
Figure 211. Counter timing diagram, Update event with ARPE=1 (counter overflow).....	619
Figure 212. Control circuit in normal mode, internal clock divided by 1.....	620
Figure 213. TI2 external clock connection example.....	620
Figure 214. Control circuit in external clock mode 1	621
Figure 215. External trigger input block	622
Figure 216. Control circuit in external clock mode 2	623
Figure 217. Capture/Compare channel (example: channel 1 input stage)	624
Figure 218. Capture/Compare channel 1 main circuit	624
Figure 219. Output stage of Capture/Compare channel (channel 1)	625
Figure 220. PWM input mode timing	627
Figure 221. Output compare mode, toggle on OC1.....	629
Figure 222. Edge-aligned PWM waveforms (ARR=8)	630
Figure 223. Center-aligned PWM waveforms (ARR=8).	631
Figure 224. Generation of 2 phase-shifted PWM signals with 50% duty cycle	632
Figure 225. Combined PWM mode on channels 1 and 3	634
Figure 226. Clearing TIMx OCxREF	635
Figure 227. Example of one-pulse mode.	636
Figure 228. Retriggerable one pulse mode	638
Figure 229. Example of counter operation in encoder interface mode	639
Figure 230. Example of encoder interface mode with TI1FP1 polarity inverted	640
Figure 231. Control circuit in reset mode	641
Figure 232. Control circuit in gated mode	642
Figure 233. Control circuit in trigger mode.....	643
Figure 234. Control circuit in external clock mode 2 + trigger mode	644
Figure 235. Master/Slave timer example	644
Figure 236. Gating TIM with OC1REF of TIM3	645
Figure 237. Gating TIM with Enable of TIM3	646
Figure 238. Triggering TIM with update of TIM3	647
Figure 239. Triggering TIM with Enable of TIM3	647
Figure 240. Triggering TIM3 and TIM2 with TIM3 TI1 input.	648
Figure 241. General-purpose timer block diagram (TIM9/TIM12)	676
Figure 242. General-purpose timer block diagram (TIM10/TIM11/TIM13/TIM14)	677
Figure 243. Counter timing diagram with prescaler division change from 1 to 2	679
Figure 244. Counter timing diagram with prescaler division change from 1 to 4	679
Figure 245. Counter timing diagram, internal clock divided by 1	680
Figure 246. Counter timing diagram, internal clock divided by 2	681
Figure 247. Counter timing diagram, internal clock divided by 4	681
Figure 248. Counter timing diagram, internal clock divided by N.....	682
Figure 249. Counter timing diagram, update event when ARPE=0 (TIMx_ARR not preloaded).....	682
Figure 250. Counter timing diagram, update event when ARPE=1 (TIMx_ARR preloaded).....	683
Figure 251. Control circuit in normal mode, internal clock divided by 1	684
Figure 252. TI2 external clock connection example.....	684

Figure 253. Control circuit in external clock mode 1	685
Figure 254. Capture/compare channel (example: channel 1 input stage	686
Figure 255. Capture/compare channel 1 main circuit	686
Figure 256. Output stage of capture/compare channel (channel 1).	687
Figure 257. PWM input mode timing	689
Figure 258. Output compare mode, toggle on OC1	690
Figure 259. Edge-aligned PWM waveforms (ARR=8)	691
Figure 260. Combined PWM mode on channel 1 and 2	692
Figure 261. Example of one pulse mode.	693
Figure 262. Retriggerable one pulse mode	695
Figure 263. Control circuit in reset mode	696
Figure 264. Control circuit in gated mode	697
Figure 265. Control circuit in trigger mode.	697
Figure 266. Basic timer block diagram.	724
Figure 267. Counter timing diagram with prescaler division change from 1 to 2	726
Figure 268. Counter timing diagram with prescaler division change from 1 to 4	726
Figure 269. Counter timing diagram, internal clock divided by 1	727
Figure 270. Counter timing diagram, internal clock divided by 2	728
Figure 271. Counter timing diagram, internal clock divided by 4	728
Figure 272. Counter timing diagram, internal clock divided by N.	729
Figure 273. Counter timing diagram, update event when ARPE = 0 (TIMx_ARR not preloaded).	729
Figure 274. Counter timing diagram, update event when ARPE=1 (TIMx_ARR preloaded).	730
Figure 275. Control circuit in normal mode, internal clock divided by 1.	731
Figure 276. Low-power timer block diagram	738
Figure 277. Glitch filter timing diagram	740
Figure 278. LPTIM output waveform, single counting mode configuration	741
Figure 279. LPTIM output waveform, Single counting mode configuration and Set-once mode activated (WAVE bit is set)	742
Figure 280. LPTIM output waveform, Continuous counting mode configuration	742
Figure 281. Waveform generation	744
Figure 282. Encoder mode counting sequence	747
Figure 283. Independent watchdog block diagram	759
Figure 284. Watchdog block diagram	769
Figure 285. Window watchdog timing diagram	770
Figure 286. RTC block diagram	777
Figure 287. I2C block diagram	822
Figure 288. I2C bus protocol	824
Figure 289. Setup and hold timings	825
Figure 290. I2C initialization flowchart	828
Figure 291. Data reception	829
Figure 292. Data transmission	830
Figure 293. Slave initialization flowchart	833
Figure 294. Transfer sequence flowchart for I2C slave transmitter, NOSTRETCH=0.	835
Figure 295. Transfer sequence flowchart for I2C slave transmitter, NOSTRETCH=1.	836
Figure 296. Transfer bus diagrams for I2C slave transmitter	837
Figure 297. Transfer sequence flowchart for slave receiver with NOSTRETCH=0	838
Figure 298. Transfer sequence flowchart for slave receiver with NOSTRETCH=1	839
Figure 299. Transfer bus diagrams for I2C slave receiver	839
Figure 300. Master clock generation	841
Figure 301. Master initialization flowchart	843

Figure 302. 10-bit address read access with HEAD10R=0	843
Figure 303. 10-bit address read access with HEAD10R=1	844
Figure 304. Transfer sequence flowchart for I2C master transmitter for N≤255 bytes	845
Figure 305. Transfer sequence flowchart for I2C master transmitter for N>255 bytes	846
Figure 306. Transfer bus diagrams for I2C master transmitter	847
Figure 307. Transfer sequence flowchart for I2C master receiver for N≤255 bytes	849
Figure 308. Transfer sequence flowchart for I2C master receiver for N >255 bytes	850
Figure 309. Transfer bus diagrams for I2C master receiver	851
Figure 310. Timeout intervals for $t_{LOW:SEXT}$, $t_{LOW:MEXT}$	855
Figure 311. Transfer sequence flowchart for SMBus slave transmitter N bytes + PEC	859
Figure 312. Transfer bus diagrams for SMBus slave transmitter (SBC=1)	860
Figure 313. Transfer sequence flowchart for SMBus slave receiver N Bytes + PEC	861
Figure 314. Bus transfer diagrams for SMBus slave receiver (SBC=1)	862
Figure 315. Bus transfer diagrams for SMBus master transmitter	863
Figure 316. Bus transfer diagrams for SMBus master receiver	865
Figure 317. USART block diagram	890
Figure 318. Word length programming	892
Figure 319. Configurable stop bits	894
Figure 320. TC/TXE behavior when transmitting	895
Figure 321. Start bit detection when oversampling by 16 or 8	896
Figure 322. Data sampling when oversampling by 16	899
Figure 323. Data sampling when oversampling by 8	900
Figure 324. Mute mode using Idle line detection	907
Figure 325. Mute mode using address mark detection	908
Figure 326. Break detection in LIN mode (11-bit break length - LBDL bit is set)	911
Figure 327. Break detection in LIN mode vs. Framing error detection	912
Figure 328. USART example of synchronous transmission	913
Figure 329. USART data clock timing diagram (M bits = 00)	913
Figure 330. USART data clock timing diagram (M bits = 01)	914
Figure 331. RX data setup/hold time	914
Figure 332. ISO 7816-3 asynchronous protocol	916
Figure 333. Parity error detection using the 1.5 stop bits	917
Figure 334. IrDA SIR ENDEC- block diagram	921
Figure 335. IrDA data modulation (3/16) -Normal Mode	922
Figure 336. Transmission using DMA	923
Figure 337. Reception using DMA	924
Figure 338. Hardware flow control between 2 USARTs	924
Figure 339. RS232 RTS flow control	925
Figure 340. RS232 CTS flow control	926
Figure 341. USART interrupt mapping diagram	928
Figure 342. SPI block diagram	953
Figure 343. Full-duplex single master/ single slave application	954
Figure 344. Half-duplex single master/ single slave application	955
Figure 345. Simplex single master/single slave application (master in transmit-only/ slave in receive-only mode)	956
Figure 346. Master and three independent slaves	957
Figure 347. Multi-master application	958
Figure 348. Hardware/software slave select management	959
Figure 349. Data clock timing diagram	960
Figure 350. Data alignment when data length is not equal to 8-bit or 16-bit	961
Figure 351. Packing data in FIFO for transmission and reception	965
Figure 352. Master full-duplex communication	968

Figure 353. Slave full-duplex communication	969
Figure 354. Master full-duplex communication with CRC	970
Figure 355. Master full-duplex communication in packed mode	971
Figure 356. NSSP pulse generation in Motorola SPI master mode	974
Figure 357. TI mode transfer	975
Figure 358. I ² S block diagram	978
Figure 359. Full-duplex communication	980
Figure 360. I ² S Philips protocol waveforms (16/32-bit full accuracy)	981
Figure 361. I ² S Philips standard waveforms (24-bit frame)	981
Figure 362. Transmitting 0x8EAA33	982
Figure 363. Receiving 0x8EAA33	982
Figure 364. I ² S Philips standard (16-bit extended to 32-bit packet frame)	982
Figure 365. Example of 16-bit data frame extended to 32-bit channel frame	982
Figure 366. MSB Justified 16-bit or 32-bit full-accuracy length	983
Figure 367. MSB justified 24-bit frame length	983
Figure 368. MSB justified 16-bit extended to 32-bit packet frame	984
Figure 369. LSB justified 16-bit or 32-bit full-accuracy	984
Figure 370. LSB justified 24-bit frame length	984
Figure 371. Operations required to transmit 0x3478AE	985
Figure 372. Operations required to receive 0x3478AE	985
Figure 373. LSB justified 16-bit extended to 32-bit packet frame	985
Figure 374. Example of 16-bit data frame extended to 32-bit channel frame	986
Figure 375. PCM standard waveforms (16-bit)	986
Figure 376. PCM standard waveforms (16-bit extended to 32-bit packet frame)	987
Figure 377. Start sequence in master mode	988
Figure 378. Audio sampling frequency definition	989
Figure 379. I ² S clock generator architecture	989
Figure 380. SAI functional block diagram	1012
Figure 381. Audio frame	1016
Figure 382. FS role is start of frame + channel side identification (FSDEF = TRIS = 1)	1018
Figure 383. FS role is start of frame (FSDEF = 0)	1019
Figure 384. Slot size configuration with FBOFF = 0 in SAI_xSLOTR	1020
Figure 385. First bit offset	1020
Figure 386. Audio block clock generator overview	1021
Figure 387. AC'97 audio frame	1025
Figure 388. Example of typical AC'97 configuration on devices featuring at least 2 embedded SAIs (three external AC'97 decoders)	1026
Figure 389. SPDIF format	1027
Figure 390. SAI_xDR register ordering	1028
Figure 391. Data companding hardware in an audio block in the SAI	1031
Figure 392. Tristate strategy on SD output line on an inactive slot	1033
Figure 393. Tristate on output data line in a protocol like I ² S	1034
Figure 394. Overrun detection error	1035
Figure 395. FIFO underrun event	1035
Figure 396. "No response" and "no data" operations	1065
Figure 397. (Multiple) block read operation	1065
Figure 398. (Multiple) block write operation	1065
Figure 399. Sequential read operation	1066
Figure 400. Sequential write operation	1066
Figure 401. SDMMC block diagram	1066
Figure 402. SDMMC adapter	1068
Figure 403. Control unit	1069

Figure 404. SDMMC_CK clock dephasing (BYPASS = 0)	1070
Figure 405. SDMMC adapter command path	1070
Figure 406. Command path state machine (SDMMC)	1071
Figure 407. SDMMC command transfer	1072
Figure 408. Data path	1074
Figure 409. Data path state machine (DPSM)	1075
Figure 410. CAN network topology	1124
Figure 411. Single-CAN block diagram	1125
Figure 412. bxCAN operating modes	1127
Figure 413. bxCAN in silent mode	1128
Figure 414. bxCAN in loop back mode	1128
Figure 415. bxCAN in combined mode	1129
Figure 416. Transmit mailbox states	1130
Figure 417. Receive FIFO states	1131
Figure 418. Filter bank scale configuration - register organization	1134
Figure 419. Example of filter numbering	1135
Figure 420. Filtering mechanism - example	1136
Figure 421. CAN error state diagram	1137
Figure 422. Bit timing	1139
Figure 423. CAN frames	1140
Figure 424. Event flags and interrupt generation	1141
Figure 425. CAN mailbox registers	1153
Figure 426. OTG full-speed block diagram	1172
Figure 427. OTG high-speed block diagram for STM32F7x2xx	1173
Figure 428. OTG high-speed block diagram for STM32F7x3xx	1174
Figure 429. OTG_FS A-B device connection	1177
Figure 430. USB_FS peripheral-only connection	1179
Figure 431. USB_FS host-only connection	1184
Figure 432. SOF connectivity (SOF trigger output to TIM and ITR1 connection)	1188
Figure 433. Updating OTG_HFIR dynamically (RLDCTRL = 0)	1190
Figure 434. Device-mode FIFO address mapping and AHB FIFO access mapping	1191
Figure 435. Host-mode FIFO address mapping and AHB FIFO access mapping	1192
Figure 436. Interrupt hierarchy	1196
Figure 437. Transmit FIFO write task	1301
Figure 438. Receive FIFO read task	1302
Figure 439. Normal bulk/control OUT/SETUP	1304
Figure 440. Bulk/control IN transactions	1308
Figure 441. Normal interrupt OUT	1311
Figure 442. Normal interrupt IN	1316
Figure 443. Isochronous OUT transactions	1318
Figure 444. Isochronous IN transactions	1321
Figure 445. Normal bulk/control OUT/SETUP transactions - DMA	1323
Figure 446. Normal bulk/control IN transaction - DMA	1325
Figure 447. Normal interrupt OUT transactions - DMA mode	1326
Figure 448. Normal interrupt IN transactions - DMA mode	1327
Figure 449. Normal isochronous OUT transaction - DMA mode	1328
Figure 450. Normal isochronous IN transactions - DMA mode	1329
Figure 451. Receive FIFO packet read	1335
Figure 452. Processing a SETUP packet	1337
Figure 453. Bulk OUT transaction	1343
Figure 454. TRDT max timing case	1352
Figure 455. A-device SRP	1353

Figure 456. B-device SRP	1354
Figure 457. A-device HNP	1355
Figure 458. B-device HNP	1357
Figure 459. USBPHYC block diagram	1360
Figure 460. Block diagram of STM32 MCU and Cortex®-M7 with FPU -level debug support	1365
Figure 461. SWJ debug port	1367
Figure 462. JTAG Debug Port connections	1371
Figure 463. TPIU block diagram	1390

1 Documentation conventions

1.1 General information

The STM32F72xxx and STM32F73xxx devices have an Arm^{®(a)} Cortex[®]-M7 core



1.2 List of abbreviations for registers

The following abbreviations^(b) are used in register descriptions:

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

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

1.3 Glossary

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

- The CPU core integrates two debug ports:
 - JTAG debug port (**JTAG-DP**) provides a 5-pin standard interface based on the Joint Test Action Group (JTAG) protocol.
 - SWD debug port (**SWD-DP**) provides a 2-pin (clock and data) interface based on the Serial Wire Debug (SWD) protocol.
For both the JTAG and SWD protocols, refer to the Cortex®-M7 Technical Reference Manual.
- **Word**: data of 32-bit length.
- **Half-word**: data of 16-bit length.
- **Byte**: data of 8-bit length.
- **Double word**: data of 64-bit length.
- **IAP (in-application programming)**: IAP is the ability to re-program the Flash memory of a microcontroller while the user program is running.
- **ICP (in-circuit programming)**: ICP is the ability to program the Flash memory of a microcontroller using the JTAG protocol, the SWD protocol or the bootloader while the device is mounted on the user application board.
- **Option bytes**: product configuration bits stored in the Flash memory.
- **AHB**: advanced high-performance bus.
- **AHBS**: AHB Slave bus.
- **AXIM**: AXI master bus.
- **ITCM**: Instruction Tightly Coupled Memory.
- **DTCM**: Data Tightly Coupled Memory.
- **CPU**: refers to the Cortex®-M7 core.

1.4 Availability of peripherals

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

1.5 Memory organization

1.5.1 Introduction

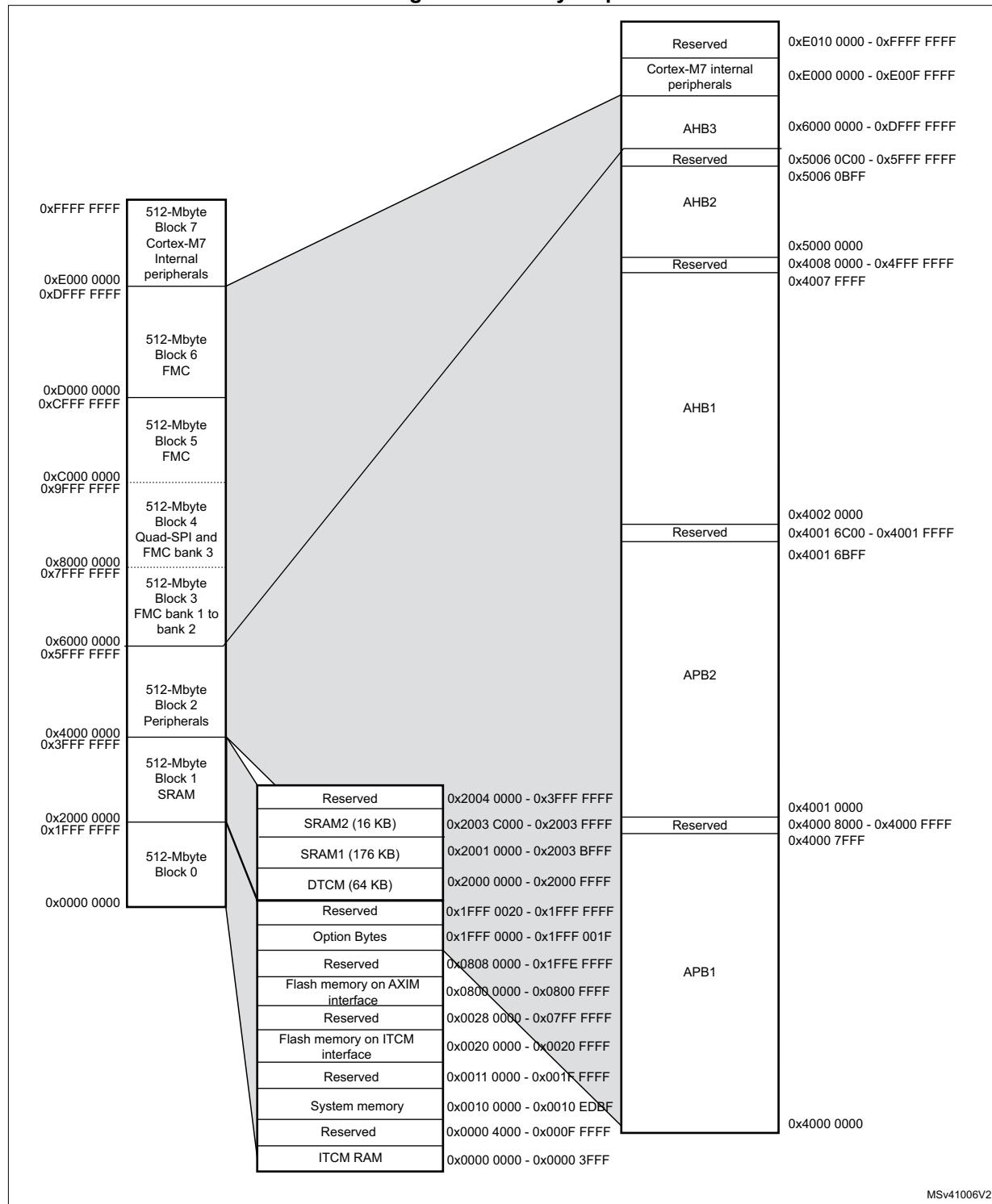
Program memory, data memory, registers and I/O ports are organized within the same linear 4-Gbyte address space.

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

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

1.5.2 Memory map and register boundary addresses

Figure 1. Memory map



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

The following table gives the boundary addresses of the peripherals available in the devices.

Table 1. STM32F72xxx and STM32F73xxx register boundary addresses

Boundary address	Peripheral	Bus	Register map
0xA000 1000 - 0xA0001FFF	QuadSPI Control Register	AHB3	Section 13.5.14: QUADSPI register map on page 375
0xA000 0000 - 0xA000 0FFF	FMC control register		Section 12.8: FMC register map on page 345
0x5006 0800 - 0x5006 0BFF	RNG	AHB2	Section 16.8.4: RNG register map on page 456
0x5006 0000 - 0x5006 03FF	AES		Section 17.7.18: AES register map on page 508
0x5000 0000 - 0x5003 FFFF	USB OTG FS		Section 32.15.61: OTG_FS/OTG_HS register map on page 1285
0x4004 0000 - 0x4007 FFFF	USB OTG HS	AHB1	Section 32.15.61: OTG_FS/OTG_HS register map on page 1285
0x4002 6400 - 0x4002 67FF	DMA2		Section 8.5.11: DMA register map on page 248
0x4002 6000 - 0x4002 63FF	DMA1		Section 5.3.27: RCC register map on page 190
0x4002 4000 - 0x4002 4FFF	BKPSRAM		Section 3.7.9: Flash interface register map
0x4002 3C00 - 0x4002 3FFF	Flash interface register		Section 5.3.27: RCC register map on page 190
0x4002 3800 - 0x4002 3BFF	RCC		Section 11.4.6: CRC register map on page 269
0x4002 3000 - 0x4002 33FF	CRC		
0x4002 2000 - 0x4002 23FF	GPIOI		
0x4002 1C00 - 0x4002 1FFF	GPIOH		
0x4002 1800 - 0x4002 1BFF	GPIOG		
0x4002 1400 - 0x4002 17FF	GPIOF		
0x4002 1000 - 0x4002 13FF	GPIOE		Section 6.4.11: GPIO register map on page 208
0x4002 0C00 - 0x4002 0FFF	GPIOD		
0x4002 0800 - 0x4002 0BFF	GPIOC		
0x4002 0400 - 0x4002 07FF	GPIOB		
0x4002 0000 - 0x4002 03FF	GPIOA		

Table 1. STM32F72xxx and STM32F73xxx register boundary addresses (continued)

Boundary address	Peripheral	Bus	Register map
0x4001 7C00 - 0x4001 7FFF	USBPHYC	APB2	Section 33.4.4: USBPHYC register map
0x4001 5C00 - 0x4001 5FFF	SAI2		Section 29.5.18: SAI register map on page 1063
0x4001 5800 - 0x4001 5BFF	SAI1		Section 29.5.18: SAI register map on page 1063
0x4001 5000 - 0x4001 53FF	SPI5		Section 28.9.10: SPI/I2S register map on page 1009
0x4001 4800 - 0x4001 4BFF	TIM11		Section 20.5.12: TIM10/TIM11/TIM13/TIM14 register map on page 722
0x4001 4400 - 0x4001 47FF	TIM10		Section 20.4.13: TIM9/TIM12 register map on page 712
0x4001 4000 - 0x4001 43FF	TIM9		Section 10.9.7: EXTI register map on page 263
0x4001 3C00 - 0x4001 3FFF	EXTI		Section 7.2.8: SYSCFG register maps on page 216
0x4001 3800 - 0x4001 3BFF	SYSCFG		Section 28.9.10: SPI/I2S register map on page 1009
0x4001 3400 - 0x4001 37FF	SPI4		Section 28.9.10: SPI/I2S register map on page 1009
0x4001 3000 - 0x4001 33FF	SPI1		Section 28.9.10: SPI/I2S register map on page 1009
0x4001 2C00 - 0x4001 2FFF	SDMMC1		Section 30.8.16: SDMMC register map on page 1121
0x4001 2000 - 0x4001 23FF	ADC1 - ADC2 - ADC3		Section 14.13.18: ADC register map on page 420
0x4001 1C00 - 0x4001 1FFF	SDMMC2		Section 30.8.16: SDMMC register map on page 1121
0x4001 1400 - 0x4001 17FF	USART6		Section 27.8.12: USART register map on page 949
0x4001 1000 - 0x4001 13FF	USART1		
0x4001 0400 - 0x4001 07FF	TIM8		
0x4001 0000 - 0x4001 03FF	TIM1		Section 18.4.24: TIM1 register map on page 601

Table 1. STM32F72xxx and STM32F73xxx register boundary addresses (continued)

Boundary address	Peripheral	Bus	Register map
0x4000 7C00 - 0x4000 7FFF	UART8	APB1	Section 27.8.12: USART register map on page 949
0x4000 7800 - 0x4000 7BFF	UART7		Section 15.5.15: DAC register map on page 444
0x4000 7400 - 0x4000 77FF	DAC		Section 4.4.4: PWR power control register 2 (PWR_CSR2) on page 129
0x4000 7000 - 0x4000 73FF	PWR		Section 31.9.5: bxCAN register map on page 1163
0x4000 6400 - 0x4000 67FF	CAN1		
0x4000 5C00 - 0x4000 5FFF	I2C3		
0x4000 5800 - 0x4000 5BFF	I2C2		Section 26.7.12: I2C register map on page 884
0x4000 5400 - 0x4000 57FF	I2C1		
0x4000 5000 - 0x4000 53FF	UART5		
0x4000 4C00 - 0x4000 4FFF	UART4		Section 27.8.12: USART register map on page 949
0x4000 4800 - 0x4000 4BFF	USART3		
0x4000 4400 - 0x4000 47FF	USART2		
0x4000 3C00 - 0x4000 3FFF	SPI3 / I2S3		Section 28.9.10: SPI/I2S register map on page 1009
0x4000 3800 - 0x4000 3BFF	SPI2 / I2S2		
0x4000 3000 - 0x4000 33FF	IWDG		Section 23.4.6: IWDG register map on page 767
0x4000 2C00 - 0x4000 2FFF	WWDG		Section 24.4.4: WWDG register map on page 774
0x4000 2800 - 0x4000 2BFF	RTC & BKP Registers		Section 25.6.21: RTC register map on page 818
0x4000 2400 - 0x4000 27FF	LPTIM1		Section 22.7.9: LPTIM register map on page 758
0x4000 2000 - 0x4000 23FF	TIM14		Section 20.5.12: TIM10/TIM11/TIM13/TIM14 register map on page 722
0x4000 1C00 - 0x4000 1FFF	TIM13		
0x4000 1800 - 0x4000 1BFF	TIM12		Section 20.4.13: TIM9/TIM12 register map on page 712
0x4000 1400 - 0x4000 17FF	TIM7		Section 21.4.9: TIM6/TIM7 register map on page 736
0x4000 1000 - 0x4000 13FF	TIM6		
0x4000 0C00 - 0x4000 0FFF	TIM5		
0x4000 0800 - 0x4000 0BFF	TIM4		
0x4000 0400 - 0x4000 07FF	TIM3		
0x4000 0000 - 0x4000 03FF	TIM2		Section 19.4.21: TIMx register map on page 672

1.6 Embedded SRAM

The STM32F72xxx and STM32F73xxx feature:

- System SRAM up to 256 Kbytes including Data TCM RAM 64 Kbytes
- Instruction RAM (ITCM-RAM) 16 Kbytes.
- 4 Kbytes of backup SRAM (see section 5.1.2: Battery backup domain)

The embedded SRAM is divided into up to four blocks:

- System SRAM:
 - SRAM1 mapped at address 0x2001 0000 and accessible by all AHB masters from AHB bus Matrix.
 - SRAM2 mapped at address 0x2003 C000 and accessible by all AHB masters from AHB bus Matrix.
 - DTCM-RAM on TCM interface (Tightly Coupled Memory interface) mapped at address 0x2000 0000 and accessible by all AHB masters from AHB bus Matrix but through a specific AHB slave bus of the CPU.
- Instruction SRAM:
 - Instruction RAM (ITCM-RAM) mapped at address 0x0000 0000 and accessible only by CPU.

The SRAM1 and SRAM2 can be accessed as bytes, half-words (16 bits) or full words (32 bits). While DTCM and ITCM RAMs can be accessed as bytes, half-words (16 bits), full words (32 bits) or double words (64 bits). These memories can be addressed at maximum system clock frequency without wait state.

The AHB masters support concurrent SRAM accesses (from the USB OTG HS): for instance, the USB OTG HS can read/write from/to SRAM2 while the CPU is reading/writing from/to SRAM1.

1.7 Flash memory overview

The Flash memory interface manages CPU AXI and TCM accesses to the Flash memory. It implements the erase and program Flash memory operations and the read and write protection mechanisms. It accelerates code execution with ART on TCM interface or L1-Cache on AXIM interface.

The Flash memory is organized as follows:

- A main memory block divided into sectors.
- An information block:
 - System memory from which the device boots in System memory boot mode
 - 528 OTP (one-time programmable) bytes for user data.
 - Option bytes to configure read and write protection, BOR level, software/hardware watchdog, boot memory base address and reset when the device is in Standby or Stop mode.

Refer to [Section 3: Embedded Flash memory \(FLASH\)](#) for more details.

1.8 Boot configuration

In the STM32F72xxx and STM32F73xxx, two different boot areas can be selected through the BOOT pin and the boot base address programmed in the BOOT_ADD0 and BOOT_ADD1 option bytes as shown in the [Table 2](#).

Table 2. Boot modes

Boot mode selection		Boot area
BOOT	Boot address option bytes	
0	BOOT_ADD0[15:0]	Boot address defined by user option byte BOOT_ADD0[15:0] ST programmed value: Flash on ITCM at 0x0020 0000
1	BOOT_ADD1[15:0]	Boot address defined by user option byte BOOT_ADD1[15:0] ST programmed value: System bootloader at 0x0010 0000

The values on the BOOT pin are latched on the 4th rising edge of SYSCLK after reset release. It is up to the user to set the BOOT pin after reset.

The BOOT pin is also resampled when the device exits the Standby mode. Consequently, they must be kept in the required Boot mode configuration when the device is in the Standby mode.

After startup delay, the selection of the boot area is done before releasing the processor reset.

The BOOT_ADD0 and BOOT_ADD1 address option bytes allows to program any boot memory address from 0x0000 0000 to 0x3FFF FFFF which includes:

- All Flash address space mapped on ITCM or AXIM interface
- All RAM address space: ITCM, DTCM RAMs and SRAMs mapped on AXIM interface
- The System memory bootloader

The BOOT_ADD0 / BOOT_ADD1 option bytes can be modified after reset in order to boot from any other boot address after next reset.

If the programmed boot memory address is out of the memory mapped area or a reserved area, the default boot fetch address is programmed as follows:

- Boot address 0: ITCM-FLASH at 0x0020 0000
- Boot address 1: ITCM-RAM at 0x0000 0000

When flash level 2 protection is enabled, only boot from Flash (on ITCM or AXIM interface) or system bootloader will be available. If the already programmed boot address in the BOOT_ADD0 and/or BOOT_ADD1 option bytes is out of the memory range or RAM address (on ITCM or AXIM) the default fetch will be forced from Flash on ITCM interface at address 0x00200000.

Embedded bootloader

The embedded bootloader code is located in the system memory. It is programmed by ST during production. For full information, refer to the application note (AN2606) STM32 microcontroller system memory boot mode.

By default, when the boot from system bootloader is selected, the code is executed from TCM interface. It could be executed from AXIM interface by reprogramming the BOOT_ADDx address option bytes to 0x1FF0 0000.

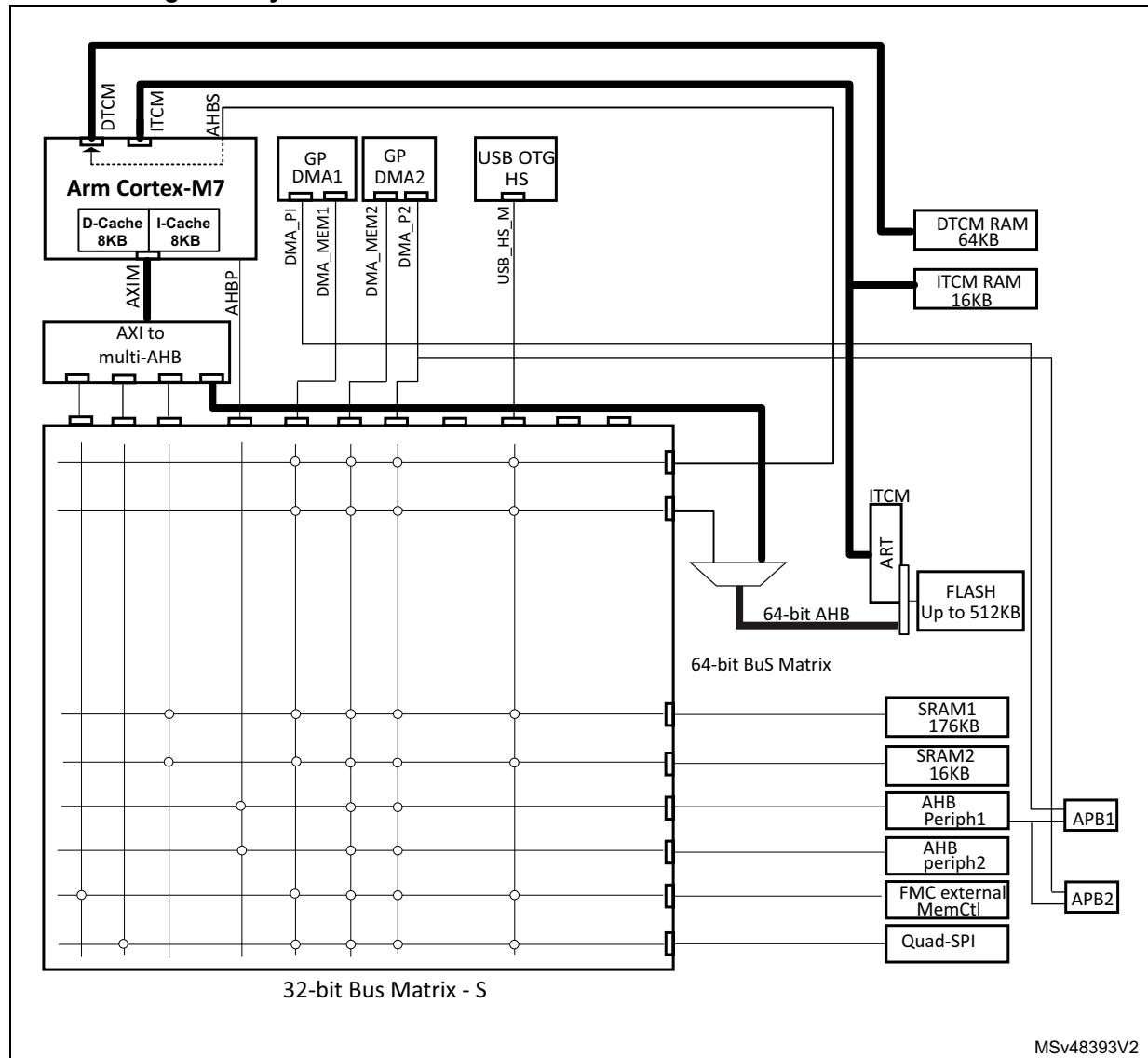
2 System and memory overview

2.1 System architecture

The main system architecture is based on 2 sub-systems:

- An AXI to multi AHB bridge converting AXI4 protocol to AHB-Lite protocol:
 - 1x AXI to 64-bit AHB bridge connected to the embedded flash
 - 3x AXI to 32bit AHB bridge connected to AHB bus matrix
- A multi-AHB Bus-Matrix

Figure 2. System architecture for STM32F72xxx and STM32F73xxx devices



The multi-AHB Bus-Matrix interconnects all the masters and slaves and it consists on:

- 32-bit multi-AHB Bus-Matrix
- 64-bit multi-AHB Bus-Matrix: It interconnects the 64-bit AHB bus from CPU through the AXI to AHB bridge and the 32-bit AHB bus from GP DMAs and peripheral DMAs upsized to 64-bit to the internal flash.

The multi AHB bus matrix interconnects:

- 9 bus masters:
 - 3x32-bit AHB bus Cortex®-M7 AXI Master bus 64-bits, splitted 4 masters through the AXI to AHB bridge.
 - 1x64-bit AHB bus connected to the embedded flash
 - Cortex® -M7 AHB Peripherals bus
 - DMA1 memory bus
 - DMA2 memory bus
 - DMA2 peripheral bus
 - USB OTG HS DMA bus
- Eight bus slaves:
 - the embedded Flash on AHB bus (for Flash read/write access, for code execution and data access)
 - Cortex®-M7 AHBS slave interface for DMAs data transfer on DTCM RAM only.
 - Main internal SRAM1 (176 Kbytes)
 - Auxiliary internal SRAM2 (16 Kbytes)
 - AHB1peripherals including AHB to APB bridges and APB peripherals
 - AHB2 peripherals including AHB to APB bridges and APB peripherals
 - FMC
 - Quad SPI

2.1.1 Multi AHB BusMatrix

The multi AHB BusMatrix manages the access arbitration between masters. The arbitration uses a round-robin algorithm.

It provides access from a master to a slave, enabling concurrent access and efficient operation even when several high-speed peripherals work simultaneously.

The DTCM and ITCM RAMs (tightly coupled memories) are not part of the bus matrix.

The Data TCM RAM is accessible by the GP-DMAs and peripherals DMAs through specific AHB slave bus of the CPU.

The instruction TCM RAM is reserved only for CPU. it is accessed at CPU clock speed with 0 wait states. The architecture is shown in [Figure 2](#).

2.1.2 AHB/APB bridges (APB)

The two AHB/APB bridges, APB1 and APB2, provide full synchronous connections between the AHB and the two APB buses, allowing flexible selection of the peripheral frequency.

Refer to the device datasheets for more details on APB1 and APB2 maximum frequencies, and to [Table 1](#) for the address mapping of AHB and APB peripherals.

After each device reset, all peripheral clocks are disabled (except for the SRAM, DTCM, ITCM RAM and Flash memory interface). Before using a peripheral the user has to enable its clock in the RCC_AHBxENR or RCC_APBxENR register.

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

2.1.3 CPU AXIM bus

This bus connects the Instruction and data bus of the Cortex®-M7 with FPU core to the multi-AHB Bus-Matrix through AXI to AHB bridge. There are 4 AXI bus accesses:

- CPU AXI bus access 1: The target of this AXI bus is the external memory FMC containing code or data. For the NAND Bank mapped at address 0x8000 0000 to 0x8FFF FFFF, the MPU memory attribute for this space must be reconfigured by software to Device.
- CPU AXI bus access 2: The target of this AXI bus is the external memory Quad SPI containing code or data.
- CPU AXI bus access 3: The target of this AXI bus is the internal SRAMs (SRAM1 and SRAM2) containing code or data.
- CPU AXI bus access 4: The target of this AXI bus is the embedded Flash mapped on AXI interface containing code or data.

2.1.4 ITCM bus

This bus is used by the Cortex®-M7 for instruction fetches and data access on the embedded flash mapped on ITCM interface and instruction fetches only on ITCM RAM.

2.1.5 DTCM bus

This bus is used by the Cortex®-M7 for data access on the DTCM RAM. It can be also used for instruction fetches.

2.1.6 CPU AHBS bus

This bus connects the AHB Slave bus of the Cortex®-M7 to the BusMatrix. This bus is used by DMAs and Peripherals DMAs for Data transfer on DTCM RAM only.

The ITCM bus is not accessible on AHBS. So the DMA data transfer to/from ITCM RAM is not supported. For DMA transfer to/from Flash on ITCM interface, all the transfers are forced through AHB bus

2.1.7 AHB peripheral bus

This bus connects the AHB Peripheral bus of the Cortex®-M7 to the BusMatrix. This bus is used by the core to perform all data accesses to peripherals.

The target of this bus is the AHB1 peripherals including the APB peripherals and the AHB2 peripherals.

2.1.8 DMA memory bus

This bus connects the DMA memory bus master interface to the BusMatrix. It is used by the DMA to perform transfer to/from memories. The targets of this bus are data memories:

internal SRAM1, SRAM2 and DTCM (through the AHBS bus of Cortex®-M7) internal Flash memory and external memories through the FMC or Quad SPI.

2.1.9 DMA peripheral bus

This bus connects the DMA peripheral master bus interface to the BusMatrix. This bus is used by the DMA to access AHB peripherals or to perform memory-to-memory transfers. The targets of this bus are the AHB and APB peripherals plus data memories: internal SRAM1, SRAM2 and DTCM (through the AHBS bus of Cortex®-M7) internal Flash memory and external memories through the FMC or Quad SPI.

2.1.10 USB OTG HS DMA bus

This bus connects the USB OTG HS DMA master interface to the BusMatrix. This bus is used by the USB OTG DMA to load/store data to a memory. The targets of this bus are data memories: internal SRAM1, SRAM2 and DTCM (through the AHBS bus of Cortex®-M7), internal Flash memory, and external memories through the FMC or Quad SPI.

3 Embedded Flash memory (FLASH)

3.1 Introduction

The Flash memory interface manages Cortex®-M7 AXI and TCM accesses to the Flash memory. It implements the erase and program Flash memory operations and the read and write protection mechanisms.

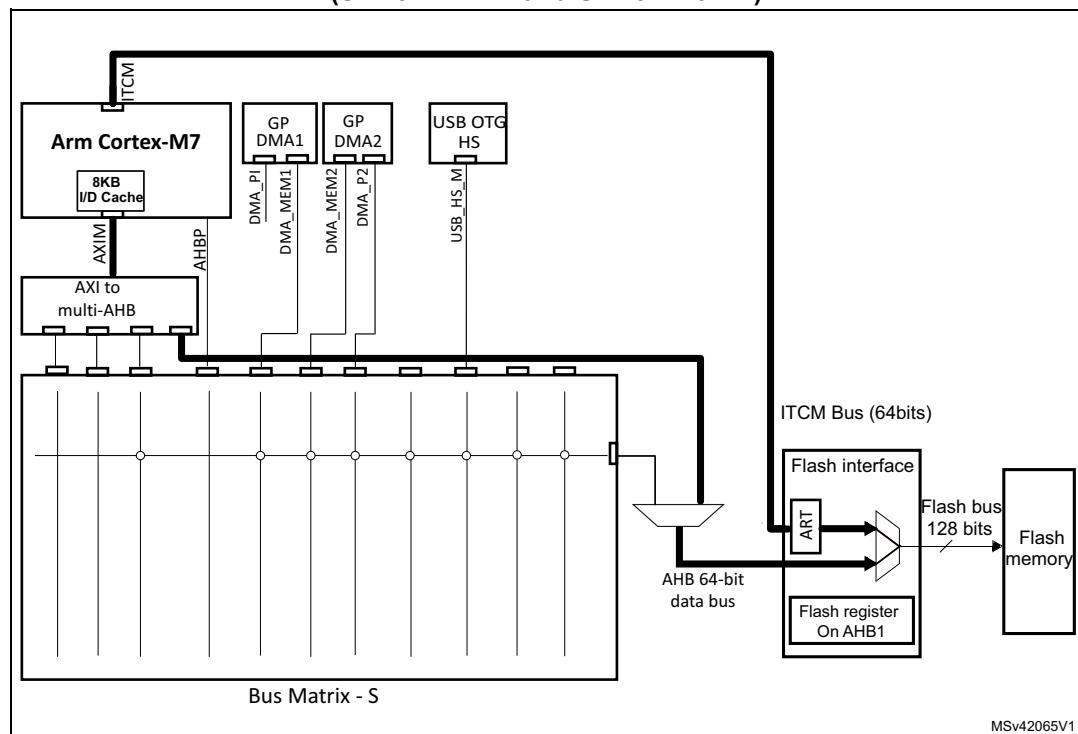
The Flash memory interface accelerates code execution with a system of instruction prefetch and cache lines on ITCM interface (ART Accelerator™).

3.2 Flash main features

- Flash memory read operations
- Flash memory program/erase operations
- Read / write protections
- 64 cache lines of 128 bits on ITCM interface (ART Accelerator™)
- Prefetch on TCM instruction code
- Security features (PCROP)

Figure 3 shows the Flash memory interface connection inside the system architecture.

Figure 3. Flash memory interface connection inside system architecture (STM32F72xxx and STM32F73xxx)



3.3 Flash functional description

3.3.1 Flash memory organization

The Flash memory has the following main features:

- Capacity up to 512 Kbytes on STM32F72xxx and STM32F732xx/F733xx devices and 64 Kbytes on STM32F730xx devices
- 128 bits wide data read
- Byte, half-word, word and double word write
- Sector and mass erase

The Flash memory is organized as follows:

- On STM32F72xxx and STM32F732xx/F733xx devices, a main memory block divided into 4 sectors of 16 Kbytes, 1 sector of 64 Kbytes, and 3 sectors of 128 Kbytes
- On STM32F730xx devices, a main memory block divided into 4 sectors of 16 Kbytes.
- Information blocks containing:
 - System memory from which the device boots in System memory boot mode
 - 528 bytes OTP (one-time programmable) for user data
 - The OTP area contains 16 additional bytes used to lock the corresponding OTP data block.
 - Option bytes to configure read and write protection, BOR level, software/hardware watchdog, boot memory base address and reset when the device is in Standby or Stop mode.

The embedded flash has three main interfaces:

- 64-bits ITCM interface:
 - It is connected to the ITCM bus of Cortex-M7 and used for instruction execution and data read access.
 - Write accesses are not supported on ITCM interface
 - Supports a unified 64 cache lines of 128 bits (ART accelerator)
- 64-bits AHB interface:
 - It is connected to the AXI bus of Cortex-M7 through the AHB bus matrix and used for code execution, read and write accesses.
 - DMAs and peripherals DMA data transfer on Flash are done through the AHB interface whatever the addressed flash interface TCM or AHB.
- 32-bits AHB register interface:
 - It is used for control and status register accesses.

The main memory and information block organization are shown in [Table 3](#).

Table 3. STM32F72xxx and STM32F732xx/F733xx Flash memory organization

Block	Name	Bloc base address on AXIM interface	Block base address on ICTM interface	Sector size
Main memory block	Sector 0	0x0800 0000 - 0x0800 3FFF	0x0020 0000 - 0x0020 3FFF	16 Kbytes
	Sector 1	0x0800 4000 - 0x0800 7FFF	0x0020 4000 - 0x0020 7FFF	16 Kbytes
	Sector 2	0x0800 8000 - 0x0800 BFFF	0x0020 8000 - 0x0020 BFFF	16 Kbytes
	Sector 3	0x0800 C000 - 0x0800 FFFF	0x0020 C000 - 0x0020 FFFF	16 Kbytes
	Sector 4	0x0801 0000 - 0x0801 FFFF	0x0021 0000 - 0x0021 FFFF	64 Kbytes
	Sector 5	0x0802 0000 - 0x0803 FFFF	0x0022 0000 - 0x0023 FFFF	128 Kbytes
	Sector 6	0x0804 0000 - 0x0805 FFFF	0x0024 0000 - 0x0025 FFFF	128 Kbytes
	Sector 7	0x0806 0000 - 0x0807 FFFF	0x0026 0000 - 0x027 FFFF	128 Kbytes
Information block	System memory	0x1FF0 0000 - 0x1FF0 76D7	0x0010 0000 - 0x0010 76D7	~ 30 Kbytes
	OTP	0x1FF0 7800 - 0x1FF0 7A0F	0x0010 7800 - 0x0010 7A0F	528 bytes
	Option bytes	0x1FFF 0000 - 0x1FFF 001F	-	32 bytes

Table 4. STM32F730xx Flash memory organization

Block	Name	Bloc base address on AXIM interface	Block base address on ICTM interface	Sector size
Main memory block	Sector 0	0x0800 0000 - 0x0800 3FFF	0x0020 0000 - 0x0020 3FFF	16 Kbytes
	Sector 1	0x0800 4000 - 0x0800 7FFF	0x0020 4000 - 0x0020 7FFF	16 Kbytes
	Sector 2	0x0800 8000 - 0x0800 BFFF	0x0020 8000 - 0x0020 BFFF	16 Kbytes
	Sector 3	0x0800 C000 - 0x0800 FFFF	0x0020 C000 - 0x0020 FFFF	16 Kbytes
Information block	System memory	0x1FF0 0000 - 0x1FF0 76D7	0x0010 0000 - 0x0010 76D7	~ 30 Kbytes
	OTP	0x1FF0 7800 - 0x1FF0 7A0F	0x0010 7800 - 0x0010 7A0F	528 bytes
	Option bytes	0x1FFF 0000 - 0x1FFF 001F	-	32 bytes

3.3.2 Read access latency

To correctly read data from Flash memory, the number of wait states (LATENCY) must be correctly programmed in the Flash access control register (FLASH_ACR) according to the frequency of the CPU clock (HCLK) and the supply voltage of the device.

The correspondence between wait states and CPU clock frequency is given in [Table 14](#) and [Table 5](#).

- Note:**
- when $VOS[1:0] = '0x01'$, the maximum value of f_{HCLK} is 144 MHz.
 - when $VOS[1:0] = '0x10'$, the maximum value of f_{HCLK} is 168 MHz. It can be extended to 180 MHz by activating the over-drive mode.
 - when $VOS[1:0] = '0x11'$, the maximum value of f_{HCLK} is 180 MHz. It can be extended to 216 MHz by activating the over-drive mode.
 - The over-drive mode is not available when V_{DD} ranges from 1.8 to 2.1 V.

Refer to [Section 4.1.4: Voltage regulator](#) for details on how to activate the over-drive mode.

Table 5. Number of wait states according to CPU clock (HCLK) frequency

Wait states (WS) (LATENCY)	HCLK (MHz)			
	Voltage range 2.7 V - 3.6 V	Voltage range 2.4 V - 2.7 V	Voltage range 2.1 V - 2.4 V	Voltage range 1.8 V - 2.1 V
0 WS (1 CPU cycle)	$0 < HCLK \leq 30$	$0 < HCLK \leq 24$	$0 < HCLK \leq 22$	$0 < HCLK \leq 20$
1 WS (2 CPU cycles)	$30 < HCLK \leq 60$	$24 < HCLK \leq 48$	$22 < HCLK \leq 44$	$20 < HCLK \leq 40$
2 WS (3 CPU cycles)	$60 < HCLK \leq 90$	$48 < HCLK \leq 72$	$44 < HCLK \leq 66$	$40 < HCLK \leq 60$
3 WS (4 CPU cycles)	$90 < HCLK \leq 120$	$72 < HCLK \leq 96$	$66 < HCLK \leq 88$	$60 < HCLK \leq 80$
4 WS (5 CPU cycles)	$120 < HCLK \leq 150$	$96 < HCLK \leq 120$	$88 < HCLK \leq 110$	$80 < HCLK \leq 100$
5 WS (6 CPU cycles)	$150 < HCLK \leq 180$	$120 < HCLK \leq 144$	$110 < HCLK \leq 132$	$100 < HCLK \leq 120$
6 WS (7 CPU cycles)	$180 < HCLK \leq 210$	$144 < HCLK \leq 168$	$132 < HCLK \leq 154$	$120 < HCLK \leq 140$
7 WS (8 CPU cycles)	$210 < HCLK \leq 216$	$168 < HCLK \leq 192$	$154 < HCLK \leq 176$	$140 < HCLK \leq 160$
8 WS (9 CPU cycles)	-	$192 < HCLK \leq 216$	$176 < HCLK \leq 198$	$160 < HCLK \leq 180$
9 WS (10 CPU cycles)	-	-	$198 < HCLK \leq 216$	-

After reset, the CPU clock frequency is 16 MHz and 0 wait state (WS) is configured in the FLASH_ACR register.

It is highly recommended to use the following software sequences to tune the number of wait states to access the Flash memory with the CPU frequency.

Increasing the CPU frequency

1. Program the new number of wait states to the LATENCY bits in the FLASH_ACR register
2. Check that the new number of wait states is taken into account to access the Flash memory by reading the FLASH_ACR register
3. Modify the CPU clock source by writing the SW bits in the RCC_CFGR register
4. If needed, modify the CPU clock prescaler by writing the HPRE bits in RCC_CFGR
5. Check that the new CPU clock source or/and the new CPU clock prescaler value is/are taken into account by reading the clock source status (SWS bits) or/and the AHB prescaler value (HPRE bits), respectively, in the RCC_CFGR register.

Decreasing the CPU frequency

1. Modify the CPU clock source by writing the SW bits in the RCC_CFGR register
2. If needed, modify the CPU clock prescaler by writing the HPRE bits in RCC_CFGR
3. Check that the new CPU clock source or/and the new CPU clock prescaler value is/are taken into account by reading the clock source status (SWS bits) or/and the AHB prescaler value (HPRE bits), respectively, in the RCC_CFGR register
4. Program the new number of wait states to the LATENCY bits in FLASH_ACR
5. Check that the new number of wait states is used to access the Flash memory by reading the FLASH_ACR register

Note:

A change in CPU clock configuration or wait state (WS) configuration may not be effective straight away. To make sure that the current CPU clock frequency is the one the user has configured, the user can check the AHB prescaler factor and clock source status values. To make sure that the number of WS programmed is effective, the user can read the FLASH_ACR register.

Instruction prefetch

Each flash read operation provides 256 bits representing 8 instructions of 32 bits to 16 instructions of 16 bits according to the program launched. So, in case of sequential code, at least 8 CPU cycles are needed to execute the previous instruction line read. The prefetch on ITCM bus allows to read the sequential next line of instructions in the flash while the current instruction line is requested by the CPU. The prefetch can be enabled by setting the PRFTEN bit of the FLASH_ACR register. This feature is useful if at least one Wait State is needed to access the flash. When the code is not sequential (branch), the instruction may not be present neither in the current instruction line used nor in the prefetched instruction line. In this case (miss), the penalty in term of number of cycles is at least equal to the number of Wait States.

Adaptive real-time memory accelerator (ART Accelerator™)

The proprietary Adaptive real-time (ART) memory accelerator is optimized for STM32 industry-standard Arm® Cortex®-M7 with FPU processors. It balances the inherent performance advantage of the Arm® Cortex®-M7 with FPU over Flash memory technologies, which normally requires the processor to wait for the Flash memory at higher operating frequencies.

To release the processor full performance, the accelerator implements a unified cache of an instruction and branch cache which increases program execution speed from the 256-bit Flash memory. Based on CoreMark benchmark, the performance achieved thanks to the ART accelerator is equivalent to 0 wait state program execution from Flash memory at a CPU frequency up to 216 MHz.

The ART accelerator is available only for flash access on ITCM interface.

To limit the time lost due to jumps, it is possible to retain 64 lines of 256 bits in the ART accelerator. This feature can be enabled by setting the ARTEN bit of the FLASH_CR register. The ART Accelerator is unified, it contains instruction as well as data literal pools. Each time a miss occurs (requested data not present in the current data line used or in the instruction cache memory), the read line is copied in the instruction cache memory of ART. If a data contained in the instruction cache memory is requested by the CPU, the data is provided without inserting delay. Once all the cache memory lines are filled, the LRU (Least Recently Used) policy is used to determine the line to replace in the memory cache. This feature is particularly useful in case of code containing loops.

Note: Data in user configuration sector are not cacheable.

3.3.3 Flash program and erase operations

For any Flash memory program operation (erase or program), the CPU clock frequency (HCLK) must be at least 1 MHz. The contents of the Flash memory are not guaranteed if a device reset occurs during a Flash memory operation.

Any attempt to read the Flash memory while it is being written or erased, causes the bus to stall. Read operations are processed correctly once the program operation has completed. This means that code or data fetches cannot be performed while a write/erase operation is ongoing.

3.3.4 Unlocking the Flash control register

After reset, write is not allowed in the Flash control register (FLASH_CR) to protect the Flash memory against possible unwanted operations due, for example, to electric disturbances. The following sequence is used to unlock this register:

1. Write KEY1 = 0x45670123 in the Flash key register (FLASH_KEYR)
2. Write KEY2 = 0xCDEF89AB in the Flash key register (FLASH_KEYR)

Any wrong sequence will return a bus error and lock up the FLASH_CR register until the next reset.

The FLASH_CR register can be locked again by software by setting the LOCK bit in the FLASH_CR register.

Note: The FLASH_CR register is not accessible in write mode when the BSY bit in the FLASH_SR register is set. Any attempt to write to it with the BSY bit set will cause the AHB bus to stall until the BSY bit is cleared.

3.3.5 Program/erase parallelism

The Parallelism size is configured through the PSIZE field in the FLASH_CR register. It represents the number of bytes to be programmed each time a write operation occurs to the Flash memory. PSIZE is limited by the supply voltage and by whether the external V_{PP} supply is used or not. It must therefore be correctly configured in the FLASH_CR register before any programming/erasing operation.

A Flash memory erase operation can only be performed by sector, bank or for the whole Flash memory (mass erase). The erase time depends on PSIZE programmed value. For more details on the erase time, refer to the electrical characteristics section of the device datasheet.

Table 6 provides the correct PSIZE values.

Table 6. Program/erase parallelism

	Voltage range 2.7 - 3.6 V with External V _{PP}	Voltage range 2.7 - 3.6 V	Voltage range 2.4 - 2.7 V	Voltage range 2.1 - 2.4 V	Voltage range 1.8 V - 2.1 V
Parallelism size	x64	x32	x16	x8	
PSIZE(1:0)	11	10	01	00	

Note: Any program or erase operation started with inconsistent program parallelism/voltage range settings may lead to unpredicted results. Even if a subsequent read operation indicates that the logical value was effectively written to the memory, this value may not be retained.

To use V_{PP} an external high-voltage supply (between 8 and 9 V) must be applied to the V_{PP} pad. The external supply must be able to sustain this voltage range even if the DC consumption exceeds 10 mA. It is advised to limit the use of V_{PP} to initial programming on the factory line. The V_{PP} supply must not be applied for more than an hour, otherwise the Flash memory might be damaged.

3.3.6 Flash erase sequences

The Flash memory erase operation can be performed at sector level or on the whole Flash memory (Mass Erase). Mass Erase does not affect the OTP sector or the configuration sector.

Sector Erase

To erase a sector, follow the procedure below:

1. Check that no Flash memory operation is ongoing by checking the BSY bit in the FLASH_SR register
2. Set the SER bit and select the sector (out of the 8 in the main memory block) to erase (SNB) in the FLASH_CR register
3. Set the STRT bit in the FLASH_CR register
4. Wait for the BSY bit to be cleared

Mass Erase

To perform Mass Erase, the following sequence is recommended:

1. Check that no Flash memory operation is ongoing by checking the BSY bit in the FLASH_SR register
2. Set the MER bit in the FLASH_CR register
3. Set the STRT bit in the FLASH_CR register
4. Wait for the BSY bit to be cleared

Note: If MERx and SER bits are both set in the FLASH_CR register, mass erase is performed.

If both MERx and SER bits are reset and the STRT bit is set, an unpredictable behavior may occur without generating any error flag. This condition should be forbidden.

Note: When setting the STRT bit in the FLASH_CR register and before polling the BSY bit to be cleared, the software can issue a DSB instruction to guarantee the completion of a previous access to FLASH_CR register.

3.3.7 Flash programming sequences

Standard programming

The Flash memory programming sequence is as follows:

1. Check that no main Flash memory operation is ongoing by checking the BSY bit in the FLASH_SR register.
2. Set the PG bit in the FLASH_CR register
3. Perform the data write operation(s) to the desired memory address (inside main memory block or OTP area):
 - Byte access in case of x8 parallelism
 - Half-word access in case of x16 parallelism
 - Word access in case of x32 parallelism
 - Double word access in case of x64 parallelism
4. Wait for the BSY bit to be cleared.

Note: Successive write operations are possible without the need of an erase operation when changing bits from '1' to '0'. Writing '1' requires a Flash memory erase operation.

If an erase and a program operation are requested simultaneously, the erase operation is performed first.

Note: After performing a data write operation and before polling the BSY bit to be cleared, the software can issue a DSB instruction to guarantee the completion of a previous data write operation.

Programming errors

In case of error, the Flash operation (programming or erasing) is aborted with one of the following errors:

- **PGAERR:** Alignment Programming error
It is not allowed to program data to the Flash memory that would cross the 128-bit row boundary. In such a case, the write operation is not performed and the program alignment error flag (PGAERR) is set in the FLASH_SR register.
- **PGEPRR:** Programming parallelism error
The write access type (byte, half-word, word or double word) must correspond to the type of parallelism chosen (x8, x16, x32 or x64). If not, the write operation is not performed and the program parallelism error flag (PGEPRR) is set in the FLASH_SR register.
- **ERSERR:** Erase sequence error
When an erase operation to the flash is performed by the code while the control register has not been correctly configured, the ERSERR error flag is set
- **WRPERR:** Write Protection Error
WRPERR is set if one of the following conditions occurs:
 - Attempt to program or erase in a write protected area (WRP)
 - Attempt to program or erase the system memory area.
 - A write in the OTP area which is already locked
 - Attempt to modify the option bytes when the read protection (RDP) is set to Level
 - The Flash memory is read protected and an intrusion is detected

If an erase operation in Flash memory also concerns data in the ART accelerator, the user has to make sure that these data are rewritten before they are accessed during code execution. If this cannot be done safely, it is recommended to flush and/or deactivate the ART accelerator by setting respectively the bits ARTRST or ARTEN of the FLASH_CR register.

Note: The ART cache can be flushed only if the ART accelerator is disabled (ARTEN = 0).

3.3.8 Flash Interrupts

Setting the end of operation interrupt enable bit (EOPIE) in the FLASH_CR register enables interrupt generation when an erase or program operation ends, that is when the busy bit (BSY) in the FLASH_SR register is cleared (operation completed, correctly or not). In this case, the end of operation (EOP) bit in the FLASH_SR register is set.

If an error occurs during a program, an erase, or a read operation request, one of the following error flags is set in the FLASH_SR register:

- PGAERR, PGPERR, ERSERR (Program error flags)
- WRPERR (Protection error flag)

In this case, if the error interrupt enable bit (ERRIE) is set in the FLASH_CR register, an interrupt is generated and the operation error bit (OPERR) is set in the FLASH_SR register.

Note: If several successive errors are detected (for example, in case of DMA transfer to the Flash memory), the error flags cannot be cleared until the end of the successive write requests.

Table 7. Flash interrupt request

Interrupt event	Event flag	Enable control bit
End of operation	EOP	EOPIE
Write protection error	WRPERR	ERRIE
Programming error	PGAERR, PGPERR, ERSERR	ERRIE

3.4 FLASH Option bytes

3.4.1 Option bytes description

The option bytes are configured by the end user depending on the application requirements. [Table 8](#) shows the organization of these bytes inside the information block.

The option bytes can be read from the user configuration memory locations or from the Option byte registers:

- [Flash option control register \(FLASH_OPTCR\)](#)
- [Flash option control register \(FLASH_OPTCR1\)](#)

Table 8. Option byte organization

AXI address	[63:16]	[15:0]
0x1FFF 0000	Reserved	ROP & user option bytes (RDP & USER)
0x1FFF 0008	Reserved	IWDG_STOP, IWDG_STBY and Write protection nWRP (sector 0 to 7) and user option bytes
0x1FFF 0010	Reserved	BOOT_ADD0
0x1FFF 0018	Reserved	BOOT_ADD1
0x1FFF 0020	Reserved	Reserved & PCPROP
0x1FFF 0028	Reserved	PCPROP_RDP & reserved

User and read protection option bytes

Memory address: 0x1FFF 0000

ST programmed value: 0x5500AAFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RDP								nRST_STDBY	nRST_STOP	IWDG_SW	WWDG_SW	BOR_LEV[1:0]	Res.	Res.	
r								r	r	r	r	r	r		

Bits 31:13 Not used.

Bit 15:8 **RDP**: Read Out Protection

The read protection helps the user protect the software code stored in Flash memory.

0xAA: Level0, no Protection

0xCC: Level2, chip protection (debug & boot in RAM features disabled)

others: Level1, read protection of memories (debug features limited)

Bit 7 **nRST_STDBY**

0: Reset generated when entering Standby mode.

1: No reset generated.

Bit 6 **nRST_STOP**

0: Reset generated when entering Stop mode.

1: No reset generated.

Bit 5 **IWDG_SW**: Independant watchdog selection

0: Hardware independant watchdog.

1: Software independant watchdog.

Bit 4 **WWDG_SW**: Window watchdog selection

0: Hardware window watchdog.

1: Software window watchdog.

Bits 3:2 **BOR_LEV**: BOR reset Level

These bits contain the supply level threshold that activates/releases the reset. They can be written to program a new BOR level value into Flash memory.

00: BOR Level 3 (VBOR3), brownout threshold level 3

01: BOR Level 2 (VBOR2), brownout threshold level 2

10: BOR Level 1 (VBOR1), brownout threshold level 1

11: BOR off, POR/PDR reset threshold level is applied

Note: For full details on BOR characteristics, refer to the “Electrical characteristics” section of the product datasheet.

Bits 1:0 Not used

User and write protection option bytes

Memory address: 0x1FFF 0008

ST programmed value: 0x0000FFFF

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

Bits 31:16 Not used.

Bit 15 **IWDG_STOP:** Independent watchdog counter freeze in stop mode

- 0: Freeze IWDG counter in stop mode.
- 1: IWDG counter active in stop mode.

Bit 14 **IWDG_STDBY:** Independent watchdog counter freeze in Standby mode

- 1: IWDG counter active in standby mode.

Bit 13:8 Not used

Bits 7:0 **nWRPi:** Non Write Protection of sector i

- 0: Write protection active on sector i.
- 1: Write protection not active on sector i.

Boot address option bytes when Boot pin =0

Memory address: 0x1FFF 0010

ST programmed value: 0xFF7F 0080 (ITCM-FLASH base address)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
BOOT_ADD0[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Not used.

Bit 15:0 **BOOT_ADD0[15:0]**: Boot memory base address when Boot pin =0

BOOT_ADD0[15:0] correspond to address [29:14],

The boot base address supports address range only from 0x0000 0000 to 0x2004 FFFF with a granularity of 16 Kbytes.

Example:

BOOT_ADD0 = 0x0000: Boot from ITCM RAM (0x0000 0000)

BOOT_ADD0 = 0x0040: Boot from system memory bootloader (0x0010 0000)

BOOT_ADD0 = 0x0080: Boot from Flash on ITCM interface (0x0020 0000)

BOOT_ADD0 = 0x2000: Boot from Flash on AXIM interface (0x0800 0000)

BOOT_ADD0 = 0x8000: Boot from DTCM RAM (0x2000 0000)

BOOT_ADD0 = 0x8004: Boot from SRAM1 (0x2001 0000)

BOOT_ADD0 = 0x8013: Boot from SRAM2 (0x2004 C000)

Boot address option bytes when Boot pin =1

Memory address: 0x1FFF 0018

ST programmed value: 0xFFBF0040 (system memory bootloader address)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
BOOT_ADD1[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 **Not used**

Bit 15:0 **BOOT_ADD1[15:0]**: Boot memory base address when Boot pin =1
 BOOT_ADD1[15:0] correspond to address [29:14],
 The boot base address supports address range only from 0x0000 0000 to 0x2004 FFFF
 with a granularity of 16KB.

Example:

BOOT_ADD1 = 0x0000: Boot from ITCM RAM(0x0000 0000)
 BOOT_ADD1 = 0x0040: Boot from system memory bootloader (0x0010 0000)
 BOOT_ADD1 = 0x0080: Boot from Flash on ITCM interface (0x0020 0000)
 BOOT_ADD1 = 0x2000: Boot from Flash on AXIM interface (0x0800 0000)
 BOOT_ADD1 = 0x8000: Boot from DTCM RAM (0x2000 0000)
 BOOT_ADD1 = 0x8004: Boot from SRAM1 (0x2001 0000)
 BOOT_ADD1 = 0x8013: Boot from SRAM2 (0x2003 C000)

PCPROP option bytes

Memory address: 0x1FFF 0020

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

Bits 31:8 Reserved.

Bits 7:0 **PCROPi**: PCROP option byte

- 0: PCROP protection not active on sector i; i = 0..7
- 1: PCROP protection active on sector i; i = 0..7

PCPROP RDP option bytes

Memory address: 0x1FFF 0028

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

Bits 31:16 Reserved.

Bit 15 **PCROP_RDP**: PCROP zone preserved when RDP level decreased.

- 0: PCROP zone is kept when RDP is decreased: partial mass erase is done.
- 1: PCROP zone is erased when RDP is decreased: full mass erase is done.

Bits 14:0 Reserved.

3.4.2 Option bytes programming

To run any operation on this sector, the option lock bit (OPTLOCK) in the Flash option control register (FLASH_OPTCR) must be cleared. To be allowed to clear this bit, the user has to perform the following sequence:

1. Write OPTKEY1 = 0x0819 2A3B in the Flash option key register (FLASH_OPTKEYR)
2. Write OPTKEY2 = 0x4C5D 6E7F in the Flash option key register (FLASH_OPTKEYR)

The user option bytes can be protected against unwanted erase/program operations by setting the OPTLOCK bit by software.

Modifying user option bytes

To modify the user option value, follow the sequence below:

1. Check that no Flash memory operation is ongoing by checking the BSY bit in the FLASH_SR register
2. Write the desired option value in the FLASH_OPTCR register.
3. Set the option start bit (OPTSTRT) in the FLASH_OPTCR register
4. Wait for the BSY bit to be cleared.

Note: *The value of an option is automatically modified by first erasing the information block and then programming all the option bytes with the values contained in the FLASH_OPTCR register.*

Note: *When setting the OPTSTRT bit in the FLASH_OPTCR register and before polling the BSY bit to be cleared, the software can issue a DSB instruction to guarantee the completion of a previous access to the FLASH_OPTCR register.*

3.5 FLASH memory protection

3.5.1 Read protection (RDP)

The user area in the Flash memory can be protected against read operations by an entrusted code. Three read protection levels are defined:

- Level 0: no read protection

When the read protection level is set to Level 0 by writing 0xAA into the read protection option byte (RDP), all read/write operations (if no write protection is set) from/to the Flash memory or the backup SRAM are possible in all boot configurations (Flash user boot, debug or boot from RAM).

- Level 1: read protection enabled

It is the default read protection level after option byte erase. The read protection Level 1 is activated by writing any value (except for 0xAA and 0xCC used to set Level 0 and Level 2, respectively) into the RDP option byte. When the read protection Level 1 is set:

- No access (read, erase, program) to Flash memory or backup SRAM can be performed while the debug feature is connected or while booting from RAM or system memory bootloader. A bus error is generated in case of read request.
- When booting from Flash memory, accesses (read, erase, program) to Flash memory and backup SRAM from user code are allowed.

When Level 1 is active, programming the protection option byte (RDP) to Level 0 causes the Flash memory and the backup SRAM to be mass-erased. As a result the

user code area is cleared before the read protection is removed. The mass erase only erases the user code area. The other option bytes including write protections remain unchanged from before the mass-erase operation.

A complete mass erase of the FLASH user code is done if PCROP_RDP is set.

A partial mass erase of the FLASH user code is done if PCROP_RDP is cleared. Only the sectors with no PCROP protection are erased.

The OTP area is not affected by mass erase and remains unchanged. The mass erase is performed only when Level 1 is active and Level 0 requested. When the protection level is increased (0->1, 1->2, 0->2) there is no mass erase.

- Level 2: debug/chip read disabled

The read protection Level 2 is activated by writing 0xCC to the RDP option byte. When the read protection Level 2 is set:

- All protections provided by Level 1 are active.
- Booting from RAM is no more allowed.
- Booting system memory bootloader is possible and all the commands are not accessible except Get, GetID and GetVersion. Refer to AN2606.
- JTAG, SWV (serial-wire viewer), ETM, and boundary scan are disabled.
- User option bytes can no longer be changed.
- When booting from Flash memory, accesses (read, erase and program) to Flash memory and backup SRAM from user code are allowed.

Memory read protection Level 2 is an irreversible operation. When Level 2 is activated, the level of protection cannot be decreased to Level 0 or Level 1.

Note: *The JTAG port is permanently disabled when Level 2 is active (acting as a JTAG fuse). As a consequence, boundary scan cannot be performed. STMicroelectronics is not able to perform analysis on defective parts on which the Level 2 protection has been set.*

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

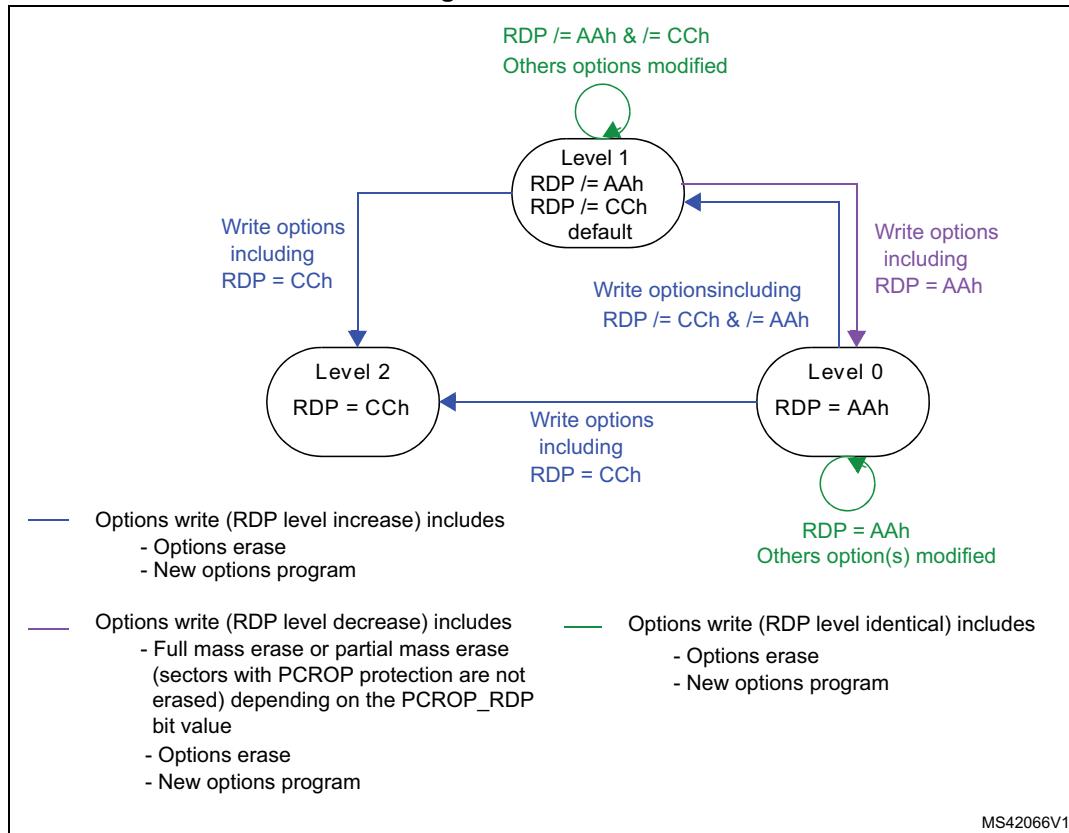
Table 9. Access versus read protection level

Memory area	Protection Level	Debug features, Boot from RAM or from System memory bootloader			Booting from Flash memory		
		Read	Write	Erase	Read	Write	Erase
Main Flash Memory and Backup SRAM	Level 1	NO		NO ⁽¹⁾	YES		
	Level 2	NO			YES		
Option Bytes	Level 1	YES			YES		
	Level 2	NO			NO		
OTP	Level 1	NO		NA	YES	NA	
	Level 2	NO		NA	YES	NA	

1. The main Flash memory and backup SRAM are only erased when the RDP changes from level 1 to 0. The OTP area remains unchanged.

[Figure 4](#) shows how to go from one RDP level to another.

Figure 4. RDP levels



3.5.2 Write protections

Up to 8 user sectors in Flash memory can be protected against unwanted write operations due to loss of program counter contexts. When the non-write protection nWRPi bit ($0 \leq i \leq 7$) in the FLASH_OPTCR or FLASH_OPTCR1 registers is low, the corresponding sector cannot be erased or programmed. Consequently, a mass erase cannot be performed if one of the sectors is write-protected.

If an erase/program operation to a write-protected part of the Flash memory is attempted (sector protected by write protection bit, OTP part locked or part of the Flash memory that can never be written like the ICP), the write protection error flag (WRPERR) is set in the FLASH_SR register.

Note:

When the memory read protection level is selected (RDP level = 1), it is not possible to program or erase Flash memory sector i if the CPU debug features are connected (JTAG or single wire) or boot code is being executed from RAM, even if nWRPi = 1.

Write protection error flag

If an erase/program operation to a write protected area of the Flash memory is performed, the Write Protection Error flag (WRPERR) is set in the FLASH_SR register.

If an erase operation is requested, the WRPERR bit is set when:

- A sector erase is requested and the Sector Number SNB field is not valid
- A mass erase is requested while at least one of the user sector is write protected by option bit (MER and nWR_i = 0 with $0 \leq i \leq 7$ bits in the FLASH_OPTCR register)
- A sector erase is requested on a write protected sector. (SER = 1, SNB = i and nWR_i = 0 with $0 \leq i \leq 7$ bits in the FLASH_OPTCR register)
- The Flash memory is readout protected and an intrusion is detected.

If a program operation is requested, the WRPERR bit is set when:

- A write operation is performed on system memory or on the reserved part of the user specific sector.
- A write operation is performed to the information block
- A write operation is performed on a sector write protected by option bit.
- A write operation is requested on an OTP area which is already locked
- The Flash memory is read protected and an intrusion is detected.

3.5.3 Proprietary code readout protection (PCROP)

The Flash memory user sectors (0 to 7) can be protected against D-bus read accesses by using the proprietary readout protection (PCROP).

The PCROP protection is activated sector by sector through the option bit PCROP[i] in the FLASH_OPTCR2 register:

- PCROP[i] = 0: PCROP protection not active on sector i ($i = 0..7$)
- PCROP[i] = 1: PCROP protection active on sector i ($i = 0..7$)

The PCROPed sectors are also write protected so they have all features described in [Section 3.5.2: Write protections](#).

The debug events are masked while executing the code in PCPROOed sectors.

Any read access (other than fetch) to PCROPed sectors performed through ITCM or AXI bus will trigger:

- A bus error on the given bus
- The RDERR flag to be set in the FLASH_SR status register. An interrupt is also generated if the Read Error Interrupt Enable bit RDERRIE is set in the FLASH_CR register.

Any program/erase operation on a PCROPed sector triggers a WRPERR flag error.

PCROP modification

It is possible to add PCROP sectors (by setting PCROPi, $i = 0..7$) with no restriction in Level 0 or Level 1.

It is possible to remove PCROP sectors (PCROPi from 1 to 0) only by:

- Clearing the PCROPi bit of the corresponding sectors (multiple sectors could be done at same time)
- Doing a regression level from Level 1 to Level 0
- Having PCROP_RDP already set (see note below)

Note: *Removing the PCROP attribute on at least one sector is mass erasing the FLASH memory. It is highly recommended to remove all PCROP attributes at same time to have a virgin Flash memory with no PCROP attribute.*

PCROPi and nWRPi are independent. [Table 10](#) shows what type of protection is set on a sector i according to WRPi and PCROPi bits values:

Table 10. Protections on sector i

nWRPi	PCROPi	Protection on sector i
1	0	No protection
0	0	Write protection
X	1	PCROP protection

PCROP_RDP

When the PCROP_RDP option bit is cleared, a full mass erase during RDP regression (level 1 to level 0) is replaced by a partial mass erase. Only non PCROP sectors are erased during this partial mass erase. The sectors which have PCROP protection are not erased. If the PCROP_RDP option bit is set, the PCROPed sectors are managed as the other sectors and are erased.

Note: *Doing regression level from Level 1 to Level 0 at same time than modifying the PCROP_RDP bit is allowed. As the full or partial mass erased is launched before an option modification, the current PCROP_RDP is used and not the new PCROP_RDP being programmed.*

3.6 One-time programmable bytes

[Table 11](#) shows the organization of the one-time programmable (OTP) part of the OTP area.

Table 11. OTP area organization

OTP Block	[127:96]	[95:64]	[63:32]	[31:0]	Address byte 0
0	OTP0	OTP0	OTP0	OTP0	0x1FF0 0x7800
	OTP0	OTP0	OTP0	OTP0	0x1FF0 0x7810
1	OTP1	OTP1	OTP1	OTP1	0x1FF0 0x7820
	OTP1	OTP1	OTP1	OTP1	0x1FF0 0x7830
-	-	-	-	-	-
-	-	-	-	-	-
-	-	-	-	-	-
14	OPT14	OPT14	OPT14	OPT14	0x1FF0 0x79C0
	OPT14	OPT14	OPT14	OPT14	0x1FF0 0x79D0

Table 11. OTP area organization (continued)

OTP Block	[127:96]	[95:64]	[63:32]	[31:0]	Address byte 0
15	OPT15	OPT15	OPT15	OPT15	0x1FF0 0x79E0
	OPT15	OPT15	OPT15	OPT15	0x1FF0 0x79F0
Lock block	LOCK15... LOCKB12	LOCK11... LOCKB8	LOCK7... LOCKB4	LOCK3... LOCKB0	0x1FF0 0x7A00

The OTP area is divided into 16 OTP data blocks of 32 bytes and one lock OTP block of 16 bytes. The OTP data and lock blocks cannot be erased. The lock block contains 16 bytes LOCKBi ($0 \leq i \leq 15$) to lock the corresponding OTP data block (blocks 0 to 15). Each OTP data block can be programmed until the value 0x00 is programmed in the corresponding OTP lock byte. The lock bytes must only contain 0x00 and 0xFF values, otherwise the OTP bytes might not be taken into account correctly.

3.7 FLASH registers

3.7.1 Flash access control register (FLASH_ACR)

The Flash access control register is used to enable/disable the acceleration features and control the Flash memory access time according to CPU frequency.

Address offset: 0x000

Reset value: 0x0000 0000

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

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

Bits 31:12 Reserved, must be kept cleared.

Bit 11 **ARTRST**: ART Accelerator reset

- 0: ART Accelerator is not reset
- 1: ART Accelerator is reset

Bit 10 Reserved, must be kept cleared.

Bit 9 **ARTEN**: ART Accelerator Enable

- 0: ART Accelerator is disabled
- 1: ART Accelerator is enabled

Bit 8 **PRFTEN**: Prefetch enable

0: Prefetch is disabled

1: Prefetch is enabled

Bits 7:4 Reserved, must be kept cleared.

Bits 3:0 **LATENCY[3:0]**: Latency

These bits represent the ratio of the CPU clock period to the Flash memory access time.

0000: Zero wait state

0001: One wait state

0010: Two wait states

-

-

-

1110: Fourteen wait states

1111: Fifteen wait states

3.7.2 Flash key register (FLASH_KEYR)

The Flash key register is used to allow access to the Flash control register and so, to allow program and erase operations.

Address offset: 0x04

Reset value: 0x0000 0000

Access: no wait state, word access

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

Bits 31:0 **FKEYR[31:0]**: FPEC key

The following values must be programmed consecutively to unlock the FLASH_CR register and allow programming/erasing it:

- a) KEY1 = 0x45670123
- b) KEY2 = 0xCDEF89AB

3.7.3 Flash option key register (FLASH_OPTKEYR)

The Flash option key register is used to allow program and erase operations in the information block.

Address offset: 0x08

Reset value: 0x0000 0000

Access: no wait state, word access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OPTKEYR[31:16]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

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

Bits 31:0 **OPTKEYR[31:0]**: Option byte key

The following values must be programmed consecutively to unlock the FLASH_OPTCR register and allow programming it:

- a) OPTKEY1 = 0x08192A3B
- b) OPTKEY2 = 0x4C5D6E7F

3.7.4 Flash status register (FLASH_SR)

The Flash status register gives information on ongoing program and erase operations.

Address offset: 0x0C

Reset value: 0x0000 0000

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	BSY														
															r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	RDERR	ERSERR	PGPERR	PGAERR	WRPERR	Res.	Res.	OPERR	EOP						
							rc_w1	rc_w1	rc_w1	rc_w1	rc_w1			rc_w1	rc_w1

Bits 31:17 Reserved, must be kept cleared.

Bit 16 **BSY**: Busy

This bit indicates that a Flash memory operation is in progress. It is set at the beginning of a Flash memory operation and cleared when the operation finishes or an error occurs.

- 0: no Flash memory operation ongoing
- 1: Flash memory operation ongoing

Bits 15:9 Reserved, must be kept cleared.

Bit 8 **RDERR**: PCROP protection error

Set by hardware when an address to be read is data (ITCM or AXI) and belongs to a PCROPed sector. If RDERRIE bit in the FLASH_CR register is set, an interrupt is generated.

Bit 7 **ERSERR**: Erase Sequence Error

Set by hardware when a write access to the Flash memory is performed by the code while the control register has not been correctly configured.

Cleared by writing 1.

Bit 6 **PGPERR**: Programming parallelism error

Set by hardware when the size of the access (byte, half-word, word, double word) during the program sequence does not correspond to the parallelism configuration PSIZE (x8, x16, x32, x64).

Cleared by writing 1.

Bit 5 **PGAERR**: Programming alignment error

Set by hardware when the data to program cannot be contained in the same 128-bit Flash memory row.

Cleared by writing 1.

Bit 4 **WRPERR**: Write protection error

Set by hardware when an address to be erased/programmed belongs to a write-protected part of the Flash memory.

Cleared by writing 1.

Bits 3:2 Reserved, must be kept cleared.

Bit 1 **OPERR**: Operation error

Set by hardware when a flash operation (programming / erase /read) request is detected and can not be run because of parallelism, alignment, or write protection error. This bit is set only if error interrupts are enabled (ERRIE = 1).

Bit 0 **EOP**: End of operation

Set by hardware when one or more Flash memory operations (program/erase) has/have completed successfully. It is set only if the end of operation interrupts are enabled (EOPIE = 1).

Cleared by writing a 1.

3.7.5 Flash control register (FLASH_CR)

The Flash control register is used to configure and start Flash memory operations.

Address offset: 0x10

Reset value: 0x8000 0000

Access: no wait state when no Flash memory operation is ongoing, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
LOCK	Res.	Res.	Res.	Res.	RDERRIE	ERRIE	EOPIE	Res.	STRT							
rs					rw	rw	rw								rs	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	Res.	Res.	Res.	Res.	Res.	PSIZE[1:0]	Res.							MER	SER	PG
						rw	rw		rw							

Bit 31 **LOCK**: Lock

Write to 1 only. When it is set, this bit indicates that the FLASH_CR register is locked. It is cleared by hardware after detecting the unlock sequence.

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

Bits 30:27 Reserved, must be kept cleared.

Bit 26 **RDERRIE**: PCROP error interrupt enable

This bit enables the interrupt generation when the RDERR flag is set in the FLASH_SR register.

0: PCROP error interrupt generation is disabled.

1: PCROP error interrupt generation is enabled.

Bit 25 **ERRIE**: Error interrupt enable

This bit enables the interrupt generation when the OPERR bit in the FLASH_SR register is set to 1.

0: Error interrupt generation disabled

1: Error interrupt generation enabled

Bit 24 **EOPIE:** End of operation interrupt enable

This bit enables the interrupt generation when the EOP bit in the FLASH_SR register goes to 1.

0: Interrupt generation disabled

1: Interrupt generation enabled

Bits 23:17 Reserved, must be kept cleared.

Bit 16 **STRT:** Start

This bit triggers an erase operation when set. It is set only by software and cleared when the BSY bit is cleared.

Bits 15:10 Reserved, must be kept cleared.

Bits 9:8 **PSIZE:** Program size

These bits select the program parallelism.

00 program x8

01 program x16

10 program x32

11 program x64

Bit 7 Reserved, must be kept cleared.

Bits 6:3 **SNB[3:0]:** Sector number

These bits select the sector to erase.

0000 sector 0

0001 sector 1

...

0111 sector 7

Others not allowed

Bit 2 **MER:** Mass Erase

Erase activated for all user sectors.

Bit 1 **SER:** Sector Erase

Sector Erase activated.

Bit 0 **PG:** Programming

Flash programming activated.

3.7.6 Flash option control register (FLASH_OPTCR)

The FLASH_OPTCR register is used to modify the user option bytes.

Address offset: 0x14

Reset value: 0xC0FF AA FD. The option bytes are loaded with values from Flash memory at reset release.

Access: no wait state when no Flash memory operation is ongoing, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IWDG_STOP	IWDG_STDBY	Res.	Res.	Res.	Res.	Res.	Res.	nWRP[7:0]							
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
RDP[7:0]								nRST_STDBY	nRST_STOP	IWDG_SW	WWDG_SW	BOR_LEV[1:0]	OPTST_RT	OPTLO_CK	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rs	rs

Bit 31 **IWDG_STOP**: Independent watchdog counter freeze in Stop mode

- 0: Freeze IWDG counter in STOP mode.
- 1: IWDG counter active in STOP mode.

Bit 30 **IWDG_STDBY**: Independent watchdog counter freeze in standby mode

- 0: Freeze IWDG counter in standby mode.
- 1: IWDG counter active in standby mode.

Bits 29:24 Reserved, must be kept cleared.

Bits 23:16 **nWRP[7:0]**: Not write protect

These bits contain the value of the write-protection option bytes for sectors 0 to 7 after reset. They can be written to program a new write-protect into Flash memory.

- 0: Write protection active on sector i
- 1: Write protection not active on sector i

Bits 15:8 **RDP[7:0]**: Read protect

These bits contain the value of the read-protection option level after reset. They can be written to program a new read protection value into Flash memory.

- 0xAA: Level 0, read protection not active
- 0xCC: Level 2, chip read protection active
- Others: Level 1, read protection of memories active

Bits 7:4 **USER**: User option bytes

These bits contain the value of the user option byte after reset. They can be written to program a new user option byte value into Flash memory.

- Bit 7: nRST_STDBY
- Bit 6: nRST_STOP
- Bit 5: IWDG_SW
- Bit 4: WWDG_SW

Bits 3:2 BOR_lev[1:0]: BOR reset Level

These bits contain the supply level threshold that activates/releases the reset. They can be written to program a new BOR level. By default, BOR is off. When the supply voltage (V_{DD}) drops below the selected BOR level, a device reset is generated.

00: BOR Level 3 (VBOR3), brownout threshold level 3

01: BOR Level 2 (VBOR2), brownout threshold level 2

10: BOR Level 1 (VBOR1), brownout threshold level 1

11: BOR off, POR/PDR reset threshold level is applied

Note: For full details on BOR characteristics, refer to the “Electrical characteristics” section of the product datasheet.

Bit 1 OPTSTRT: Option start

This bit triggers a user option operation when set. It is set only by software and cleared when the BSY bit is cleared.

Bit 0 OPTLOCK: Option lock

Write to 1 only. When this bit is set, it indicates that the FLASH_OPTCR register is locked.

This bit is cleared by hardware after detecting the unlock sequence.

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

Note: When modifying the IWDG_SW, IWDG_STOP or IWDG_STDBY option byte, a system reset is required to make the change effective.

3.7.7 Flash option control register (FLASH_OPTCR1)

The FLASH_OPTCR1 register is used to modify the user option bytes.

Address offset: 0x18

Reset value: 0x0040 0080 (ITCM-FLASH). The option bytes are loaded with values from Flash memory at reset release.

Access: no wait state when no Flash memory operation is ongoing, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BOOT_ADD1[15:0]															
rw															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BOOT_ADD0[15:0]															
rw															

Bits 31:16 **BOOT_ADD1[15:0]**: Boot base address when Boot pin =1

BOOT_ADD1[15:0] correspond to address [29:14],

The boot memory address can be programmed to any address in the range 0x0000 0000 to 0x2004 FFFF with a granularity of 16KB.

Example:

BOOT_ADD1 = 0x0000: Boot from ITCM RAM (0x0000 0000)

BOOT_ADD1 = 0x0040: Boot from System memory bootloader (0x0010 0000)

BOOT_ADD1 = 0x0080: Boot from Flash on ITCM interface (0x0020 0000)

BOOT_ADD1 = 0x2000: Boot from Flash on AXIM interface (0x0800 0000)

BOOT_ADD1 = 0x8000: Boot from DTCM RAM (0x2000 0000)

BOOT_ADD1 = 0x8004: Boot from SRAM1 (0x2001 0000)

BOOT_ADD1 = 0x8013: Boot from SRAM2 (0x2003 C000)

Bits 15:0 **BOOT_ADD0[15:0]**: Boot base address when Boot pin =0

BOOT_ADD0[15:0] correspond to address [29:14],

The boot base address can be programmed to any address in the range 0x0000 0000 to 0x2004 FFFF with a granularity of 16KB.

Example:

BOOT_ADD0 = 0x0000: Boot from ITCM RAM (0x0000 0000)

BOOT_ADD0 = 0x0040: Boot from System memory bootloader (0x0010 0000)

BOOT_ADD0 = 0x0080: Boot from Flash on ITCM interface (0x0020 0000)

BOOT_ADD0 = 0x2000: Boot from Flash on AXIM interface (0x0800 0000)

BOOT_ADD0 = 0x8000: Boot from DTCM RAM (0x2000 0000)

BOOT_ADD0 = 0x8004: Boot from SRAM1 (0x2001 0000)

BOOT_ADD0 = 0x8013: Boot from SRAM2 (0x2003 C000)

3.7.8 Flash option control register (FLASH_OPTCR2)

The FLASH_OPTCR2 register is used to modify the user option bytes.

Address offset: 0x1C

Reset value: 0x8000 00FF. The option bytes are loaded with values from the Flash memory at reset release.

Access: no wait state when no Flash memory operation is ongoing, word, half-word and byte access.

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

Bit 31 **PCROP_RDP**: PCROP zone preserved when RDP level decreased.

0: PCROP zone is kept when RDP is decreased: Partial mass erase is done.

1: PCROP zone is erased when RDP is decreased: Full mass erase is done

Bits 31:8 Reserved.

Bits 7:0 **PCROPi**: PCROP option byte

0: PCROP protection not active on sector i; i = 0..7

1: PCROP protection active on sector i; i = 0..7

3.7.9 Flash interface register map

Table 12. Flash 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	FLASH_ACR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	LATENCY[3:0]			
	Reset value																																
0x04	FLASH_KEYR	KEY[31:16]												KEY[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	0		
0x08	FLASH_OPTKEYR	OPTKEYR[31:16]												OPTKEYR[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	0		
0x0C	FLASH_SR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x10	FLASH_CR	1	IWDG_STOP	LOCK																													
	Reset value	0	IWDG_STDBY		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.												
0x14	FLASH_OPTCR	nWRP[7:0]												RDP[7:0]																			
	Reset value	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1		
0x18	FLASH_OPTCR1	BOOT_ADD1[15:0]												BOOT_ADD0[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	0		
0x1C	FLASH_OPTCR2	PCROPi												PCROPi																			
	Reset value	1	PCROP_RDP	Res.	0	0	1	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1											

4 Power controller (PWR)

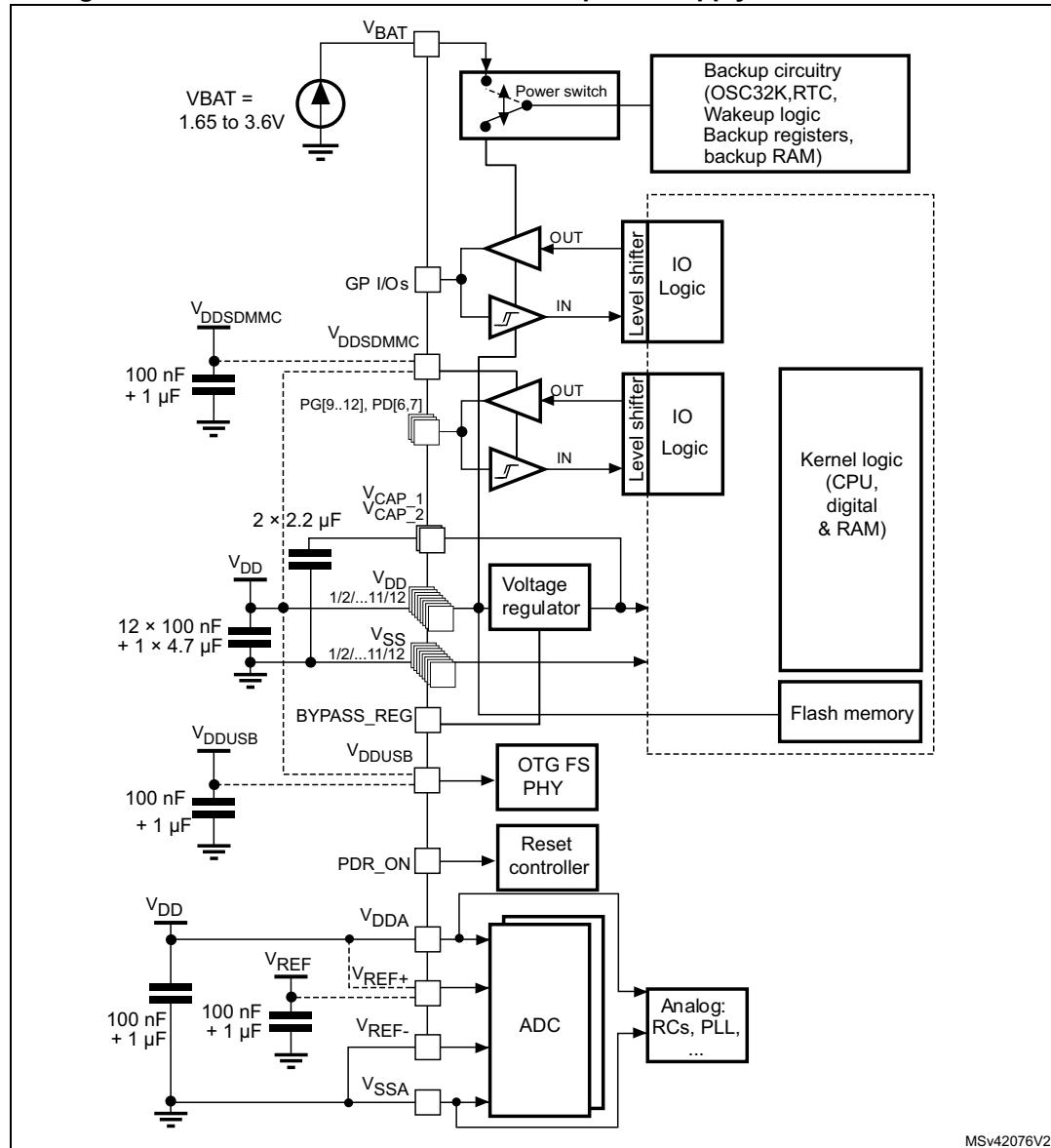
4.1 Power supplies

The device requires a 1.8 to 3.6 V operating voltage supply (V_{DD}). An embedded linear voltage regulator is used to supply the internal 1.2 V digital power.

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

Note: Depending on the operating power supply range, some peripheral may be used with limited functionality and performance. For more details refer to section "General operating conditions" in STM32F72xxx and STM32F73xxx datasheets.

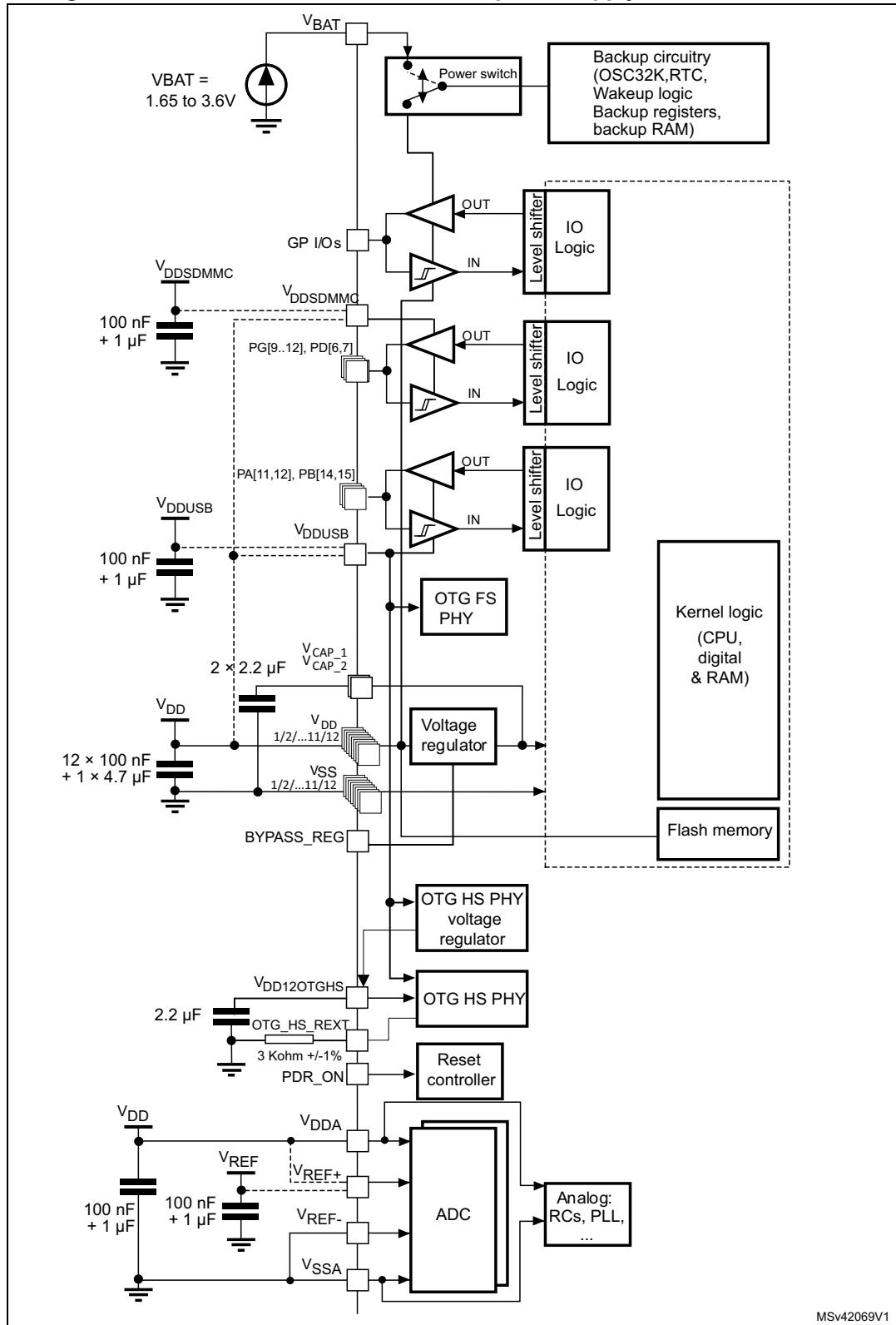
Figure 5. STM32F7x2xx and STM32F730xx power supply overview



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

MSv42076V2

Figure 6. STM32F7x3xx and STM32F730xx power supply overview



1. The V_{DDUSB} allows supplying the PHY FS in PA11/PA12 and the PHY HS on PB14/PB15.

MSv42069V1

2. V_{DDA} and V_{SSA} must be connected to V_{DD} and V_{SS} , respectively.
3. Refer to STM32F730xx datasheet for more details on available packages supporting the integrated OTG_HS PHY.

4.1.1 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.

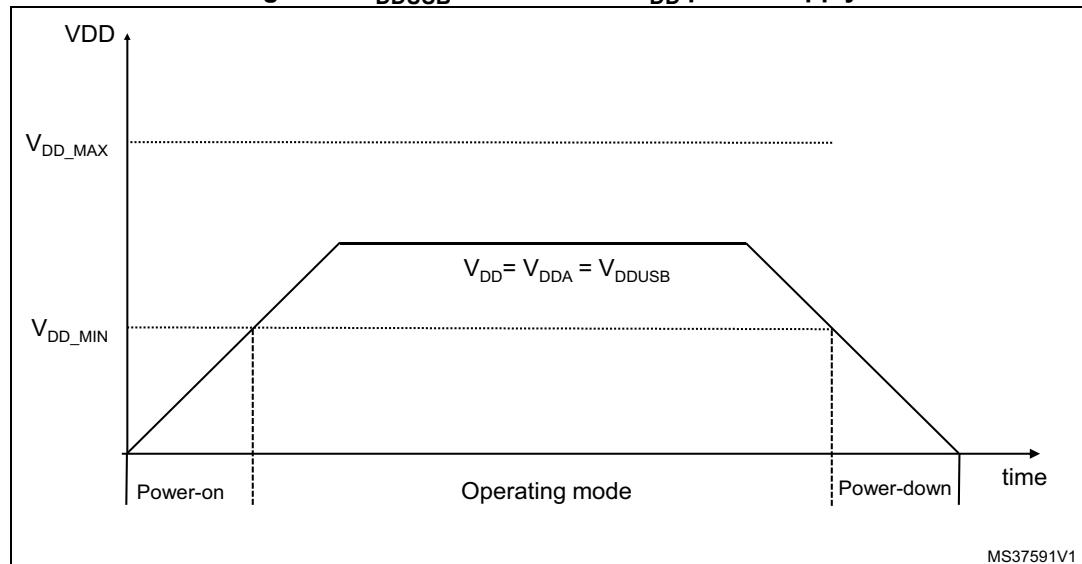
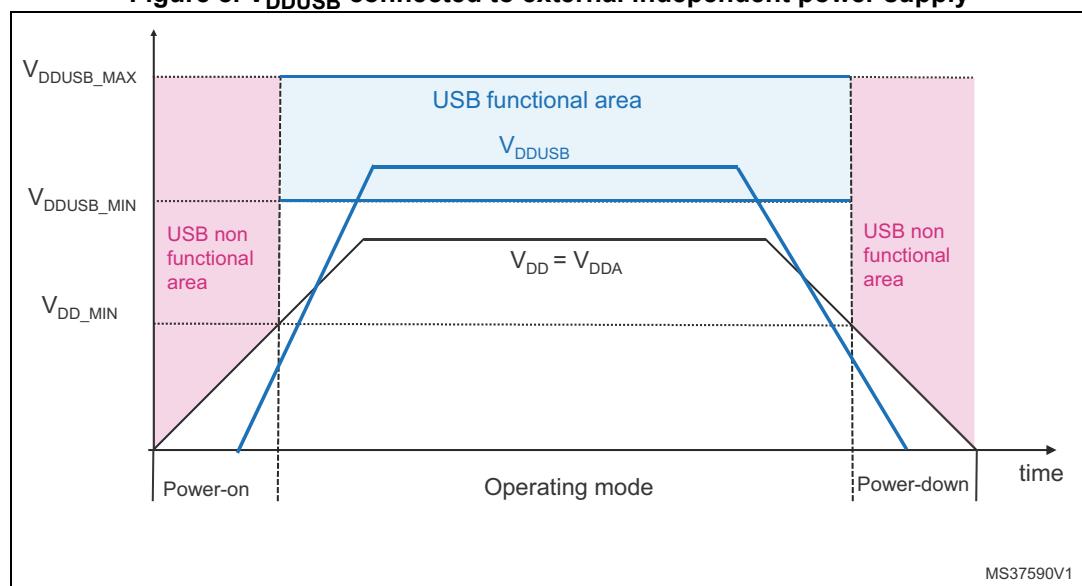
- The ADC voltage supply input is available on a separate V_{DDA} pin.
- An isolated supply ground connection is provided on pin V_{SSA} .

To ensure a better accuracy of low voltage inputs, the user can connect a separate external reference voltage ADC input on V_{REF} . The voltage on V_{REF} ranges from 1.8 V to V_{DDA} .

4.1.2 Independent USB transceivers supply

The V_{DDUSB} is an independent USB power supply for full speed transceivers (USB OTG FS and USB OTG HS in FS mode). It can be connected either to V_{DD} or an external independent power supply (3.0 to 3.6V) for USB transceivers (refer [Figure 7](#) and [Figure 8](#)). For example, when the device is powered at 1.8V, an independent power supply 3.3V can be connected to V_{DDUSB} . When the V_{DDUSB} is connected to a separated power supply, it is independent from V_{DD} or V_{DDA} but it must be the last supply to be provided and the first to disappear. The following conditions V_{DDUSB} must be respected:

- During power-on phase ($V_{DD} < V_{DD_MIN}$), V_{DDUSB} should be always lower than V_{DD}
- During power-down phase ($V_{DD} < V_{DD_MIN}$), V_{DDUSB} should be always lower than V_{DD}
- V_{DDUSB} rising and falling time rate specifications must be respected
- In operating mode phase, V_{DDUSB} could be lower or higher than V_{DD} :
 - If USB (USB OTG_HS/OTG_FS) is used, the associated GPIOs powered by V_{DDUSB} are operating between V_{DDUSB_MIN} and V_{DDUSB_MAX} .
 - The V_{DDUSB} supplies both USB transceiver (USB OTG_HS and USB OTG_FS). If only one USB transceiver is used in the application, the GPIOs associated to the other USB transceiver are still supplied at by V_{DDUSB} .
 - If USB (USB OTG_HS/OTG_FS) is not used, the associated GPIOs powered by V_{DDUSB} are operating between V_{DD_MIN} and V_{DD_MAX} .

Figure 7. V_{DDUSB} connected to V_{DD} power supply**Figure 8. V_{DDUSB} connected to external independent power supply**

In the STM32F7x3xx and STM32F730xx devices, the USB PHY HS sub-system uses an additional power supply pin:

- The VDD12OTGHS pin is the output of the PHY HS regulator (1.2 V). An external capacitor of 2.2 μ F must be connected on the VDD12OTGHS pin.

Note:

The PHY HS has another OTG_HS_REXT pin needed for calibration. This pin must be connected to gnd via an external precise resistor (3 Kohm +/- 1%).

4.1.3 Battery backup domain

Backup domain description

To retain the content of the RTC backup registers, backup SRAM, and supply the RTC 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.

To allow the RTC to operate even when the main digital supply (V_{DD}) is turned off, the V_{BAT} pin powers the following blocks:

- The RTC
- The LSE oscillator
- The backup SRAM when the low-power backup regulator is enabled
- PC13 to PC15 I/Os, plus PI8 I/O (when available)

The switch to the V_{BAT} supply is controlled by the power-down reset embedded in the Reset block.

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

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

If the power supply/battery connected to the V_{BAT} pin cannot support this current injection, it is strongly recommended to connect an external low-drop diode between this power supply and the V_{BAT} pin.

If no external battery is used in the application, it is recommended to connect the V_{BAT} pin to V_{DD} with a 100 nF external decoupling ceramic capacitor in parallel.

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 and PI8 can be used as a GPIO pin (refer to [Table 137: RTC pin PC13 configuration](#) and [Table 138: RTC pin PI8 configuration](#) for more details about these pins configuration)

Note: Due to the fact that the switch only sinks a limited amount of current (3 mA), the use of GPIOs PI8 and PC13 to PC15 are restricted: the speed has to be limited to 2 MHz with a maximum load of 30 pF and these I/Os 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 (TAMP1)
- PI8 can be used as tamper pin (TAMP2)

Backup domain access

After reset, the backup domain (RTC registers, RTC backup register and backup SRAM) is protected against possible unwanted write accesses. To enable access to the backup domain, proceed as follows:

- Access to the RTC and RTC backup registers
- 1. Enable the power interface clock by setting the PWREN bits in the RCC_APB1ENR register (see [Section 5.3.13](#))
- 2. Set the DBP bit in the *PWR power control register (PWR_CR1)* to enable access to the backup domain
- 3. Select the RTC clock source: see [Section 5.2.8: RTC/AWU clock](#)
- 4. Enable the RTC clock by programming the RTCEN [15] bit in the *RCC backup domain control register (RCC_BDCR)*
- Access to the backup SRAM
- 1. Enable the power interface clock by setting the PWREN bits in the RCC_APB1ENR register (see [Section 5.3.13](#))
- 2. Set the DBP bit in the *PWR power control register (PWR_CR1)* to enable access to the backup domain
- 3. Enable the backup SRAM clock by setting BKPSRAMEN bit in the *RCC AHB1 peripheral clock register (RCC_AHB1ENR)*.

RTC and RTC backup registers

The real-time clock (RTC) is an independent BCD timer/counter. The RTC provides a time-of-day clock/calendar, two programmable alarm interrupts, and a periodic programmable wakeup flag with interrupt capability. The RTC contains 32 backup data registers (128 bytes) which are reset when a tamper detection event occurs. For more details refer to [Section 25: Real-time clock \(RTC\)](#).

Backup SRAM

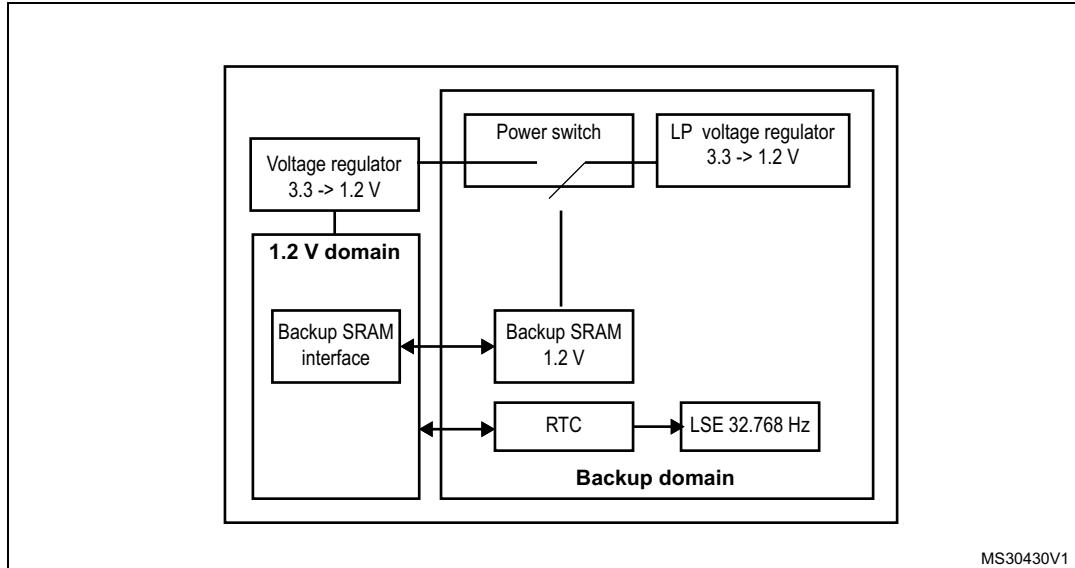
The backup domain includes 4 Kbytes of backup SRAM addressed in 32-bit, 16-bit or 8-bit mode. Its content is retained even in Standby or V_{BAT} mode when the low-power backup regulator is enabled. It can be considered as an internal EEPROM when V_{BAT} is always present.

When the backup domain is supplied by V_{DD} (analog switch connected to V_{DD}), the backup SRAM is powered from V_{DD} which replaces the V_{BAT} power supply to save battery life.

When the backup domain is supplied by V_{BAT} (analog switch connected to V_{BAT} because V_{DD} is not present), the backup SRAM is powered by a dedicated low-power regulator. This regulator can be ON or OFF depending whether the application needs the backup SRAM function in Standby and V_{BAT} modes or not. The power-down of this regulator is controlled by a dedicated bit, the BRE control bit of the PWR_CSR1 register (see [Section 4.4.2: PWR power control/status register \(PWR_CSR1\)](#)).

The backup SRAM is not mass erased by a tamper event. It is read protected to prevent confidential data, such as cryptographic private key, from being accessed. The backup SRAM can be erased only through the Flash interface when a protection level change from level 1 to level 0 is requested. Refer to the description of Read protection (RDP) option byte.

Figure 9. Backup domain



4.1.4 Voltage regulator

An embedded linear voltage regulator supplies all the digital circuitries except for the backup domain and the Standby circuitry. The regulator output voltage is around 1.2 V.

This voltage regulator requires two external capacitors to be connected to two dedicated pins, V_{CAP_1} and V_{CAP_2} available in all packages. Specific pins must be connected either to V_{SS} or V_{DD} to activate or deactivate the voltage regulator. These pins depend on the package.

When activated by software, the voltage regulator is always enabled after Reset. It works in three different modes depending on the application modes (Run, Stop, or Standby mode).

- In **Run mode**, the main regulator supplies full power to the 1.2 V domain (core, memories and digital peripherals). In this mode, the regulator output voltage (around 1.2 V) can be scaled by software to different voltage values (scale 1, scale 2, and scale 3 can be configured through VOS[1:0] bits of the PWR_CR1 register). The scale can be modified only when the PLL is OFF and the HSI or HSE clock source is selected as system clock source. The new value programmed is active only when the PLL is ON. When the PLL is OFF, the voltage scale 3 is automatically selected.

The voltage scaling allows optimizing the power consumption when the device is clocked below the maximum system frequency. After exit from Stop mode, the voltage scale 3 is automatically selected.(see [Section 4.4.1: PWR power control register](#)

(PWR_CR1).

2 operating modes are available:

- **Normal mode:** The CPU and core logic operate at maximum frequency at a given voltage scaling (scale 1, scale 2 or scale 3)
- **Over-drive mode:** This mode allows the CPU and the core logic to operate at a higher frequency than the normal mode for the voltage scaling scale 1 and scale 2.
- In **Stop mode:** the main regulator or low-power regulator supplies a low-power voltage to the 1.2V domain, thus preserving the content of registers and internal SRAM. The voltage regulator can be put either in main regulator mode (MR) or in low-power mode (LPR). Both modes can be configured by software as follows:
 - **Normal mode:** the 1.2 V domain is preserved in nominal leakage mode. It is the default mode when the main regulator (MR) or the low-power regulator (LPR) is enabled.
 - **Under-drive mode:** the 1.2 V domain is preserved in reduced leakage mode. This mode is only available with the main regulator or the low-power regulator mode (see [Table 13](#)).
- In **Standby mode:** the regulator is powered down. The content of the registers and SRAM are lost except for the Standby circuitry and the backup domain.

Note: Over-drive and under-drive mode are not available when the regulator is bypassed.

For more details, refer to the voltage regulator section in the datasheets.

Table 13. Voltage regulator configuration mode versus device operating mode⁽¹⁾

Voltage regulator configuration	Run mode	Sleep mode	Stop mode	Standby mode
Normal mode	MR	MR	MR or LPR	-
Over-drive mode ⁽²⁾	MR	MR	-	-
Under-drive mode	-	-	MR or LPR	-
Power-down mode	-	-	-	Yes

1. '-' means that the corresponding configuration is not available.

2. The over-drive mode is not available when $V_{DD} = 1.8$ to 2.1 V.

Entering Over-drive mode

It is recommended to enter Over-drive mode when the application is not running critical tasks and when the system clock source is either HSI or HSE. To optimize the configuration time, enable the Over-drive mode during the PLL lock phase.

To enter Over-drive mode, follow the sequence below:

1. Select HSI or HSE as system clock.
2. Configure RCC_PLLCFG register and set PLLON bit of RCC_CR register.
3. Set ODEN bit of PWR_CR1 register to enable the Over-drive mode and wait for the ODRDY flag to be set in the PWR_CSR1 register.
4. Set the ODSW bit in the PWR_CR1 register to switch the voltage regulator from Normal mode to Over-drive mode. The System will be stalled during the switch but the PLL clock system will be still running during locking phase.
5. Wait for the ODSWRDY flag in the PWR_CSR1 to be set.
6. Select the required Flash latency as well as AHB and APB prescalers.
7. Wait for PLL lock.
8. Switch the system clock to the PLL.
9. Enable the peripherals that are not generated by the System PLL (I2S clock, SAI1 clock, USB_48MHz clock....).

Note: The PLLI2S and PLLSAI can be configured at the same time as the system PLL.

During the Over-drive switch activation, no peripheral clocks should be enabled. The peripheral clocks must be enabled once the Over-drive mode is activated.

Entering Stop mode disables the Over-drive mode, as well as the PLL. The application software has to configure again the Over-drive mode and the PLL after exiting from Stop mode.

Exiting from Over-drive mode

It is recommended to exit from Over-drive mode when the application is not running critical tasks and when the system clock source is either HSI or HSE. There are two sequences that allow exiting from over-drive mode:

- By resetting simultaneously the ODEN and ODSW bits in the PWR_CR1 register (sequence 1)
- By resetting first the ODSW bit to switch the voltage regulator to Normal mode and then resetting the ODEN bit to disable the Over-drive mode (sequence 2).

Example of sequence 1:

1. Select HSI or HSE as system clock source.
2. Disable the peripheral clocks that are not generated by the System PLL (I2S clock, SAI1 clock, USB_48MHz clock,...)
3. Reset simultaneously the ODEN and the ODSW bits in the PWR_CR1 register to switch back the voltage regulator to Normal mode and disable the Over-drive mode.
4. Wait for the ODWRDY flag of PWR_CSR1 to be reset.

Example of sequence 2:

1. Select HSI or HSE as system clock source.
2. Disable the peripheral clocks that are not generated by the System PLL (I2S clock, SAI1 clock, USB_48MHz clock,...).
3. Reset the ODSW bit in the PWR_CR1 register to switch back the voltage regulator to Normal mode. The system clock is stalled during voltage switching.
4. Wait for the ODWRDY flag of PWR_CSR1 to be reset.
5. Reset the ODEN bit in the PWR_CR1 register to disable the Over-drive mode.

Note:

During step 3, the ODEN bit remains set and the Over-drive mode is still enabled but not active (ODSW bit is reset). If the ODEN bit is reset instead, the Over-drive mode is disabled and the voltage regulator is switched back to the initial voltage.

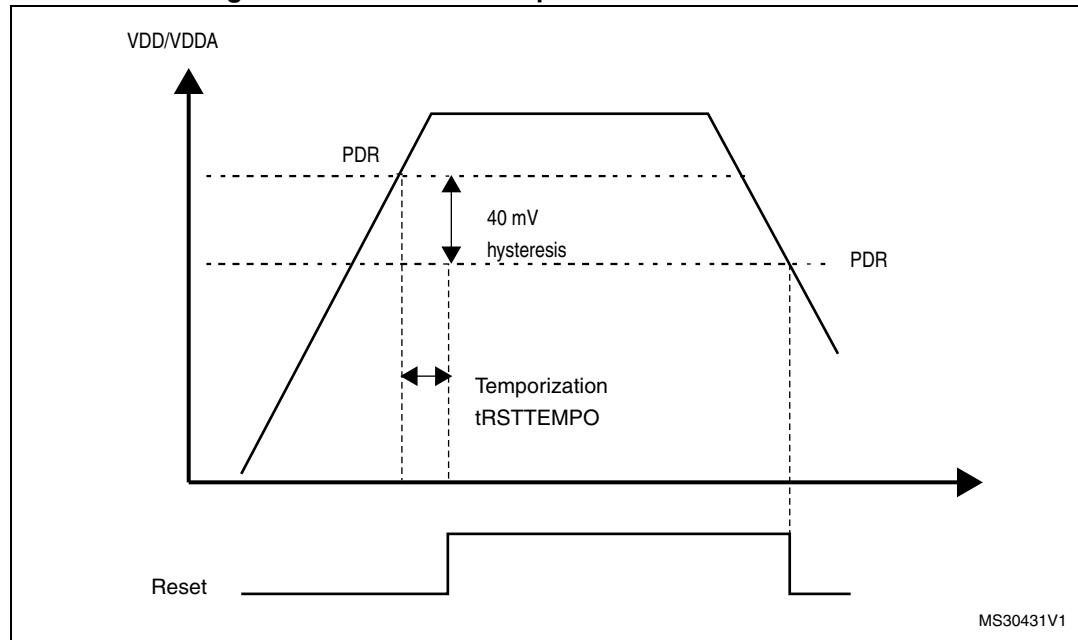
4.2 Power supply supervisor

4.2.1 Power-on reset (POR)/power-down reset (PDR)

The device has an integrated POR/PDR circuitry that allows proper operation starting from 1.8 V.

The device remains in Reset mode when V_{DD}/V_{DDA} 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 10. Power-on reset/power-down reset waveform



4.2.2 Brownout reset (BOR)

During power on, the Brownout reset (BOR) keeps the device under reset until the supply voltage reaches the specified V_{BOR} threshold.

V_{BOR} is configured through device option bytes. By default, BOR is off. 3 programmable V_{BOR} threshold levels can be selected:

- BOR Level 3 (VBOR3). Brownout threshold level 3.
- BOR Level 2 (VBOR2). Brownout threshold level 2.
- BOR Level 1 (VBOR1). Brownout threshold level 1.

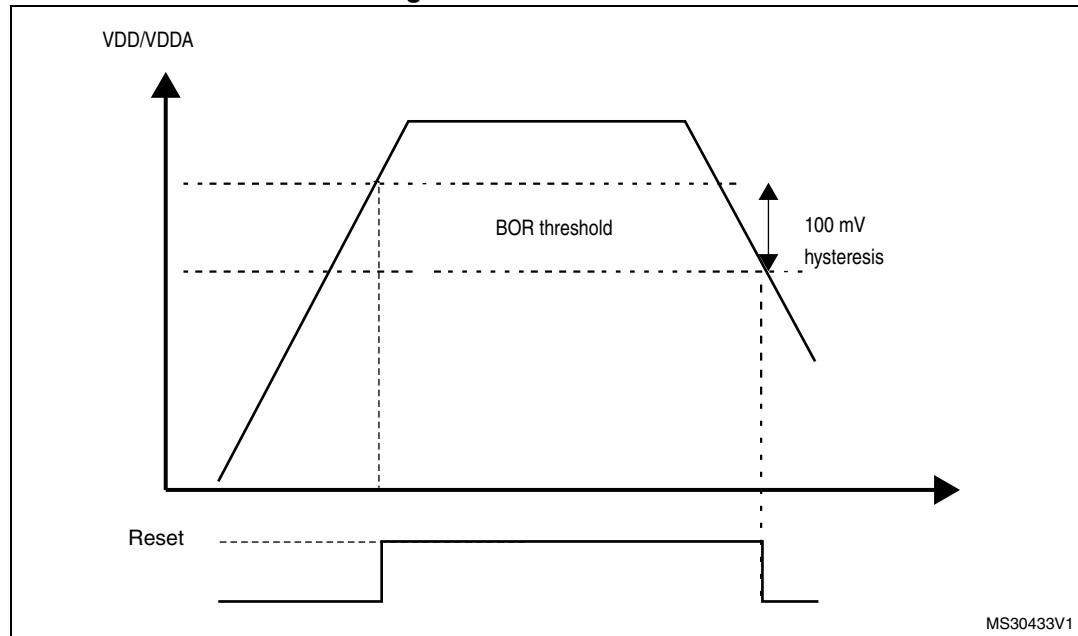
Note: *For full details about BOR characteristics, refer to the "Electrical characteristics" section in the device datasheet.*

When the supply voltage (V_{DD}) drops below the selected V_{BOR} threshold, a device reset is generated.

The BOR can be disabled by programming the device option bytes. In this case, the power-on and power-down is then monitored by the POR/ PDR (see [Section 4.2.1: Power-on reset \(POR\)/power-down reset \(PDR\)](#)).

The BOR threshold hysteresis is ~100 mV (between the rising and the falling edge of the supply voltage).

Figure 11. BOR thresholds



4.2.3 Programmable voltage detector (PVD)

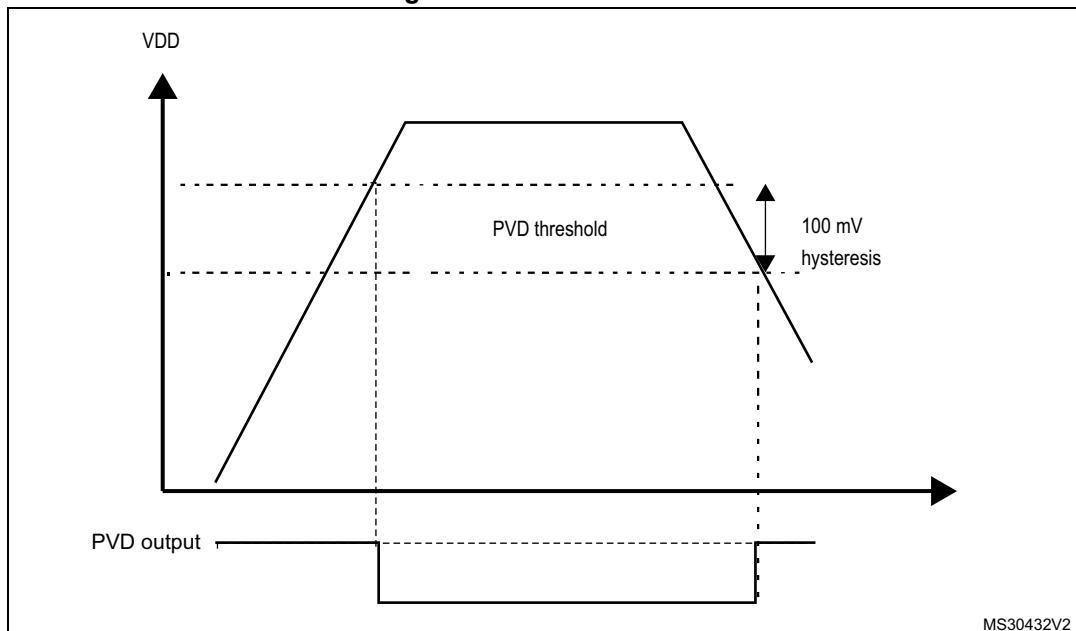
You can use the PVD to monitor the V_{DD} power supply by comparing it to a threshold selected by the PLS[2:0] bits in the [PWR power control register \(PWR_CR1\)](#).

The PVD is enabled by setting the PVDE bit.

A PVDO flag is available, in the [PWR power control/status register \(PWR_CSR1\)](#), 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 12. PVD thresholds



4.3 Low-power modes

By default, the microcontroller is in Run mode after a system or a power-on 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 devices feature three low-power modes:

- Sleep mode (Cortex[®]-M7 core stopped, peripherals kept running)
- Stop mode (all clocks are stopped)
- Standby mode (1.2 V 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 APBx and AHBx peripherals when they are unused.

Entering low-power mode

Low-power modes are entered by the MCU by executing the WFI (Wait For Interrupt), or WFE (Wait for Event) instructions, or when the SLEEPONEXIT bit in the Cortex[®]-M7 System Control register is set on Return from ISR.

Entering Low-power mode through WFI or WFE will be executed only if no interrupt is pending or no event is pending.

Exiting low-power mode

The MCU exits from Sleep and Stop modes low-power mode depending on the way the low-power mode was entered:

- If the WFI instruction or Return from ISR was used to enter the low-power mode, any peripheral interrupt acknowledged by the NVIC can wake up the device.
- If the WFE instruction is used to enter the low-power mode, the MCU exits the low-power mode as soon as an event occurs. The wakeup event can be generated either by:

- NVIC IRQ interrupt:

When SEVONPEND = 0 in the Cortex®-M7 System Control register: by enabling an interrupt in the peripheral control register and in the NVIC. When the MCU resumes from WFE, the peripheral interrupt pending bit and the NVIC peripheral IRQ channel pending bit (in the NVIC interrupt clear pending register) have to be cleared. Only NVIC interrupts with sufficient priority will wakeup and interrupt the MCU.

When SEVONPEND = 1 in the Cortex®-M7 System Control register: by enabling an interrupt in the peripheral control register and optionally in the NVIC. When the MCU resumes from WFE, the peripheral interrupt pending bit and when enabled the NVIC peripheral IRQ channel pending bit (in the NVIC interrupt clear pending register) have to be cleared. All NVIC interrupts will wakeup the MCU, even the disabled ones. Only enabled NVIC interrupts with sufficient priority will wakeup and interrupt the MCU.

- Event

This is done by configuring a EXTI line in event mode. When the CPU resumes from WFE, it is not necessary to clear the EXTI peripheral interrupt pending bit or the NVIC IRQ channel pending bit as the pending bits corresponding to the event line is not set. It may be necessary to clear the interrupt flag in the peripheral.

The MCU exits from Standby low-power mode through an external reset (NRST pin), an IWDG reset, a rising edge on one of the enabled WKUPx pins or a RTC event occurs (see [Figure 286: RTC block diagram](#)).

After waking up from Standby mode, program execution restarts in the same way as after a Reset (boot pin sampling, option bytes loading, reset vector is fetched, etc.).

Only enabled NVIC interrupts with sufficient priority will wakeup and interrupt the MCU.

Table 14. Low-power mode summary

Mode name	Entry	Wakeup	Effect on 1.2 V 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			

Table 14. Low-power mode summary (continued)

Mode name	Entry	Wakeup	Effect on 1.2 V domain clocks	Effect on VDD domain clocks	Voltage regulator
Stop	SLEEPDEEP bit + WFI or WFE	Any EXTI line (configured in the EXTI registers, internal and external lines)	All 1.2 V domain clocks OFF	HSI and HSE oscillators OFF	Main regulator or Low-power regulator (depends on <i>PWR power control register (PWR_CR1)</i>)
Standby	PDDS bit + SLEEPDEEP bit + WFI or WFE	WKUP pin rising or falling edge, RTC alarm (Alarm A or Alarm B), RTC Wakeup event, RTC tamper events, RTC time stamp event, external reset in NRST pin, IWDG reset			OFF

Table 15. Features over all modes ⁽¹⁾

Peripheral	Run	Sleep	Stop		Standby		VBAT
				Wakeup		Wakeup	
CPU	Y	-	-	-	-	-	-
Flash access	Y	Y	-	-	-	-	-
DTCM RAM	Y	Y	Y	-	-	-	-
ITCM RAM	Y	Y	Y	-	-	-	-
SRAM1	Y	Y	Y	-	-	-	-
SRAM2	Y	Y	Y	-	-	-	-
FMC	O	O	-	-	-	-	-
QUADSPI	O	O	-	-	-	-	-
Backup Registers	Y	Y	Y	-	Y	-	Y
Backup RAM	Y	Y	Y	-	Y	-	Y
Brown-out reset (BOR)	Y	Y	Y	Y	Y	Y	
Programmable Voltage Detector (PVD)	O	O	O	O	-	-	-
High Speed Internal (HSI)	O	O	(2)	-	-	-	-
High Speed External (HSE)	O	O		-	-	-	-
Low Speed Internal (LSI)	O	O	O	-	O	-	-
Low Speed External (LSE)	O	O	O	-	O	-	O
RTC	O	O	O	O	O	O	O

Table 15. Features over all modes (continued)⁽¹⁾

Peripheral	Run	Sleep	Stop		Standby		VBAT
				Wakeup		Wakeup	
Number of RTC tamper pins	3	3	3	3	3	3	2
CRC calculation unit	O	O	-	-	-	-	-
GPIOs	Y	Y	Y	Y	-	6 pins	2 tamper
DMA	O	O	-	-	-	-	-
USARTx (x=1..8)	O	O	-	-	-	-	-
I2Cx (x=1,2,3)	O	O	-	-	-	-	-
SPIx (x=1..5)	O	O	-	-	-	-	-
SAIx (x=1,2)	O	O	-	-	-	-	-
ADCx (x=1,2,3)	O	O	-	-	-	-	-
DACx (x=1,2)	O	O	-	-	-	-	-
Temperature sensor	O	O	-	-	-	-	-
Timers (TIMx)	O	O	-	-	-	-	-
Low-power timer 1 (LPTIM1)	O	O	O	O	-	-	-
Independent watchdog (IWDG)	O	O	O	O	O	O	-
Window watchdog (WWDG)	O	O	-	-	-	-	-
Systick timer	O	O	-	-	-	-	-
Random number generator (RNG)	O	O	-	-	-	-	-
AES processor (AES)	O	O	-	-	-	-	-
SDMMC1/2	O	O	-	-	-	-	-
CAN1	O	O	-	-	-	-	-
USB OTG FS	O	O	-	O	-	-	-
USB OTG HS	O	O	-	O	-	-	-

1. Legend: Y = Yes (Enable). O = Optional (Disable by default. Can be enabled by software). - = Not available. Wakeup highlighted in gray.
2. Some peripherals with wakeup from Stop capability can request HSI to be enabled. In this case, HSI is woken up by the peripheral, and only feeds the peripheral which requested it. HSI is automatically put off when the peripheral does not need it anymore.

4.3.1 Debug mode

By default, the debug connection is lost when the device enters in Stop or Standby mode while the debug features are used. This is due to the fact that the Cortex®-M7 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 40.16.1: Debug support for low-power modes](#).

4.3.2 Run mode

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 5.3.3: RCC clock configuration register \(RCC_CFGR\)](#).

Peripheral clock gating

In Run mode, the HCLKx 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 AHB1 peripheral clock enable register (RCC_AHB1ENR), AHB2 peripheral clock enable register (RCC_AHB2ENR), AHB3 peripheral clock enable register (RCC_AHB3ENR) (see [Section 5.3.10: RCC AHB1 peripheral clock register \(RCC_AHB1ENR\)](#), [Section 5.3.11: RCC AHB2 peripheral clock enable register \(RCC_AHB2ENR\)](#), and [Section 5.3.12: RCC AHB3 peripheral clock enable register \(RCC_AHB3ENR\)](#)).

Disabling the peripherals clocks in Sleep mode can be performed automatically by resetting the corresponding bit in RCC_AHBxLPENR and RCC_APBxLPENR registers.

4.3.3 Low-power mode

Entering low-power mode

Low-power modes are entered by the MCU executing the WFI (Wait For Interrupt), or WFE (Wait For Event) instructions, or when the SLEEPONEXIT bit in the Cortex®-M7 System Control register is set on Return from ISR.

Exiting low-power mode

From Sleep and Stop modes the MCU exits low-power mode depending on the way the mode was entered:

- If the WFI instruction or Return from ISR was used to enter the low-power mode, any peripheral interrupt acknowledged by the NVIC can wake up the device
- If the WFE instruction was used to enter the low-power mode, the MCU exits the mode as soon as an event occurs. The wakeup event can be generated either by:
 - NVIC IRQ interrupt
 - When SEVEONPEND=0 in the Cortex®-M7 System Control register.

By enabling an interrupt in the peripheral control register and in the NVIC. When the MCU resumes from WFE, the peripheral interrupt pending bit and the NVIC

peripheral IRQ channel pending bit (in the NVIC interrupt clear pending register) have to be cleared.

Only NVIC interrupts with sufficient priority will wakeup and interrupt the MCU.

- When SEVEONPEND=1 in the Cortex®-M7 System Control register.

By enabling an interrupt in the peripheral control register and optionally in the NVIC. When the MCU resumes from WFE, the peripheral interrupt pending bit and (when enabled) the NVIC peripheral IRQ channel pending bit (in the NVIC interrupt clear pending register) have to be cleared.

All NVIC interrupts will wakeup the MCU, even the disabled ones.

Only enabled NVIC interrupts with sufficient priority will wakeup and interrupt the MCU.

- Event

Configuring a EXTI line in event mode. When the CPU resumes from WFE, it is not necessary to clear the EXTI peripheral interrupt pending bit or the NVIC IRQ channel pending bit as the pending bits corresponding to the event line is not set. It may be necessary to clear the interrupt flag in the peripheral.

From Standby mode the MCU exits Low-power mode through an external reset (NRST pin), an IWDG reset, a rising edge on one of the enabled WKUPx pins or a RTC event (see [Figure 286: RTC block diagram](#)).

4.3.4 Sleep mode

I/O states in Sleep mode

In Sleep mode, all I/O pins keep the same state as in Run 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®-M7 System Control register:

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

Refer to [Table 16](#) and [Table 17](#) for details on how to enter Sleep mode.

Exiting Sleep mode

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

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

- Enabling an interrupt in the peripheral control register but not in the NVIC, and enabling the SEVONPEND bit in the Cortex®-M7 System Control register. When the MCU resumes from WFE, the peripheral interrupt pending bit and the peripheral NVIC IRQ channel pending bit (in the NVIC interrupt clear pending register) have to be cleared.
- Or configuring an external or internal EXTI line in event mode. When the CPU resumes from WFE, it is not necessary to clear the peripheral interrupt pending bit or the NVIC IRQ channel pending bit as the pending bit corresponding to the event line is not set.

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

Refer to [Table 16](#) and [Table 17](#) for more details on how to exit Sleep mode.

Table 16. Sleep-now entry and exit

Sleep-now mode	Description
Mode entry	WFI (Wait for Interrupt) or WFE (Wait for Event) while: – SLEEPDEEP = 0, and – No interrupt (for WFI) or event (for WFE) is pending. Refer to the Cortex®-M7 System Control register.
	On Return from ISR while: – SLEEPDEEP = 0 and – SLEEPONEXIT = 1, – No interrupt is pending. Refer to the Cortex®-M7 System Control register.
Mode exit	If WFI or Return from ISR was used for entry: Interrupt: Refer to Table 36: STM32F72xxx and STM32F73xxx vector table If WFE was used for entry and SEVONPEND = 0 Wakeup event: Refer to Section 11.3: Wakeup event management If WFE was used for entry and SEVONPEND = 1 Interrupt even when disabled in NVIC: refer to Table 36: STM32F72xxx and STM32F73xxx vector table and Wakeup event (see Section 11.3: Wakeup event management). Wakeup latency
Wakeup latency	None

Table 17. Sleep-on-exit entry and exit

Sleep-on-exit	Description
Mode entry	WFI (Wait for Interrupt) or WFE (Wait for Event) while: – SLEEPDEEP = 0, and – No interrupt (for WFI) or event (for WFE) is pending. Refer to the Cortex®-M7 System Control register.
	On Return from ISR while: – SLEEPDEEP = 0, and – SLEEPONEXIT = 1, and – No interrupt is pending. Refer to the Cortex®-M7 System Control register.
Mode exit	Interrupt: refer to Table 36: STM32F72xxx and STM32F73xxx vector table
Wakeup latency	None

4.3.5 Stop mode

The Stop mode is based on the Cortex®-M7 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 1.2 V domain are stopped, the PLLs, the HSI and the HSE RC oscillators are disabled. Internal SRAM and register contents are preserved.

In Stop mode, the power consumption can be further reduced by using additional settings in the PWR_CR1 register. However this will induce an additional startup delay when waking up from Stop mode (see [Table 18](#)).

Table 18. Stop operating modes

Voltage Regulator Mode		UDEN[1:0] bits	MRUDS bit	LPUDS bit	LPDS bit	FPDS bit	Wakeup latency
Normal mode	STOP MR (Main Regulator)	-	0	-	0	0	HSI RC startup time
	STOP MR-FPD	-	0	-	0	1	HSI RC startup time + Flash wakeup time from power-down mode
	STOP LP	-	0	0	1	0	HSI RC startup time + regulator wakeup time from LP mode
	STOP LP-FPD	-	-	0	1	1	HSI RC startup time + Flash wakeup time from power-down mode + regulator wakeup time from LP mode
Under-drive Mode	STOP UMR-FPD	3	1	-	0	-	HSI RC startup time + Flash wakeup time from power-down mode + Main regulator wakeup time from under-drive mode + Core logic to nominal mode
	STOP ULP-FPD	3	-	1	1	-	HSI RC startup time + Flash wakeup time from power-down mode + regulator wakeup time from LP under-drive mode + Core logic to nominal mode

I/O states in Stop mode

In stop mode, all I/Os pins keep the same state as in the run mode

Entering Stop mode

The Stop mode is entered according to *Entering low-power mode*, when the SLEEPDEEP bit in Cortex®-M7 System Control register is set.

Refer to [Table 19](#) for details on how to enter the Stop mode.

When the microcontroller enters in Stop mode, the voltage scale 3 is automatically selected. To further reduce power consumption in Stop mode, the internal voltage regulator can be put in low-power or low voltage mode. This is configured by the LPDS, MRUDS, LPUDS and UDEN bits of the [PWR power control register \(PWR_CR1\)](#).

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

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

If the Over-drive mode was enabled before entering Stop mode, it is automatically disabled during when the Stop mode is activated.

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

- Independent watchdog (IWDG): the IWDG is started by writing to its Key register or by hardware option. Once started it cannot be stopped except by a Reset. See [Section 23.3 in Section 23: Independent watchdog \(IWDG\)](#).
- Real-time clock (RTC): this is configured by the RTCEN bit in the [RCC backup domain control register \(RCC_BDCR\)](#)
- Internal RC oscillator (LSI RC): this is configured by the LSION bit in the [RCC clock control & status register \(RCC_CSR\)](#).
- External 32.768 kHz oscillator (LSE OSC): this is configured by the LSEON bit in the [RCC backup domain control register \(RCC_BDCR\)](#).

The ADC or DAC can also consume power during the Stop mode, unless they are disabled before entering it. To disable them, the ADON bit in the ADC_CR2 register and the ENx bit in the DAC_CR register must both be written to 0.

Note:

Before entering Stop mode, it is recommended to enable the clock security system (CSS) feature to prevent external oscillator (HSE) failure from impacting the internal MCU behavior.

Exiting Stop mode

Refer to [Table 19](#) 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.

If the Under-drive mode was enabled, it is automatically disabled after exiting Stop mode.

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.

When the voltage regulator operates in Under-drive mode, an additional startup delay is induced when waking up from Stop mode.

Table 19. Stop mode entry and exit (STM32F72xxx and STM32F73xxx)

Stop mode	Description
Mode entry	<p>WFI (Wait for Interrupt) or WFE (Wait for Event) while:</p> <ul style="list-style-type: none"> – No interrupt (for WFI) or event (for WFE) is pending. – SLEEPDEEP bit is set in Cortex®-M7 System Control register, – PDSS bit is cleared in Power Control register (PWR_CR1), – Select the voltage regulator mode by configuring LPDS, MRUDS, LPUDS and UDEN bits in PWR_CR (see Table 18: Stop operating modes).
	<p>On Return from ISR while:</p> <ul style="list-style-type: none"> – No interrupt is pending, – SLEEPDEEP bit is set in Cortex®-M7 System Control register, and – SLEEPONEXIT = 1, and – PDSS is cleared in Power Control register (PWR_CR1).
	<p><i>Note: To enter Stop mode, all EXTI Line pending bits (in Pending register (EXTI_PR)), all peripheral interrupts pending bits, the RTC Alarm (Alarm A and Alarm B), RTC wakeup, RTC tamper, and RTC time stamp flags, must be reset. Otherwise, the Stop mode entry procedure is ignored and program execution continues.</i></p>
Mode exit	<p>If WFI or Return from ISR was used for entry:</p> <p>All EXTI lines configured in Interrupt mode (the corresponding EXTI Interrupt vector must be enabled in the NVIC). The interrupt source can be external interrupts or peripherals with wakeup capability. Refer to Table 36: STM32F72xxx and STM32F73xxx vector table on page 252.</p> <p>If WFE was used for entry and SEVONPEND = 0:</p> <p>All EXTI Lines configured in event mode. Refer to Section 11.3: Wakeup event management on page 295</p> <p>If WFE was used for entry and SEVONPEND = 1:</p> <ul style="list-style-type: none"> – Any EXTI lines configured in Interrupt mode (even if the corresponding EXTI Interrupt vector is disabled in the NVIC). The interrupt source can be external interrupts or peripherals with wakeup capability. Refer to Table 36: STM32F72xxx and STM32F73xxx vector table on page 252. – Wakeup event: refer to Section 11.3: Wakeup event management on page 295.
Wakeup latency	Refer to Table 18: Stop operating modes

4.3.6 Standby mode

The Standby mode allows to achieve the lowest power consumption. It is based on the Cortex®-M7 deepsleep mode, with the voltage regulator disabled. The 1.2 V domain is consequently powered off. The PLLs, the HSI oscillator and the HSE oscillator are also switched off. SRAM and register contents are lost except for registers in the backup domain (RTC registers, RTC backup register and backup SRAM), and Standby circuitry (see [Figure 5](#)).

Entering Standby mode

The Standby mode is entered according to Entering low-power mode, when the SLEEPDEEP bit in the Cortex®-M7 System Control register is set.

Refer to [Table 20](#) for more details on how to enter Standby mode.

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

- Independent watchdog (IWDG): the IWDG is started by writing to its Key register or by hardware option. Once started it cannot be stopped except by a reset. See [Section 23.3 in Section 23: 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), an IWDG Reset, a rising or falling edge on WKUP pin, an RTC alarm, a tamper event, or a time stamp event is detected. All registers are reset after wakeup from Standby except for [PWR power control/status register \(PWR_CSR1\)](#).

After waking up from Standby mode, program execution restarts in the same way as after a Reset (boot pin sampling, vector reset is fetched, etc.). The SBF status flag in the [PWR power control/status register \(PWR_CSR1\)](#) indicates that the MCU was in Standby mode.

Refer to [Table 20](#) for more details on how to exit Standby mode.

Table 20. Standby mode entry and exit

Standby mode	Description
Mode entry	WFI (Wait for Interrupt) or WFE (Wait for Event) while: – SLEEPDEEP is set in Cortex®-M7 System Control register, – PDSS bit is set in Power Control register (PWR_CR1), – No interrupt (for WFI) or event (for WFE) is pending, – WUIF bit is cleared in Power Control and Status register (PWR_CSR1), – the RTC flag corresponding to the chosen wakeup source (RTC Alarm A, RTC Alarm B, RTC wakeup, Tamper or Timestamp flags) is cleared. – EIWUP bit is set in Power Control register (PWR_CSR1).
	On return from ISR while: – SLEEPDEEP bit is set in Cortex®-M7 System Control register, and – SLEEPONEXIT = 1, and – PDSS bit is set in Power Control register (PWR_CR1), and – No interrupt is pending, – WUIF bit is cleared in Power Control and Status register (PWR_CSR1), – The RTC flag corresponding to the chosen wakeup source (RTC Alarm A, RTC Alarm B, RTC wakeup, Tamper or Timestamp flags) is cleared.
Mode exit	WKUP pin rising or falling edge, RTC alarm (Alarm A and Alarm B), RTC wakeup, tamper event, time stamp event, external reset in NRST pin, IWDG reset.
Wakeup latency	Reset phase.

I/O states in Standby mode

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

- Reset pad (still available)
- PC13 if configured for tamper, time stamp, RTC Alarm out, or RTC clock calibration out
- WKUP pins (PA0/PA2/PC1/PC13/PI8/PI11), if enabled

4.3.7 Programming the RTC alternate functions to wake up the device from the Stop and Standby modes

The MCU can be woken up from a low-power mode by an RTC alternate function.

The RTC alternate functions are the RTC alarms (Alarm A and Alarm B), RTC wakeup, RTC tamper event detection and RTC time stamp event detection.

These RTC alternate functions can wake up the system from the Stop and Standby low-power modes.

The system can also wake up from low-power modes without depending on an external interrupt (Auto-wakeup mode), by using the RTC alarm or the RTC wakeup events.

The RTC provides a programmable time base for waking up from the Stop or Standby mode at regular intervals.

For this purpose, two of the three alternate RTC clock sources can be selected by programming the RTCSEL[1:0] bits in the *RCC 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 a very low-power consumption (additional consumption of less than 1 μ A under 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 use minimum power.

RTC alternate functions to wake up the device from the Stop mode

- To wake up the device from the Stop mode with an RTC alarm event, it is necessary to:
 - a) Configure the EXTI Line 17 to be sensitive to rising edges (Interrupt or Event modes)
 - b) Enable the RTC Alarm Interrupt in the RTC_CR register
 - c) Configure the RTC to generate the RTC alarm
- To wake up the device from the Stop mode with an RTC tamper or time stamp event, it is necessary to:
 - a) Configure the EXTI Line 21 to be sensitive to rising edges (Interrupt or Event modes)
 - b) Enable the RTC time stamp Interrupt in the RTC_CR register or the RTC tamper interrupt in the RTC_TAFCR register
 - c) Configure the RTC to detect the tamper or time stamp event
- To wake up the device from the Stop mode with an RTC wakeup event, it is necessary to:
 - a) Configure the EXTI Line 22 to be sensitive to rising edges (Interrupt or Event modes)
 - b) Enable the RTC wakeup interrupt in the RTC_CR register
 - c) Configure the RTC to generate the RTC Wakeup event

RTC alternate functions to wake up the device from the Standby mode

- To wake up the device from the Standby mode with an RTC alarm event, it is necessary to:
 - a) Enable the RTC alarm interrupt in the RTC_CR register
 - b) Configure the RTC to generate the RTC alarm
 - c) Enable the RTC internal wakeup event from standby by setting the EIWUP bit in the PWR_CSR1 register
- To wake up the device from the Standby mode with an RTC tamper or time stamp event, it is necessary to:
 - a) Enable the RTC time stamp interrupt in the RTC_CR register or the RTC tamper interrupt in the RTC_TAFCR register
 - b) Configure the RTC to detect the tamper or time stamp event
 - c) Enable the RTC internal wakeup event from standby by setting the EIWUP bit in the PWR_CSR1 register
- To wake up the device from the Standby mode with an RTC wakeup event, it is necessary to:
 - a) Enable the RTC wakeup interrupt in the RTC_CR register
 - b) Configure the RTC to generate the RTC wakeup event
 - c) Enable the RTC internal wakeup event from standby by setting the EIWUP bit in the PWR_CSR1 register

Safe RTC alternate function wakeup flag clearing sequence

To avoid bouncing on the pins onto which the RTC alternate functions are mapped, and exit correctly from the Stop and Standby modes, it is recommended to follow the sequence below before entering the Standby mode:

- When using RTC alarm to wake up the device from the low-power modes:
 - a) Disable the RTC alarm interrupt (ALRAIE or ALRBIE bits in the RTC_CR register)
 - b) Clear the RTC alarm (ALRAF/ALRBF) flag
 - c) Enable the RTC alarm interrupt
 - d) Re-enter the low-power mode
- When using RTC wakeup to wake up the device from the low-power modes:
 - a) Disable the RTC Wakeup interrupt (WUTIE bit in the RTC_CR register)
 - b) Enable the RTC Wakeup interrupt
 - c) Re-enter the low-power mode
- When using RTC tamper to wake up the device from the low-power modes:
 - a) Disable the RTC tamper interrupt (TAMPIE bit in the RTC_TAFCR register)
 - b) Clear the Tamper (TAMP1F/TSF) flag
 - c) Enable the RTC tamper interrupt
 - d) Re-enter the low-power mode
- When using RTC time stamp to wake up the device from the low-power modes:
 - a) Disable the RTC time stamp interrupt (TSIE bit in RTC_CR)
 - b) Clear the RTC time stamp (TSF) flag
 - c) Enable the RTC TimeStamp interrupt
 - d) Re-enter the low-power mode

4.4 Power control registers

4.4.1 PWR power control register (PWR_CR1)

Address offset: 0x00

Reset value: 0x0000 C000 (reset by wakeup from Standby mode)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	UDEN[1:0]	ODSWE N	ODEN	
												rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VOS[1:0]	ADCDC1	Res.	MRUDS	LPUDS	FPDS	DBP	PLS[2:0]	PVDE	CSBF	Res.	PDDS	LPDS			
rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rc_w1		rw	rw

Bits 31:20 Reserved, must be kept at reset value.

Bits 19:18 **UDEN[1:0]**: Under-drive enable in stop mode

These bits are set by software. They allow to achieve a lower power consumption in Stop mode but with a longer wakeup time.

When set, the digital area has less leakage consumption when the device enters Stop mode.

00: Under-drive disable

01: Reserved

10: Reserved

11:Under-drive enable

Bit 17 **ODSWEN**: Over-drive switching enabled.

This bit is set by software. It is cleared automatically by hardware after exiting from Stop mode or when the ODEN bit is reset. When set, It is used to switch to Over-drive mode.

To set or reset the ODSWEN bit, the HSI or HSE must be selected as system clock.

The ODSWEN bit must only be set when the ODRDY flag is set to switch to Over-drive mode.

0: Over-drive switching disabled

1: Over-drive switching enabled

Note: On any over-drive switch (enabled or disabled), the system clock will be stalled during the internal voltage set up.

Bit 16 **ODEN**: Over-drive enable

This bit is set by software. It is cleared automatically by hardware after exiting from Stop mode. It is used to enabled the Over-drive mode in order to reach a higher frequency.

To set or reset the ODEN bit, the HSI or HSE must be selected as system clock. When the ODEN bit is set, the application must first wait for the Over-drive ready flag (ODRDY) to be set before setting the ODSWEN bit.

0: Over-drive disabled

1: Over-drive enabled

Bits 15:14 VOS[1:0]: Regulator voltage scaling output selection

These bits control the main internal voltage regulator output voltage to achieve a trade-off between performance and power consumption when the device does not operate at the maximum frequency (refer to the STM32F72xxx and STM32F73xxx datasheets for more details).

These bits can be modified only when the PLL is OFF. The new value programmed is active only when the PLL is ON. When the PLL is OFF, the voltage scale 3 is automatically selected.

00: Reserved (Scale 3 mode selected)

01: Scale 3 mode

10: Scale 2 mode

11: Scale 1 mode (reset value)

Bit 13 ADCDC1:

0: No effect.

1: Refer to AN4073 for details on how to use this bit.

Note: This bit can only be set when operating at supply voltage range 2.7 to 3.6V.

Bit 12 Reserved, must be kept at reset value.**Bit 11 MRUDS: Main regulator in deepsleep under-drive mode**

This bit is set and cleared by software.

0: Main regulator ON when the device is in Stop mode

1: Main Regulator in under-drive mode and Flash memory in power-down when the device is in Stop under-drive mode.

Bit 10 LPUDS: Low-power regulator in deepsleep under-drive mode

This bit is set and cleared by software.

0: Low-power regulator ON if LPDS bit is set when the device is in Stop mode

1: Low-power regulator in under-drive mode if LPDS bit is set and Flash memory in power-down when the device is in Stop under-drive mode.

Bit 9 FPDS: Flash power-down in Stop mode

When set, the Flash memory enters power-down mode when the device enters Stop mode. This allows to achieve a lower consumption in stop mode but a longer restart time.

0: Flash memory not in power-down when the device is in Stop mode

1: Flash memory in power-down when the device is in Stop mode

Bit 8 DBP: Disable backup domain write protection

In reset state, the RCC_BDCR register, the RTC registers (including the backup registers), and the BRE bit of the PWR_CSR1 register, are protected against parasitic write access. This bit must be set to enable write access to these registers.

0: Access to RTC and RTC Backup registers and backup SRAM disabled

1: Access to RTC and RTC Backup registers and backup SRAM 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: 2.0 V

001: 2.1 V

010: 2.3 V

011: 2.5 V

100: 2.6 V

101: 2.7 V

110: 2.8 V

111: 2.9 V

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 SDF Standby Flag (write)

Bit 2 Reserved, must be kept at reset value**Bit 1 PDSS: 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 PDSS bit.

0: Main voltage regulator ON during Stop mode

1: Low-power voltage regulator ON during Stop mode

4.4.2 PWR power control/status register (PWR_CSR1)

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
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	UDRDY[1:0]	ODSWRDY	ODRDY	
												rc_w1	rc_w1	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	VOS RDY	Res.	Res.	Res.	Res.	BRE	EIWUP	Res.	Res.	Res.	Res.	BRR	PVDO	SBF	WUIF
	r					rw	rw					r	r	r	r

Bits 31:20 Reserved, must be kept at reset value.

Bits 19:18 **UDRDY[1:0]**: Under-drive ready flag

These bits are set by hardware when MCU enters stop Under-drive mode and exits. When the Under-drive mode is enabled, these bits are not set as long as the MCU has not entered stop mode yet. they are cleared by programming them to 1.

00: Under-drive is disabled

01: Reserved

10: Reserved

11:Under-drive mode is activated in Stop mode.

Bit 17 **ODSWRDY**: Over-drive mode switching ready

0: Over-drive mode is not active.

1: Over-drive mode is active on digital area on 1.2 V domain

Bit 16 **ODRDY**: Over-drive mode ready

0: Over-drive mode not ready.

1: Over-drive mode ready

Bit 14 **VOSRDY**: Regulator voltage scaling output selection ready bit

0: Not ready

1: Ready

Bits 13:10 Reserved, must be kept at reset value.

Bit 9 **BRE**: Backup regulator enable

When set, the Backup regulator (used to maintain backup SRAM content in Standby and V_{BAT} modes) is enabled. If BRE is reset, the backup regulator is switched off. The backup SRAM can still be used but its content will be lost in the Standby and V_{BAT} modes. Once set, the application must wait that the Backup Regulator Ready flag (BRR) is set to indicate that the data written into the RAM will be maintained in the Standby and V_{BAT} modes.

0: Backup regulator disabled

1: Backup regulator enabled

Note: This bit is not reset when the device wakes up from Standby mode, by a system reset, or by a power reset.

Bit 8 **EIWUP**: Enable internal wakeup

This bit must be set when RTC events (Alarm A or Alarm B, RTC Tamper, RTC TimeStamp or RTC Wakeup time) are used to wake up the system from Standby mode.

This bit is always read as 0.

0: Disable internal wakeup sources (RTC events) during Standby mode

1: Enable internal wakeup sources (RTC events) during Standby mode

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

Bit 3 **BRR**: Backup regulator ready

Set by hardware to indicate that the Backup Regulator is ready.

0: Backup Regulator not ready

1: Backup Regulator ready

Note: This bit is not reset when the device wakes up from Standby mode or by a system reset or power reset.

Bit 2 PVD0: 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 [PWR power control register \(PWR_CR1\)](#)

0: Device has not been in Standby mode

1: Device has been in Standby mode

Bit 0 WUIF: Wakeup internal flag

This bit is set when a wakeup is detected on the internal wakeup line in standby mode. It is cleared when all internal wakeup sources are cleared.

0: No wakeup internal event occurred

1: A wakeup event was detected from the RTC alarm (Alarm A or Alarm B), RTC Tamper event, RTC TimeStamp event or RTC Wakeup

4.4.3 PWR power control/status register 2 (PWR_CR2)

Address offset: 0x08

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	WUPP6	WUPP5	WUPP4	WUPP3	WUPP2	WUPP1	Res.	Res.	CWUPF6	CWUPF5	CWUPF4	CWUPF3	CWUPF2	CWUPF1
		rw	rw	rw	rw	rw	rw			r	r	r	r	r	r

Bits 31:14 Reserved, always read as 0.

Bit 13 WUPP6: Wakeup pin polarity bit for PI11

These bits define the polarity used for event detection on external wake-up pin PI11.

0: Detection on rising edge

1: Detection on falling edge

Bit 12 WUPP5: Wakeup pin polarity bit for PI8

These bits define the polarity used for event detection on external wake-up pin PI8.

0: Detection on rising edge

1: Detection on falling edge

Bit 11 WUPP4: Wakeup pin polarity bit for PC13

These bits define the polarity used for event detection on external wake-up pin PC13.

0: Detection on rising edge

1: Detection on falling edge

Bit 10 WUPP3: Wakeup pin polarity bit for PC1

These bits define the polarity used for event detection on external wake-up pin PC1.

0: Detection on rising edge

1: Detection on falling edge

Bit 9 **WUPP2**: Wakeup pin polarity bit for PA2

These bits define the polarity used for event detection on external wake-up pin PA2.

0: Detection on rising edge

1: Detection on falling edge

Bit 8 **WUPP1**: Wakeup pin polarity bit for PA0

These bits define the polarity used for event detection on external wake-up pin PA0.

0: Detection on rising edge

1: Detection on falling edge

Bits 7:6 Reserved, always read as 0

Bit 5 **CWUPF6**: Clear Wakeup Pin flag for PI11

These bits are always read as 0

0: No effect

1: Clear the WUPF Wakeup Pin flag after 2 System clock cycles.

Bit 4 **CWUPF5**: Clear Wakeup Pin flag for PI8

These bits are always read as 0

0: No effect

1: Clear the WUPF Wakeup Pin flag after 2 System clock cycles.

Bit 3 **CWUPF4**: Clear Wakeup Pin flag for PC13

These bits are always read as 0

0: No effect

1: Clear the WUPF Wakeup Pin flag after 2 System clock cycles.

Bit 2 **CWUPF3**: Clear Wakeup Pin flag for PC1

These bits are always read as 0

0: No effect

1: Clear the WUPF Wakeup Pin flag after 2 System clock cycles.

Bit 1 **CWUPF2**: Clear Wakeup Pin flag for PA2

These bits are always read as 0

0: No effect

1: Clear the WUPF Wakeup Pin flag after 2 System clock cycles.

Bit 0 **CWUPF1**: Clear Wakeup Pin flag for PA0

These bits are always read as 0

0: No effect

1: Clear the WUPF Wakeup Pin flag after 2 System clock cycles.

4.4.4 PWR power control register 2 (PWR_CSR2)

Address offset: 0x0C

Reset value: 0x0000 0000 (reset by wakeup from Standby mode, except wakeup flags which are reset by RESET pin)

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	EWUP6	EWUP5	EWUP4	EWUP3	EWUP2	EWUP1	Res.	Res.	WUPF6	WUPF5	WUPF4	WUPF3	WUPF2	WUPF1
		rw	rw	rw	rw	rw	rw			r	r	r	r	r	r

Bits 31:14 Reserved, always read as 0.

Bits 13 **EWUP6**: Enable Wakeup pin for PI11

This bit is set and cleared by software.

0: An event on WKUP pin PI11 does not wake-up the device from Standby mode.

1: A rising or falling edge on WKUP pin PI11 wakes-up the system from Standby mode.

Bit 12 **EWUP5**: Enable Wakeup pin for PI8

This bit is set and cleared by software.

0: An event on WKUP pin PI8 does not wake-up the device from Standby mode.

1: A rising or falling edge on WKUP pin PI8 wakes-up the system from Standby mode.

Bit 11 **EWUP4**: Enable Wakeup pin for PC13

This bit is set and cleared by software.

0: An event on WKUP pin PC13 does not wake-up the device from Standby mode.

1: A rising or falling edge on WKUP pin PC13 wakes-up the system from Standby mode.

Bit 10 **EWUP3**: Enable Wakeup pin for PC1

This bit is set and cleared by software.

0: An event on WKUP pin PC1 does not wake-up the device from Standby mode.

1: A rising or falling edge on WKUP pin PC1 wakes-up the system from Standby mode.

Bit 9 **EWUP2**: Enable Wakeup pin for PA2

This bit is set and cleared by software.

0: An event on WKUP pin PA2 does not wake-up the device from Standby mode.

1: A rising or falling edge on WKUP pin PA2 wakes-up the system from Standby mode.

Bit 8 **EWUP1**: Enable Wakeup pin for PA0

This bit is set and cleared by software.

0: An event on WKUP pin PA0 does not wake-up the device from Standby mode.

1: A rising or falling edge on WKUP pin PA0 wakes-up the system from Standby mode.

Bits 7:6 Reserved, always read as 0

Bit 5 WUPF6: Wakeup Pin flag for PI11

This bit is set by hardware and cleared only by a Reset pin or by setting the CWUPF6 bit in the PWR power control register 2 (PWR_CR2).

0: No Wakeup event occurred

1: A wakeup event is detected on WKUP PI11

Note: An additional wakeup event is detected if WKUP pin is enabled (by setting the EWUP6 bit) when WKUP pin PI11 level is already high.

Bit 4 WUPF5: Wakeup Pin flag for PI8

This bit is set by hardware and cleared only by a Reset pin or by setting the CWUPF5 bit in the PWR power control register 2 (PWR_CR2).

0: No Wakeup event occurred

1: A wakeup event is detected on WKUP PI8

Note: An additional wakeup event is detected if WKUP pin is enabled (by setting the EWUP5 bit) when WKUP pin PI8 level is already high.

Bit 3 WUPF4: Wakeup Pin flag for PC13

This bit is set by hardware and cleared only by a Reset pin or by setting the CWUPF4 bit in the PWR power control register 2 (PWR_CR2).

0: No Wakeup event occurred

1: A wakeup event is detected on WKUP PC13

Note: An additional wakeup event is detected if WKUP pin is enabled (by setting the EWUP4 bit) when WKUP pin PC13 level is already high.

Bit 2 WUPF3: Wakeup Pin flag for PC1

This bit is set by hardware and cleared only by a Reset pin or by setting the CWUPF3 bit in the PWR power control register 2 (PWR_CR2).

0: No Wakeup event occurred

1: A wakeup event is detected on WKUP PC1

Note: An additional wakeup event is detected if WKUP pin is enabled (by setting the EWUP3 bit) when WKUP pin PC1 level is already high.

Bit 1 WUPF2: Wakeup Pin flag for PA2

This bit is set by hardware and cleared only by a Reset pin or by setting the CWUPF2 bit in the PWR power control register 2 (PWR_CR2).

0: No Wakeup event occurred

1: A wakeup event is detected on WKUP PA2

Note: An additional wakeup event is detected if WKUP pin is enabled (by setting the EWUP2 bit) when WKUP pin PA2 level is already high.

Bit 0 WUPF1: Wakeup Pin flag for PA0

This bit is set by hardware and cleared only by a Reset pin or by setting the CWUPF1 bit in the PWR power control register 2 (PWR_CR2).

0: No Wakeup event occurred

1: A wakeup event is detected on WKUP PA0

Note: An additional wakeup event is detected if WKUP pin is enabled (by setting the EWUP1 bit) when WKUP pin PA0 level is already high.

4.5 PWR register map

The following table summarizes the PWR registers.

Table 21. PWR - register map and reset values

Offset	Register	Reset	31	30	29	28	27
0x000	PWR_CR1	Res.	Res.	Res.	Res.	Res.	Res.
	Reset value	Res.	Res.	Res.	Res.	Res.	Res.
0x004	PWR_CSR1	Res.	Res.	Res.	Res.	Res.	Res.
	Reset value	Res.	Res.	Res.	Res.	Res.	Res.
0x008	PWR_CR2	Res.	Res.	Res.	Res.	Res.	Res.
	Reset value	Res.	Res.	Res.	Res.	Res.	Res.
0x00C	PWR_CSR2	Res.	Res.	Res.	Res.	Res.	Res.
	Reset value	Res.	Res.	Res.	Res.	Res.	Res.
	o EWUPP6 o WUPP6	Res.	0	ADCDC1	13		
	o EWUPP5 o WUPP5	Res.	Res.	Res.	12		
	o EWUPP4 o WUPP4	Res.	Res.	MRUDS	11		
	o EWUPP3 o WUPP3	Res.	Res.	LPUDS	10		
	o EWUPP2 o WUPP2	0	BRE	0	FPDS	9	
	o EWUPP1 o WUPP1	0	EIWUP	0	DBP	8	
	Res.	Res.	Res.	0	PLS[2:0]	7	
	Res.	Res.	Res.	0		6	
	o WUPPF6 o CWUPPF6	Res.	0			5	
	o WUPPF5 o CWUPPF5	Res.	0			4	
	o WUPPF4 o CWUPPF4	0	BRR	0	CSBF	3	
	o WUPPF3 o CWUPPF3	0	PVDO	0	Res.	2	
	o WUPPF2 o CWUPPF2	0	SBF	0	PDDS	1	
	o WUPPF1 o CWUPPF1	0	WUJF	0	LPDS	0	

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

5 Reset and clock control (RCC)

5.1 Reset

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

5.1.1 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 13](#)).

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 [Software reset](#))
5. Low-power management reset (see [Low-power management reset](#))

Software reset

The reset source can be identified by checking the reset flags in the [RCC clock control & status register \(RCC_CSR\)](#).

The SYSRESETREQ bit in Cortex®-M7 Application Interrupt and Reset Control Register must be set to force a software reset on the device. Refer to the Cortex®-M7 technical reference manual for more details.

Low-power management reset

There are two ways of generating a low-power management reset:

1. Reset generated when entering the Standby mode:

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

2. Reset when entering the Stop mode:

This type of reset is enabled by resetting the nRST_STOP bit in the user option bytes. In this case, whenever a Stop mode entry sequence is successfully executed, the device is reset instead of entering the Stop mode.

5.1.2 Power reset

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

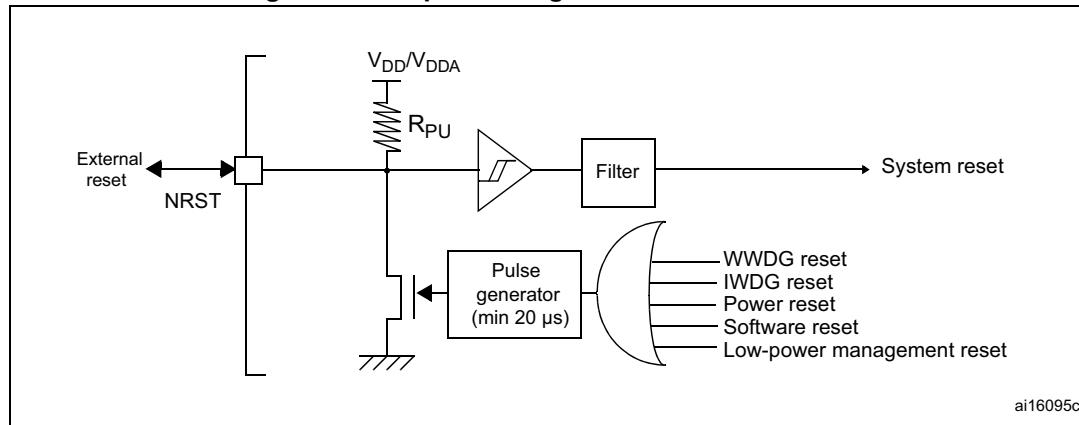
1. Power-on/power-down reset (POR/PDR reset) or brownout (BOR) reset
2. When exiting the Standby mode

A power reset sets all registers to their reset values except the Backup domain (see [Figure 13](#))

These sources act on the NRST pin and it is always kept low during the delay phase. The RESET service routine vector is fixed at address 0x0000_0004 in the memory map.

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

Figure 13. Simplified diagram of the reset circuit



The Backup domain has two specific resets that affect only the Backup domain (see [Figure 13](#)).

5.1.3 Backup domain reset

The backup domain reset sets all RTC registers and the RCC_BDCR register to their reset values. The BKPSRAM is not affected by this reset. The only way of resetting the BKPSRAM is through the Flash interface by requesting a protection level change from 1 to 0.

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

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

5.2 Clocks

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

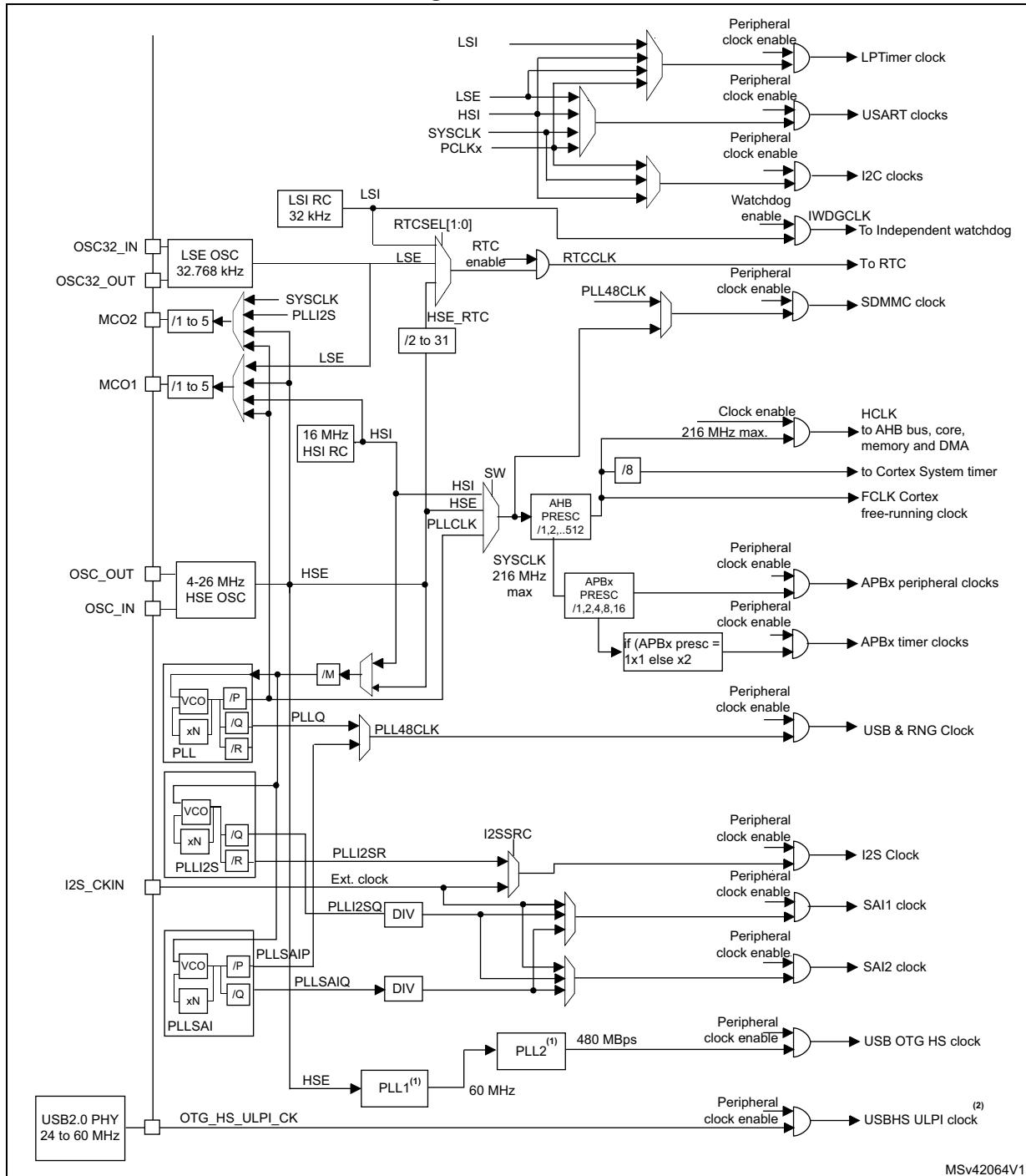
- HSI oscillator clock
- HSE oscillator clock
- Main PLL (PLL) clock

The devices have the two following secondary clock sources:

- 32 kHz low-speed internal RC (LSI RC) which drives the independent watchdog and, optionally, the RTC used for Auto-wakeup from the Stop/Standy mode.
- 32.768 kHz low-speed external crystal (LSE crystal) which optionally drives the RTC clock (RTCCLK)

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

Figure 14. Clock tree



1. PLL1 and PPL2 are embedded in the USB OTG PHY HS and configured in the USB OTG PHY HS controller (USBPHYC). Available only on the STM32F7x3xx devices. On STM32F730xx devices, available only on LQFP144 and UFBGA176 packages.
2. Available only on the STM32F7x2xx devices. On STM32F730xx devices, available only on LQFP64 and LQFP100 packages.
3. For full details about the internal and external clock source characteristics, refer to the Electrical characteristics section in the device datasheet.

4. When TIMPRE bit of the RCC_DKCFGR1 register is reset, if APBx prescaler is 1, then TIMxCLK = PCLKx, otherwise $\text{TIMxCLK} = 2 \times \text{PCLKx}$.
5. When TIMPRE bit in the RCC_DKCFGR1 register is set, if APBx prescaler is 1,2 or 4, then TIMxCLK = HCLK, otherwise $\text{TIMxCLK} = 4 \times \text{PCLKx}$.

The clock controller provides a high degree of flexibility to the application in the choice of the external crystal or the oscillator to run the core and peripherals at the highest frequency

All peripheral clocks are derived from their bus clock (HCLK,PLCK1, PCLK2) except for:

- The 48MHz clock, used for USB OTG FS, SDMMCs and RNG. This clock is derived from one of the following sources:
 - main PLL VCO (PLLQ Clock)
 - PLLSAI VCO (PLLSAI clock)
- The U(S)ARTs clocks which are derived from one of the following sources:
 - System clock (SYSCLK)
 - HSI clock
 - LSE clock
 - APB1 or APB2 clock (PCLK1 or PCLK2 depending on which APB is mapped the U(S)ART)
- The I2Cs clocks which are derived from one of the following sources:
 - System clock (SYSCLK)
 - HSI clock
 - APB1 clock (PCLK1)
- I2S clock
To achieve high-quality audio performance, the I2S clock can be derived either from a specific PLL (PLLI2S) or from an external clock mapped on the I2S_CKIN pin. For more information about I2S clock frequency and precision, refer to [Section 28.7.5: Clock generator](#).
- The SAI1 and SAI2 clocks which are derived from one of the following sources:
 - PLLSAI VCO (PLLSAIQ)
 - PLLI2S VCO (PLLI2SQ)
 - External clock mapped on the I2S_CKIN pin.
- The low-power timer (LPTIM1) clock which is derived from one of the following sources:
 - LSI clock
 - LSE clock
 - HSI clock
 - APB1 clock (PCLK1)
 - External clock mapped on LPTIM1_IN1
- The USB OTG HS (60 MHz) clock which is provided from the external PHY

- The RTC clock which is derived from one of the following sources:
 - LSE clock
 - LSI clock
 - HSE clock divided by 32
- The IWDG clock which is always the LSI clock.
- The timer clock frequencies are automatically set by hardware. There are two cases depending on the value of TIMPRE bit in RCC_CFGR register:
 - If TIMPRE bit in RCC_DKCFGR1 register is reset:
If the APB prescaler is configured to a division factor of 1, the timer clock frequencies (TIMxCLK) are set to PCLKx. Otherwise, the timer clock frequencies are twice the frequency of the APB domain to which the timers are connected: $TIMxCLK = 2 \times PCLKx$.
 - If TIMPRE bit in RCC_DKCFGR1 register is set:
If the APB prescaler is configured to a division factor of 1, 2 or 4, the timer clock frequencies (TIMxCLK) are set to HCLK. Otherwise, the timer clock frequencies are four times the frequency of the APB domain to which the timers are connected: $TIMxCLK = 4 \times PCLKx$.

The RCC feeds the external clock of the Cortex System Timer (SysTick) with the AHB clock (HCLK) divided by 8. The SysTick can work either with this clock or with the Cortex clock (HCLK), configurable in the SysTick control and status register.

FCLK acts as Cortex®-M7 free-running clock. For more details, refer to the Cortex®-M7 technical reference manual.

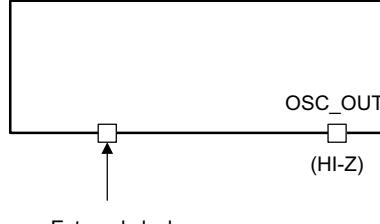
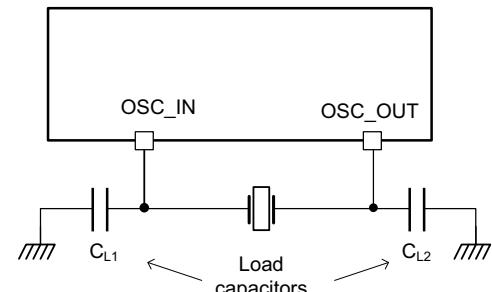
5.2.1 HSE clock

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

- HSE external crystal/ceramic resonator
- HSE external user 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 15. HSE/ LSE clock sources

Hardware configuration	
External clock	 <p>External clock source</p> <p>OSC_OUT (HI-Z)</p> <p>MSv42094V1</p>
Crystal/ceramic resonators	 <p>OSC_IN OSC_OUT</p> <p>C_{L1} Load capacitors C_{L2}</p> <p>MSv42081V1</p>

External source (HSE bypass)

In this mode, an external clock source must be provided. You select this mode by setting the HSEBYP and HSEON bits in the [RCC 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 15](#).

External crystal/ceramic resonator (HSE crystal)

The HSE has the advantage of producing a very accurate rate on the main clock.

The associated hardware configuration is shown in [Figure 15](#). Refer to the electrical characteristics section of the *datasheet* for more details.

The HSERDY flag in the [RCC 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 [RCC clock interrupt register \(RCC_CIR\)](#).

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

5.2.2 HSI clock

The HSI clock signal is generated from an internal 16 MHz RC oscillator and can be used directly as a system clock, or 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 $T_A = 25\text{ }^{\circ}\text{C}$.

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

The HSIRDY flag in the [*RCC 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 [*RCC 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 5.2.7: Clock security system \(CSS\) on page 135*](#).

5.2.3 PLL

The devices feature three PLLs:

- A main PLL (PLL) clocked by the HSE or HSI oscillator and featuring two different output clocks:
 - The first output is used to generate the high speed system clock (up to 216 MHz)
 - The second output is used to generate 48MHz clock for the USB OTG FS, SDMMCs and RNG.
- PLLI2S is used to generate an accurate clock to achieve high-quality audio performance on the I2S and SAIs interfaces.
- PLLSAI is used to generate clock for SAIs interfaces and the 48MHz (PLLSAI48CLK) that can be selected for the USB OTG FS, SDMMCs and RNG.

Since the main-PLL configuration parameters cannot be changed once PLL is enabled, it is recommended to configure PLL before enabling it (selection of the HSI or HSE oscillator as PLL clock source, and configuration of division factors M, N, P, and Q).

The PLLI2S and PLLSAI use the same input clock as PLL (PLLM[5:0] and PLLSRC bits are common to both PLLs). However, the PLLI2S and PLLSAI have dedicated enable/disable and division factors (N and R) configuration bits. Once the PLLI2S and PLLSAI are enabled, the configuration parameters cannot be changed.

The three PLLs are disabled by hardware when entering Stop and Standby modes, or when an HSE failure occurs when HSE or PLL (clocked by HSE) are used as system clock. [*RCC PLL configuration register \(RCC_PLLCFGR\)*](#), [*RCC clock configuration register \(RCC_CFGR\)*](#), and [*RCC dedicated clocks configuration register \(RCC_DKCFGREG1\)*](#) can be used to configure PLL, PLLI2S, and PLLSAI.

5.2.4 LSE clock

The LSE clock is generated from 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 oscillator is switched on and off using the LSEON bit in [RCC backup domain control register \(RCC_BDCR\)](#).

The LSERDY flag in the [RCC 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 [RCC clock interrupt register \(RCC_CIR\)](#).

External source (LSE bypass)

In this mode, an external clock source must be provided. It must have a frequency up to 1 MHz. You select this mode by setting the LSEBYP and LSEON bits in the [RCC 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 15](#).

5.2.5 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 32 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 [RCC clock control & status register \(RCC_CSR\)](#).

The LSIRDY flag in the [RCC clock 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 [RCC clock interrupt register \(RCC_CIR\)](#).

5.2.6 System clock (SYSCLK) selection

After a system reset, the HSI oscillator is selected as the system clock. When a clock source is used directly or through PLL as the 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 that is not yet ready is selected, the switch occurs when the clock source is ready. Status bits in the [RCC clock control register \(RCC_CR\)](#) indicate which clock(s) is (are) ready and which clock is currently used as the system clock.

5.2.7 Clock security system (CSS)

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

If a failure is detected on the HSE clock, this oscillator is automatically disabled, a clock failure event is sent to the break inputs of advanced-control timers TIM1 and TIM8, 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®-M7 NMI (non-maskable interrupt) exception vector.

Note: When the CSS is enabled, if the HSE clock happens to fail, the CSS generates an interrupt, which causes the automatic generation of an NMI. The NMI is executed indefinitely unless the CSS interrupt pending bit is cleared. As a consequence, the application has to clear the CSS interrupt in the NMI ISR 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 meaning that it is directly used as PLL input clock, and that PLL clock is the system clock) and a failure is detected, then the system clock switches to the HSI oscillator and the HSE oscillator is disabled.

If the HSE oscillator clock was the clock source of PLL used as the system clock when the failure occurred, PLL is also disabled. In this case, if the PLLI2S or PLLSAI was enabled, it is also disabled when the HSE fails.

5.2.8 RTC/AWU clock

Once the RTCCLK clock source has been selected, the only possible way of modifying the selection is to reset the power domain.

The RTCCLK clock source can be either the HSE 1 MHz (HSE divided by a programmable prescaler), the LSE or the LSI clock. This is selected by programming the RTCSEL[1:0] bits in the [RCC backup domain control register \(RCC_BDCR\)](#) and the RTCPRE[4:0] bits in [RCC clock configuration register \(RCC_CFGR\)](#). This selection cannot be modified without resetting the Backup domain.

If the LSE is selected as the RTC clock, the RTC will work normally if the backup or the system supply disappears. If the LSI is selected as the AWU clock, the AWU state is not guaranteed if the system supply disappears. If the HSE oscillator divided by a value between 2 and 31 is used as the RTC clock, the RTC state is not guaranteed if the backup or the system supply disappears.

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

- If LSE is selected as the RTC clock:
 - The RTC continues to work even if the V_{DD} supply is switched off, provided the V_{BAT} supply is maintained.
 - The RTC remains clocked and functional under system reset.
- If LSI is selected as the Auto-wakeup unit (AWU) clock:
 - The AWU state is not guaranteed if the V_{DD} supply is powered off. Refer to [Section 5.2.5: LSI clock on page 135](#) for more details on LSI calibration.
- If the HSE clock is used as the RTC clock:
 - The RTC state is not guaranteed if the V_{DD} supply is powered off or if the internal voltage regulator is powered off (removing power from the 1.2 V domain) and also when entering in Stop mode

Note: To read the RTC calendar register when the APB1 clock frequency is less than seven times the RTC clock frequency ($f_{APB1} < 7 \times f_{RTCCLK}$), the software must read the calendar time and date registers twice. The data are correct if the second read access to RTC_TR gives the same result than the first one. Otherwise a third read access must be performed.

5.2.9 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.

5.2.10 Clock-out capability

Two microcontroller clock output (MCO) pins are available:

- MCO1

You can output four different clock sources onto the MCO1 pin (PA8) using the configurable prescaler (from 1 to 5):

- HSI clock
- LSE clock
- HSE clock
- PLL clock

The desired clock source is selected using the MCO1PRE[2:0] and MCO1[1:0] bits in the *RCC clock configuration register (RCC_CFRG)*.

- MCO2

You can output four different clock sources onto the MCO2 pin (PC9) using the configurable prescaler (from 1 to 5):

- HSE clock
- PLL clock
- System clock (SYSCLK)
- PLLI2S clock

The desired clock source is selected using the MCO2PRE[2:0] and MCO2 bits in the *RCC clock configuration register (RCC_CFRG)*.

For the different MCO pins, the corresponding GPIO port has to be programmed in alternate function mode.

5.2.11 Internal/external clock measurement using TIM5/TIM11

It is possible to indirectly measure the frequencies of all on-board clock source generators by means of the input capture of TIM5 channel4 and TIM11 channel1 as shown in *Figure 16* and *Figure 17*.

Internal/external clock measurement using TIM5 channel4

TIM5 has an input multiplexer which allows choosing whether the input capture is triggered by the I/O or by an internal clock. This selection is performed through the TI4_RMP [1:0] bits in the TIM5_OR register.

The primary purpose of having the LSE connected to the channel4 input capture is to be able to precisely measure the HSI (this requires to have the HSI used as the system clock source). The number of HSI clock counts between consecutive edges of the LSE signal provides a measurement of the internal clock period. Taking advantage of the high precision of LSE crystals (typically a few tens of ppm) we can determine the internal clock frequency with the same resolution, and trim the source to compensate for manufacturing-process and/or temperature- and voltage-related frequency deviations.

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

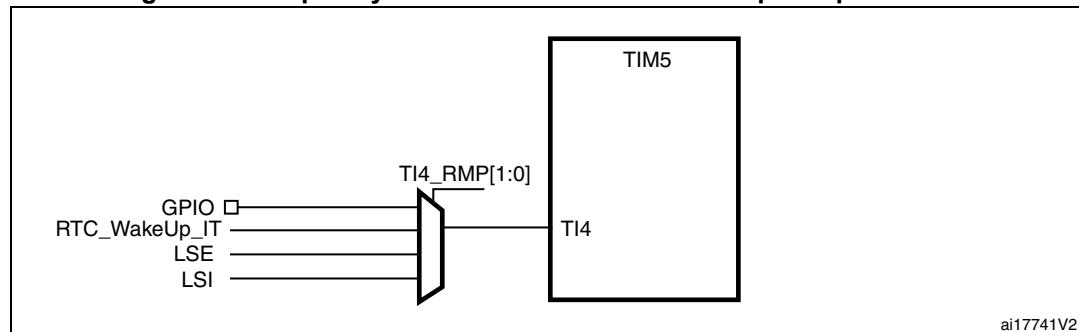
The basic concept consists in providing a relative measurement (e.g. HSI/LSE ratio): the precision is therefore tightly linked to the ratio between the two clock sources. The greater the ratio, the better the measurement.

It is also possible to measure the LSI frequency: this is useful for applications that do not have a crystal. The ultra-low-power LSI oscillator has a large manufacturing process deviation: by measuring it versus the HSI clock source, it is possible to determine its frequency with the precision of the HSI. The measured value can be used to have more accurate RTC time base timeouts (when LSI is used as the RTC clock source) and/or an IWDG timeout with an acceptable accuracy.

Use the following procedure to measure the LSI frequency:

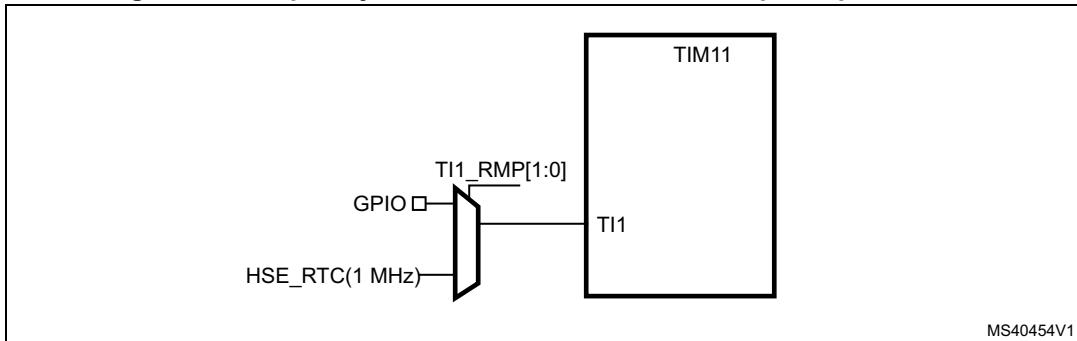
1. Enable the TIM5 timer and configure channel4 in Input capture mode.
2. This bit is set the TI4_RMP bits in the TIM5_OR register to 0x01 to connect the LSI clock internally to TIM5 channel4 input capture for calibration purposes.
3. Measure the LSI clock frequency using the TIM5 capture/compare 4 event or interrupt.
4. Use the measured LSI frequency to update the prescaler of the RTC depending on the desired time base and/or to compute the IWDG timeout.

Figure 16. Frequency measurement with TIM5 in Input capture mode



Internal/external clock measurement using TIM11 channel1

TIM11 has an input multiplexer which allows choosing whether the input capture is triggered by the I/O or by an internal clock. This selection is performed through TI1_RMP [1:0] bits in the TIM11_OR register. The HSE_RTC clock (HSE divided by a programmable prescaler) is connected to channel 1 input capture to have a rough indication of the external crystal frequency. This requires that the HSI is the system clock source. This can be useful for instance to ensure compliance with the IEC 60730/IEC 61335 standards which require to be able to determine harmonic or subharmonic frequencies (-50/+100% deviations).

Figure 17. Frequency measurement with TIM11 in Input capture mode

5.2.12 Peripheral clock enable register (RCC_AHBxENR, RCC_APBxENRy)

Each peripheral clock can be enabled by the xxxxEN bit of the RCC_AHBxENR or RCC_APBxENRy registers.

When the peripheral clock is not active, the peripheral registers read or write accesses are not supported. The peripheral enable bit has a synchronization mechanism to create a glitch free clock for the peripheral.

After the enable bit is set, there is a 2 peripheral clock cycles delay before the clock being active.

Caution: Just after enabling the clock for a peripheral, software must wait for a 2 peripheral clock cycles delay before accessing the peripheral registers.

5.3 RCC registers

Refer to [Section 1.2: List of abbreviations for registers](#) for a list of abbreviations used in register descriptions.

5.3.1 RCC clock control register (RCC_CR)

Address offset: 0x000

Reset value: 0x0000 XX83 where X is undefined.

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	PLLSAI RDY	PLLSAI ON	PLL12S RDY	PLL12S ON	PLL1RD Y	PLLON	Res.	Res.	Res.	Res.	CSS ON	HSE BYP	HSE RDY	HSE ON
		r	rw	r	rw	r	rw					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
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw		r	rw

Bits 31:28 Reserved, must be kept at reset value.

Bit 29 **PLLSAIRDY**: PLLSAI clock ready flag

Set by hardware to indicate that the PLLSAI is locked.

0: PLLSAI unlocked

1: PLLSAI locked

Bit 28 **PLLSAION**: PLLSAI enable

Set and cleared by software to enable PLLSAI.

Cleared by hardware when entering Stop or Standby mode.

0: PLLSAI OFF

1: PLLSAI ON

Bit 27 **PLL2SRDY**: PLLI2S clock ready flag

Set by hardware to indicate that the PLLI2S is locked.

0: PLLI2S unlocked

1: PLLI2S locked

Bit 26 **PLL2SON**: PLLI2S enable

Set and cleared by software to enable PLLI2S.

Cleared by hardware when entering Stop or Standby mode.

0: PLLI2S OFF

1: PLLI2S ON

Bit 25 **PLLRDY**: Main PLL (PLL) clock ready flag

Set by hardware to indicate that PLL is locked.

0: PLL unlocked

1: PLL locked

Bit 24 **PLLON**: Main PLL (PLL) enable

Set and cleared by software to enable PLL.

Cleared by hardware when entering Stop or Standby mode. This bit cannot be reset if PLL clock is used as the system clock.

0: PLL OFF

1: PLL ON

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

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

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

0: Clock security system OFF (Clock detector OFF)

1: Clock security system ON (Clock detector ON if HSE oscillator is stable, OFF if not)

Bit 18 **HSEBYP**: HSE clock bypass

Set and cleared by software to bypass the oscillator with an external clock. The external clock must be enabled with the HSEON bit, to be used by the device.

The HSEBYP bit can be written only if the HSE oscillator is disabled.

0: HSE oscillator not bypassed

1: HSE oscillator bypassed with an external clock

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

Set by hardware to indicate that the HSE oscillator is stable. After the HSEON bit is cleared, HSERDY goes low after 6 HSE oscillator clock cycles.

0: HSE oscillator not ready

1: HSE oscillator ready

Bit 16 HSEON: HSE clock enable

Set and cleared by software.

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

0: HSE oscillator OFF

1: HSE oscillator ON

Bits 15:8 HSICAL[7:0]: 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.

Bit 2 Reserved, must be kept at reset value.

Bit 1 HSIRDY: Internal high-speed clock ready flag

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

0: HSI oscillator not ready

1: HSI oscillator ready

Bit 0 HSION: Internal high-speed clock enable

Set and cleared by software.

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

0: HSI oscillator OFF

1: HSI oscillator ON

5.3.2 RCC PLL configuration register (RCC_PLLCFGR)

Address offset: 0x04

Reset value: 0x2400 3010

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

This register is used to configure the PLL clock outputs according to the formulas:

- $f_{(VCO \text{ clock})} = f_{(\text{PLL clock input})} \times (\text{PLLN} / \text{PLLM})$
- $f_{(\text{PLL general clock output})} = f_{(\text{VCO clock})} / \text{PLLP}$
- $f_{(\text{USB OTG FS, SDMMC1/2, RNG clock output})} = f_{(\text{VCO clock})} / \text{PLLQ}$

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

Bits 31:28 Reserved, must be kept at reset value.

Bits 27:24 **PLLQ[3:0]:** Main PLL (PLL) division factor for USB OTG FS, SDMMC1/2 and random number generator clocks

Set and cleared by software to control the frequency of USB OTG FS clock, the random number generator clock and the SDMMC1/2 clock. These bits should be written only if PLL is disabled.

Caution: The USB OTG FS requires a 48 MHz clock to work correctly. The SDMMC1/2 and the random number generator need a frequency lower than or equal to 48 MHz to work correctly.

USB OTG FS clock frequency = VCO frequency / PLLQ with $2 \leq \text{PLLQ} \leq 15$

0000: PLLQ = 0, wrong configuration

0001: PLLQ = 1, wrong configuration

0010: PLLQ = 2

0011: PLLQ = 3

0100: PLLQ = 4

...

1111: PLLQ = 15

Bit 23 Reserved, must be kept at reset value.

Bit 22 **PLLSC:** Main PLL(PLL) and audio PLL (PLLI2S) entry clock source

Set and cleared by software to select PLL and PLLI2S clock source. This bit can be written only when PLL and PLLI2S are disabled.

0: HSI clock selected as PLL and PLLI2S clock entry

1: HSE oscillator clock selected as PLL and PLLI2S clock entry

Bits 21:18 Reserved, must be kept at reset value.

Bits 17:16 **PLL_P[1:0]**: Main PLL (PLL) division factor for main system clock

Set and cleared by software to control the frequency of the general PLL output clock. These bits can be written only if PLL is disabled.

Caution: The software has to set these bits correctly not to exceed 216 MHz on this domain.

PLL output clock frequency = VCO frequency / PLLP with PLLP = 2, 4, 6, or 8

00: PLLP = 2
01: PLLP = 4
10: PLLP = 6
11: PLLP = 8

Bits 14:6 **PLL_N[8:0]**: Main PLL (PLL) multiplication factor for VCO

Set and cleared by software to control the multiplication factor of the VCO. These bits can be written only when PLL is disabled. Only half-word and word accesses are allowed to write these bits.

Caution: The software has to set these bits correctly to ensure that the VCO output frequency is between 100 and 432 MHz.

VCO output frequency = VCO input frequency × PLLN with $50 \leq \text{PLLN} \leq 432$

00000000: PLLN = 0, wrong configuration
00000001: PLLN = 1, wrong configuration

...

000110010: PLLN = 50

...

001100011: PLLN = 99

001100100: PLLN = 100

...

110110000: PLLN = 432

110110001: PLLN = 433, wrong configuration

...

111111111: PLLN = 511, wrong configuration

Note: Between 50 and 99, multiplication factors are possible for VCO input frequency higher than 1 MHz. However care must be taken to fulfill the minimum VCO output frequency as specified above.

Bits 5:0 **PLL_M[5:0]**: Division factor for the main PLLs (PLL, PLLI2S and PLLSAI) input clock

Set and cleared by software to divide the PLL and PLLI2S input clock before the VCO. These bits can be written only when the PLL and PLLI2S are disabled.

Caution: The software has to set these bits correctly to ensure that the VCO input frequency ranges from 1 to 2 MHz. It is recommended to select a frequency of 2 MHz to limit PLL jitter.

VCO input frequency = PLL input clock frequency / PLLM with $2 \leq \text{PLLM} \leq 63$

00000: PLLM = 0, wrong configuration
000001: PLLM = 1, wrong configuration
000010: PLLM = 2
000011: PLLM = 3
000100: PLLM = 4
...
111110: PLLM = 62
111111: PLLM = 63

5.3.3 RCC clock configuration register (RCC_CFGR)

Address offset: 0x08

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 a clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MCO2		MCO2 PRE[2:0]			MCO1 PRE[2:0]			I2SSC R	MCO1		RTCPRE[4:0]				
rw		rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PPRE2[2:0]			PPRE1[2:0]			Res.	Res.	HPRE[3:0]				SWS1	SWS0	SW1	SW0
rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	r	r	rw	rw

Bits 31:30 **MCO2[1:0]:** Microcontroller clock output 2

Set and cleared by software. Clock source selection may generate glitches on MCO2. It is highly recommended to configure these bits only after reset before enabling the external oscillators and the PLLs.

- 00: System clock (SYSCLK) selected
- 01: PLLI2S clock selected
- 10: HSE oscillator clock selected
- 11: PLL clock selected

Bits 27:29 **MCO2PRE:** MCO2 prescaler

Set and cleared by software to configure the prescaler of the MCO2. Modification of this prescaler may generate glitches on MCO2. It is highly recommended to change this prescaler only after reset before enabling the external oscillators and the PLLs.

- 0xx: no division
- 100: division by 2
- 101: division by 3
- 110: division by 4
- 111: division by 5

Bits 24:26 **MCO1PRE:** MCO1 prescaler

Set and cleared by software to configure the prescaler of the MCO1. Modification of this prescaler may generate glitches on MCO1. It is highly recommended to change this prescaler only after reset before enabling the external oscillators and the PLL.

- 0xx: no division
- 100: division by 2
- 101: division by 3
- 110: division by 4
- 111: division by 5

Bit 23 **I2SSRC:** I2S clock selection

Set and cleared by software. This bit allows to select the I2S clock source between the PLLI2S clock and the external clock. It is highly recommended to change this bit only after reset and before enabling the I2S module.

- 0: PLLI2S clock used as I2S clock source
- 1: External clock mapped on the I2S_CKIN pin used as I2S clock source

Bits 22:21 **MCO1:** Microcontroller clock output 1

Set and cleared by software. Clock source selection may generate glitches on MCO1. It is highly recommended to configure these bits only after reset before enabling the external oscillators and PLL.

- 00: HSI clock selected
- 01: LSE oscillator selected
- 10: HSE oscillator clock selected
- 11: PLL clock selected

Bits 20:16 **RTCPRE:** HSE division factor for RTC clock

Set and cleared by software to divide the HSE clock input clock to generate a 1 MHz clock for RTC.

Caution: The software has to set these bits correctly to ensure that the clock supplied to the RTC is 1 MHz. These bits must be configured if needed before selecting the RTC clock source.

- 00000: no clock
- 00001: no clock
- 00010: HSE/2
- 00011: HSE/3
- 00100: HSE/4
- ...
- 11110: HSE/30
- 11111: HSE/31

Bits 15:13 **PPRE2:** APB high-speed prescaler (APB2)

Set and cleared by software to control APB high-speed clock division factor.

Caution: The software has to set these bits correctly not to exceed 108 MHz on this domain. The clocks are divided with the new prescaler factor from 1 to 16 AHB cycles after PPRE2 write.

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

Bits 12:10 **PPRE1:** APB Low-speed prescaler (APB1)

Set and cleared by software to control APB low-speed clock division factor.

Caution: The software has to set these bits correctly not to exceed 54 MHz on this domain. The clocks are divided with the new prescaler factor from 1 to 16 AHB cycles after PPRE1 write.

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

Bits 9:8 Reserved, must be kept at reset value.

Bits 7:4 HPRE: AHB prescaler

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

Caution: The clocks are divided with the new prescaler factor from 1 to 16 AHB cycles after HPRE write.

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

Bits 3:2 SWS: System clock switch status

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

- 00: HSI oscillator used as the system clock
- 01: HSE oscillator used as the system clock
- 10: PLL used as the system clock
- 11: not applicable

Bits 1:0 SW: System clock switch

Set and cleared by software to select the system clock source.

Set by hardware to force the HSI selection when leaving the Stop or Standby mode or in case of failure of the HSE oscillator used directly or indirectly as the system clock.

- 00: HSI oscillator selected as system clock
- 01: HSE oscillator selected as system clock
- 10: PLL selected as system clock
- 11: not allowed

5.3.4 RCC clock interrupt register (RCC_CIR)

Address offset: 0x0C

Reset value: 0x0000 0000

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CSSC	PLLSAI RDYC	PLL2S RDYC	PLL RDYC	HSE RDYC	HSI RDYC	LSE RDYC	LSI RDYC
								w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	PLLSAI RDYIE	PLL2S RDYIE	PLL RDYIE	HSE RDYIE	HSI RDYIE	LSE RDYIE	LSI RDYIE	CSSF	PLLSAI RDYF	PLL2S RDYF	PLL RDYF	HSE RDYF	HSI RDYF	LSE RDYF	LSI RDYF
rw	rw	rw	rw	rw	rw	rw	rw	r	r	r	r	r	r	r	r

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

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

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

0: No effect

1: Clear CSSF flag

Bit 22 **PLLSAIRDYC:** PLLSAI Ready Interrupt Clear

This bit is set by software to clear PLLSAIRDYF flag. It is reset by hardware when the PLLSAIRDYF is cleared.

0: PLLSAIRDYF not cleared

1: PLLSAIRDYF cleared

Bit 21 **PLL2SRDYC:** PLLI2S ready interrupt clear

This bit is set by software to clear the PLLI2SRDYF flag.

0: No effect

1: PLLI2SRDYF cleared

Bit 20 **PLLRDYC:** Main PLL(PLL) ready interrupt clear

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

0: No effect

1: PLLRDYF cleared

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

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

0: No effect

1: HSERDYF cleared

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

This bit is set software to clear the HSIRDYF flag.

0: No effect

1: HSIRDYF cleared

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

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

0: No effect

1: LSERDYF cleared

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

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

0: No effect

1: LSIRDYF cleared

Bit 15 Reserved, must be kept at reset value.

Bit 14 **PLLSAIRDYIE:** PLLSAI Ready Interrupt Enable

This bit is set and reset by software to enable/disable interrupt caused by PLLSAI lock.

0: PLLSAI lock interrupt disabled

1: PLLSAI lock interrupt enabled

Bit 13 **PLL2SRDYIE:** PLLI2S ready interrupt enable

This bit is set and cleared by software to enable/disable interrupt caused by PLLI2S lock.

0: PLLI2S lock interrupt disabled

1: PLLI2S lock interrupt enabled

Bit 12 **PLLRDYIE:** Main PLL (PLL) ready interrupt enable

This bit is set and cleared by software to enable/disable interrupt caused by PLL lock.

0: PLL lock interrupt disabled

1: PLL lock interrupt enabled

Bit 11 **HSERDYIE:** HSE ready interrupt enable

This bit is set and cleared by software to enable/disable interrupt caused by the HSE oscillator stabilization.

0: HSE ready interrupt disabled

1: HSE ready interrupt enabled

Bit 10 **HSIRDYIE:** HSI ready interrupt enable

This bit is set and cleared by software to enable/disable interrupt caused by the HSI oscillator stabilization.

0: HSI ready interrupt disabled

1: HSI ready interrupt enabled

Bit 9 **LSERDYIE:** LSE ready interrupt enable

This bit is set and cleared by software to enable/disable interrupt caused by the LSE oscillator stabilization.

0: LSE ready interrupt disabled

1: LSE ready interrupt enabled

Bit 8 **LSIRDYIE:** LSI ready interrupt enable

This bit is set and cleared by software to enable/disable interrupt caused by LSI oscillator stabilization.

0: LSI ready interrupt disabled

1: LSI ready interrupt enabled

Bit 7 **CSSF:** Clock security system interrupt flag

This bit is set by hardware when a failure is detected in the HSE oscillator.

It is cleared by software by setting the CSSC bit.

0: No clock security interrupt caused by HSE clock failure

1: Clock security interrupt caused by HSE clock failure

Bit 6 **PLLSAIRDYF:** PLLSAI Ready Interrupt flag

This bit is set by hardware when the PLLSAI is locked and PLLSAIRDYIE is set.

It is cleared by software by setting the PLLSAIRDYC bit.

0: No clock ready interrupt caused by PLLSAI lock

1: Clock ready interrupt caused by PLLSAI lock

Bit 5 **PLLI2SRDYF:** PLLI2S ready interrupt flag

This bit is set by hardware when the PLLI2S is locked and PLLI2SRDYIE is set.

It is cleared by software by setting the PLLI2SDYC bit.

0: No clock ready interrupt caused by PLLI2S lock

1: Clock ready interrupt caused by PLLI2S lock

Bit 4 **PLLRDYF:** Main PLL (PLL) ready interrupt flag

This bit is set by hardware when PLL is locked and PLLRDYIE is set.

It is cleared by software setting the PLLRDYC bit.

0: No clock ready interrupt caused by PLL lock

1: Clock ready interrupt caused by PLL lock

Bit 3 HSERDYF: HSE ready interrupt flag

This bit is set by hardware when External High Speed clock becomes stable and HSERDYDIE is set.

It is cleared by software by setting the HSERDYC bit.

0: No clock ready interrupt caused by the HSE oscillator

1: Clock ready interrupt caused by the HSE oscillator

Bit 2 HSIRDYF: HSI ready interrupt flag

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

It is cleared by software by setting the HSIRDYC bit.

0: No clock ready interrupt caused by the HSI oscillator

1: Clock ready interrupt caused by the HSI oscillator

Bit 1 LSERDYF: LSE ready interrupt flag

This bit is set by hardware when the External low-speed clock becomes stable and LSERDYDIE is set.

It is cleared by software by setting the LSERDYC bit.

0: No clock ready interrupt caused by the LSE oscillator

1: Clock ready interrupt caused by the LSE oscillator

Bit 0 LSIRDYF: LSI ready interrupt flag

This bit is set by hardware when the internal low-speed clock becomes stable and LSIRDYDIE is set.

It is cleared by software by setting the LSIRDYC bit.

0: No clock ready interrupt caused by the LSI oscillator

1: Clock ready interrupt caused by the LSI oscillator

5.3.5 RCC AHB1 peripheral reset register (RCC_AHB1RSTR)

Address offset: 0x10

Reset value: 0x0000 0000

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	OTGH S RST	Res.	Res.	Res.	Res.	Res.	Res.	DMA2 RST	DMA1 RST	Res.	Res.	Res.	Res.	Res.
		rw							rw	rw					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	CRCR ST	Res.	Res.	Res.	GPIOI RST	GPIOH RST	GPIOGG RST	GPIOF RST	GPIOE RST	GPIOD RST	GPIOC RST	GPIOB RST	GPIOA RST
			rw				rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30 Reserved, must be kept at reset value.

Bit 29 OTGHSRST: USB OTG HS module reset

This bit is set and cleared by software.

0: does not reset the USB OTG HS module

1: resets the USB OTG HS module

Bits 28:26 Reserved, must be kept at reset value.

Bit 25 Reserved, must be kept at reset value.

Bit 24 Reserved, must be kept at reset value.

Bit 23 Reserved, must be kept at reset value.

Bit 22 **DMA2RST:** DMA2 reset

This bit is set and cleared by software.

0: does not reset DMA2

1: resets DMA2

Bit 21 **DMA1RST:** DMA2 reset

This bit is set and cleared by software.

0: does not reset DMA2

1: resets DMA2

Bits 20:13 Reserved, must be kept at reset value.

Bit 12 **CRCRST:** CRC reset

This bit is set and cleared by software.

0: does not reset CRC

1: resets CRC

Bits 11:9 Reserved, must be kept at reset value.

Bit 8 **GPIOIRST:** IO port I reset

This bit is set and cleared by software.

0: does not reset IO port I

1: resets IO port I

Bit 7 **GPIOHRST:** IO port H reset

This bit is set and cleared by software.

0: does not reset IO port H

1: resets IO port H

Bit 6 **GPIOGRST:** IO port G reset

This bit is set and cleared by software.

0: does not reset IO port G

1: resets IO port G

Bit 5 **GPIOFRST:** IO port F reset

This bit is set and cleared by software.

0: does not reset IO port F

1: resets IO port F

Bit 4 **GPIOERST:** IO port E reset

This bit is set and cleared by software.

0: does not reset IO port E

1: resets IO port E

Bit 3 **GPIODRST:** IO port D reset

This bit is set and cleared by software.

0: does not reset IO port D

1: resets IO port D

Bit 2 GPIOCRST: IO port C reset

This bit is set and cleared by software.

0: does not reset IO port C

1: resets IO port C

Bit 1 GPIOBRST: IO port B reset

This bit is set and cleared by software.

0: does not reset IO port B

1: resets IO port B

Bit 0 GPIOARST: IO port A reset

This bit is set and cleared by software.

0: does not reset IO port A

1: resets IO port A

5.3.6 RCC AHB2 peripheral reset register (RCC_AHB2RSTR)

Address offset: 0x14

Reset value: 0x0000 0000

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

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

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

Bit 7 **OTGFSRST:** USB OTG FS module reset

Set and cleared by software.

0: does not reset the USB OTG FS module

1: resets the USB OTG FS module

Bit 6 **RNGRST:** Random number generator module reset

Set and cleared by software.

0: does not reset the random number generator module

1: resets the random number generator module

Bit 5 Reserved, must be kept at reset value.

Bit 4 **AESRST:** AES module reset

Set and cleared by software.

0: does not reset the AES module

1: resets the AES module

Bits 3:1 Reserved, must be kept at reset value.

Bit 0 Reserved, must be kept at reset value.

5.3.7 RCC AHB3 peripheral reset register (RCC_AHB3RSTR)

Address offset: 0x18

Reset value: 0x0000 0000

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

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

Bits 31:2 Reserved, must be kept at reset value.

Bit 1 **QSPIRST**: Quad SPI memory controller reset

Set and cleared by software.

0: does not reset the QUADSPI memory controller

1: resets the QUADSPI memory controller

Bit 0 **FMCRST**: Flexible memory controller module reset

Set and cleared by software.

0: does not reset the FMC module

1: resets the FMC module

5.3.8 RCC APB1 peripheral reset register (RCC_APB1RSTR)

Address offset: 0x20

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
UART8R ST	UART7R ST	DACRST	PWR RST	Res.	Res.	CAN1 RST	Res.	I2C3 RST	I2C2 RST	I2C1 RST	UART5 RST	UART4 RST	UART3 RST	UART2 RST	Res.
rw	rw	rw	rw			rw		rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 RST	SPI2 RST	Res.	Res.	WWWDG RST	Res.	LPTIM1 RST	TIM14 RST	TIM13 RST	TIM12 RST	TIM7 RST	TIM6 RST	TIM5 RST	TIM4 RST	TIM3 RST	TIM2 RST
rw	rw			rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Bit 31 **UART8RST:** UART8 reset
Set and cleared by software.
0: does not reset UART8
1: resets UART8
- Bit 30 **UART7RST:** UART7 reset
Set and cleared by software.
0: does not reset UART7
1: resets UART7
- Bit 29 **DACRST:** DAC reset
Set and cleared by software.
0: does not reset the DAC interface
1: resets the DAC interface
- Bit 28 **PWRRST:** Power interface reset
Set and cleared by software.
0: does not reset the power interface
1: resets the power interface
- Bits 27:26 Reserved, must be kept at reset value.
- Bit 25 **CAN1RST:** CAN1 reset
Set and cleared by software.
0: does not reset CAN1
1: resets CAN1
- Bit 24 IReserved, must be kept at reset value.
- Bit 23 **I2C3RST:** I2C3 reset
Set and cleared by software.
0: does not reset I2C3
1: resets I2C3
- Bit 22 **I2C2RST:** I2C2 reset
Set and cleared by software.
0: does not reset I2C2
1: resets I2C2
- Bit 21 **I2C1RST:** I2C1 reset
Set and cleared by software.
0: does not reset I2C1
1: resets I2C1
- Bit 20 **UART5RST:** UART5 reset
Set and cleared by software.
0: does not reset UART5
1: resets UART5
- Bit 19 **UART4RST:** USART4 reset
Set and cleared by software.
0: does not reset USART4
1: resets USART4
- Bit 18 **USART3RST:** USART3 reset
Set and cleared by software.
0: does not reset USART3
1: resets USART3

- Bit 17 **USART2RST:** USART2 reset
Set and cleared by software.
0: does not reset USART2
1: resets USART2
- Bit 16 Reserved, must be kept at reset value.
- Bit 15 **SPI3RST:** SPI3 reset
Set and cleared by software.
0: does not reset SPI3
1: resets SPI3
- Bit 14 **SPI2RST:** SPI2 reset
Set and cleared by software.
0: does not reset SPI2
1: resets SPI2
- Bits 13:12 Reserved, must be kept at reset value.
- Bit 11 **WWDGRST:** Window watchdog reset
Set and cleared by software.
0: does not reset the window watchdog
1: resets the window watchdog
- Bit 10 Reserved, must be kept at reset value.
- Bit 9 **LPTIM1RST:** Low-power timer 1 reset
Set and cleared by software.
0: does not reset LPTMI1
1: resets LPTMI1
- Bit 8 **TIM14RST:** TIM14 reset
Set and cleared by software.
0: does not reset TIM14
1: resets TIM14
- Bit 7 **TIM13RST:** TIM13 reset
Set and cleared by software.
0: does not reset TIM13
1: resets TIM13
- Bit 6 **TIM12RST:** TIM12 reset
Set and cleared by software.
0: does not reset TIM12
1: resets TIM12
- Bit 5 **TIM7RST:** TIM7 reset
Set and cleared by software.
0: does not reset TIM7
1: resets TIM7
- Bit 4 **TIM6RST:** TIM6 reset
Set and cleared by software.
0: does not reset TIM6
1: resets TIM6

Bit 3 **TIM5RST:** TIM5 reset

Set and cleared by software.

0: does not reset TIM5

1: resets TIM5

Bit 2 **TIM4RST:** TIM4 reset

Set and cleared by software.

0: does not reset TIM4

1: resets TIM4

Bit 1 **TIM3RST:** TIM3 reset

Set and cleared by software.

0: does not reset TIM3

1: resets TIM3

Bit 0 **TIM2RST:** TIM2 reset

Set and cleared by software.

0: does not reset TIM2

1: resets TIM2

5.3.9 RCC APB2 peripheral reset register (RCC_APB2RSTR)

Address offset: 0x24

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
OTGPHYC RST ⁽¹⁾	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SAI2 RST	SAI1 RST	Res.	SPI5 RST	Res.	TIM11 RST	TIM10 RST	TIM9 RST
rw									rw	rw		rw		rw	rw	rw
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SYSCFG RST	SPI4 RST	SPI1 RST	SDMMC1 RST	Res.	Res.	ADC RST	SDMMC2 RST	Res.	USART6 RST	USART1 RST	Res.	Res.	TIM8 RST	TIM1 RST	
	rw	rw	rw	rw			rw	rw		rw	rw			rw	rw	

1. Available on the STM32F7x3xx and STM32F730xx devices only.

Bit 31 **OTGPHCRST**: USB OTG HS PHY controller reset

This bit is set and cleared by software.

0: does not reset USBPHYC

1: resets USBPHYC

Bits 30:24 Reserved, must be kept at reset value.

Bit 23 **SAI2RST**: SAI2 reset

This bit is set and cleared by software.

0: does not reset SAI2

1: resets SAI2

Bit 22 **SAI1RST**: SAI1 reset

This bit is set and reset by software.

0: does not reset SAI1

1: resets SAI1

Bit 21 Reserved, must be kept at reset value.

Bit 20 **SPI5RST**: SPI5 reset

This bit is set and cleared by software.

0: does not reset SPI5

1: resets SPI5

Bit 19 Reserved, must be kept at reset value.

Bit 18 **TIM11RST**: TIM11 reset

This bit is set and cleared by software.

0: does not reset TIM11

1: resets TIM14

Bit 17 **TIM10RST**: TIM10 reset

This bit is set and cleared by software.

0: does not reset TIM10

1: resets TIM10

Bit 16 **TIM9RST**: TIM9 reset

This bit is set and cleared by software.

0: does not reset TIM9

1: resets TIM9

Bit 15 Reserved, must be kept at reset value.

Bit 14 **SYSCFGRST:** System configuration controller reset

This bit is set and cleared by software.

0: does not reset the System configuration controller
1: resets the System configuration controller

Bit 13 **SPI4RST:** SPI4 reset

This bit is set and cleared by software.

0: does not reset SPI4
1: resets SPI4

Bit 12 **SPI1RST:** SPI1 reset

This bit is set and cleared by software.

0: does not reset SPI1
1: resets SPI1

Bit 11 **SDMMC1RST:** SDMMC1 reset

This bit is set and cleared by software.

0: does not reset the SDMMC1 module
1: resets the SDMMC1 module

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

Bit 8 **ADCRST:** ADC interface reset (common to all ADCs)

This bit is set and cleared by software.

0: does not reset the ADC interface
1: resets the ADC interface

Bit 7 **SDMMC2RST:** SDMMC2 reset

This bit is set and cleared by software.

0: does not reset SDMMC2
1: resets SDMMC2

Bit 6 Reserved, must be kept at reset value.

Bit 5 **USART6RST:** USART6 reset

This bit is set and cleared by software.

0: does not reset USART6
1: resets USART6

Bit 4 **USART1RST:** USART1 reset

This bit is set and cleared by software.

0: does not reset USART1
1: resets USART1

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

Bit 1 **TIM8RST:** TIM8 reset

This bit is set and cleared by software.

0: does not reset TIM8
1: resets TIM8

Bit 0 **TIM1RST:** TIM1 reset

This bit is set and cleared by software.

0: does not reset TIM1
1: resets TIM1

5.3.10 RCC AHB1 peripheral clock register (RCC_AHB1ENR)

Address offset: 0x30

Reset value: 0x0010 0000

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	OTGHS ULPIEN (1)	OTGHS EN	Res.	Res.	Res.	Res.	Res.	Res.	DMA2 EN	DMA1 EN	DTCMRA MEN	Res.	BKPSR AMEN	Res.	Res.
	rw	rw							rw	rw	rw	rw	rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	CRC EN	Res.	Res.	Res.	GPIOI EN	GPIOH EN	GPIOG EN	GPIOF EN	GPIOE EN	GPIOD EN	GPIOC EN	GPIO BEN	GPIO AEN
			rw				rw	rw	rw	rw	rw	rw	rw	rw	rw

1. Available on the STM32F7x2xx devices only.

Bit 31 Reserved, must be kept at reset value.

Bit 30 **OTGHSULPIEN**: USB OTG HSULPI clock enable

This bit is set and cleared by software.

0: USB OTG HS ULPI clock disabled

1: USB OTG HS ULPI clock enabled

Bit 29 **OTGHSEN**: USB OTG HS clock enable

This bit is set and cleared by software.

0: USB OTG HS clock disabled

1: USB OTG HS clock enabled

Bits 28:23 Reserved, must be kept at reset value.

Bit 22 **DMA2EN**: DMA2 clock enable

This bit is set and cleared by software.

0: DMA2 clock disabled

1: DMA2 clock enabled

Bit 21 **DMA1EN**: DMA1 clock enable

This bit is set and cleared by software.

0: DMA1 clock disabled

1: DMA1 clock enabled

Bit 20 **DTCMRAMEN**: DTCM data RAM clock enable

This bit is set and cleared by software.

0: DTCM RAM clock disabled

1: DTCM RAM clock enabled

Bit 19 Reserved, must be kept at reset value.

Bit 18 **BKPSRAMEN**: Backup SRAM interface clock enable

This bit is set and cleared by software.

0: Backup SRAM interface clock disabled

1: Backup SRAM interface clock enabled

Bits 17:13 Reserved, must be kept at reset value.

- Bit 12 **CRCEN**: CRC clock enable
 This bit is set and cleared by software.
 0: CRC clock disabled
 1: CRC clock enabled
- Bits 11:9 Reserved, must be kept at reset value.
- Bit 8 **GPIOIEN**: IO port I clock enable
 This bit is set and cleared by software.
 0: IO port I clock disabled
 1: IO port I clock enabled
- Bit 7 **GPIOHEN**: IO port H clock enable
 This bit is set and cleared by software.
 0: IO port H clock disabled
 1: IO port H clock enabled
- Bit 6 **GPIOGEN**: IO port G clock enable
 This bit is set and cleared by software.
 0: IO port G clock disabled
 1: IO port G clock enabled
- Bit 5 **GPIOFEN**: IO port F clock enable
 This bit is set and cleared by software.
 0: IO port F clock disabled
 1: IO port F clock enabled
- Bit 4 **GPIOEEN**: IO port E clock enable
 This bit is set and cleared by software.
 0: IO port E clock disabled
 1: IO port E clock enabled
- Bit 3 **GPIODEN**: IO port D clock enable
 This bit is set and cleared by software.
 0: IO port D clock disabled
 1: IO port D clock enabled
- Bit 2 **GPIOCEN**: IO port C clock enable
 This bit is set and cleared by software.
 0: IO port C clock disabled
 1: IO port C clock enabled
- Bit 1 **GPIOBEN**: IO port B clock enable
 This bit is set and cleared by software.
 0: IO port B clock disabled
 1: IO port B clock enabled
- Bit 0 **GPIOAEN**: IO port A clock enable
 This bit is set and cleared by software.
 0: IO port A clock disabled
 1: IO port A clock enabled

5.3.11 RCC AHB2 peripheral clock enable register (RCC_AHB2ENR)

Address offset: 0x34

Reset value: 0x0000 0000

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

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

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

Bit 7 **OTGFSEN**: USB OTG FS clock enable

This bit is set and cleared by software.

0: USB OTG FS clock disabled

1: USB OTG FS clock enabled

Bit 6 **RNGEN**: Random number generator clock enable

This bit is set and cleared by software.

0: Random number generator clock disabled

1: Random number generator clock enabled

Bit 5 Reserved, must be kept at reset value.

Bit 4 **AESEN**: AES module clock enable

This bit is set and cleared by software.

0: AES module clock disabled

1: AES module clock enabled

Bits 3:0 Reserved, must be kept at reset value.

5.3.12 RCC AHB3 peripheral clock enable register (RCC_AHB3ENR)

Address offset: 0x38

Reset value: 0x0000 0000

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

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

Bits 31:2 Reserved, must be kept at reset value.

Bit 1 **QSPIEN**: Quad SPI memory controller clock enable

This bit is set and cleared by software.

0: QUASPI controller clock disabled

1: QUASPI controller clock enabled

Bit 0 **FMCEN**: Flexible memory controller clock enable

This bit is set and cleared by software.

0: FMC clock disabled

1: FMC clock enabled

5.3.13 RCC APB1 peripheral clock enable register (RCC_APB1ENR)

Address offset: 0x40

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
UART8 EN	UART7 EN	DAC EN	PWR EN	Res.	Res.	CAN1 EN	Res.	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART3 EN	USART2 EN	Res.
rw	rw	rw	rw			rw		rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Res.	Res.	WWDG EN	RTCAP BEN	LPTIM1 EN	TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **UART8EN**: UART8 clock enable

This bit is set and cleared by software.

0: UART8 clock disabled

1: UART8 clock enabled

Bit 30 **UART7EN**: UART7 clock enable

This bit is set and cleared by software.

0: UART7 clock disabled

1: UART7 clock enabled

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

This bit is set and cleared by software.

0: DAC interface clock disabled

1: DAC interface clock enable

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

This bit is set and cleared by software.

0: Power interface clock disabled

1: Power interface clock enable

Bits 27:26 Reserved, must be kept at reset value.

Bit 25 **CAN1EN**: CAN 1 clock enable

This bit is set and cleared by software.

0: CAN 1 clock disabled

1: CAN 1 clock enabled

Bit 24 Reserved, must be kept at reset value.

Bit 23 **I2C3EN**: I2C3 clock enable

This bit is set and cleared by software.

0: I2C3 clock disabled

1: I2C3 clock enabled

Bit 22 **I2C2EN**: I2C2 clock enable

This bit is set and cleared by software.

0: I2C2 clock disabled

1: I2C2 clock enabled

- Bit 21 **I2C1EN:** I2C1 clock enable
This bit is set and cleared by software.
0: I2C1 clock disabled
1: I2C1 clock enabled
- Bit 20 **UART5EN:** UART5 clock enable
This bit is set and cleared by software.
0: UART5 clock disabled
1: UART5 clock enabled
- Bit 19 **UART4EN:** UART4 clock enable
This bit is set and cleared by software.
0: UART4 clock disabled
1: UART4 clock enabled
- Bit 18 **USART3EN:** USART3 clock enable
This bit is set and cleared by software.
0: USART3 clock disabled
1: USART3 clock enabled
- Bit 17 **USART2EN:** USART2 clock enable
This bit is set and cleared by software.
0: USART2 clock disabled
1: USART2 clock enabled
- Bit 16 Reserved, must be kept at reset value.
- Bit 15 **SPI3EN:** SPI3 clock enable
This bit is set and cleared by software.
0: SPI3 clock disabled
1: SPI3 clock enabled
- Bit 14 **SPI2EN:** SPI2 clock enable
This bit is set and cleared by software.
0: SPI2 clock disabled
1: SPI2 clock enabled
- Bits 13:12 Reserved, must be kept at reset value.
- Bit 11 **WWDGEN:** Window watchdog clock enable
This bit is set and cleared by software.
0: Window watchdog clock disabled
1: Window watchdog clock enabled
- Bit 10 **RTCAPBEN:** RTCAPB clock enable
This bit is set and cleared by software.
0: RTCAPB clock disabled
1: RTCAPB clock enabled
- Bit 9 **LPTM1EN:** Low-power timer 1 clock enable
This bit is set and cleared by software.
0: LPTIM1 clock disabled
1: LPTIM1 clock enabled
- Bit 8 **TIM14EN:** TIM14 clock enable
This bit is set and cleared by software.
0: TIM14 clock disabled
1: TIM14 clock enabled

Bit 7 TIM13EN: TIM13 clock enable

This bit is set and cleared by software.

0: TIM13 clock disabled

1: TIM13 clock enabled

Bit 6 TIM12EN: TIM12 clock enable

This bit is set and cleared by software.

0: TIM12 clock disabled

1: TIM12 clock enabled

Bit 5 TIM7EN: TIM7 clock enable

This bit is set and cleared by software.

0: TIM7 clock disabled

1: TIM7 clock enabled

Bit 4 TIM6EN: TIM6 clock enable

This bit is set and cleared by software.

0: TIM6 clock disabled

1: TIM6 clock enabled

Bit 3 TIM5EN: TIM5 clock enable

This bit is set and cleared by software.

0: TIM5 clock disabled

1: TIM5 clock enabled

Bit 2 TIM4EN: TIM4 clock enable

This bit is set and cleared by software.

0: TIM4 clock disabled

1: TIM4 clock enabled

Bit 1 TIM3EN: TIM3 clock enable

This bit is set and cleared by software.

0: TIM3 clock disabled

1: TIM3 clock enabled

Bit 0 TIM2EN: TIM2 clock enable

This bit is set and cleared by software.

0: TIM2 clock disabled

1: TIM2 clock enabled

5.3.14 RCC APB2 peripheral clock enable register (RCC_APB2ENR)

Address offset: 0x44

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
OTGPHYC EN ⁽¹⁾	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SAI2EN	SAI1EN	Res.	SPI5EN	Res.	TIM11 EN	TIM10 EN	TIM9 EN
rw								rw	rw		rw		rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SYSCFG EN	SPI4 EN	SPI1 EN	SDMMC1 EN	ADC3 EN	ADC2 EN	ADC1 EN	SDMMC2 EN	Res.	USART6 EN	USART1 EN	Res.	Res.	TIM8 EN	TIM1 EN
	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw			rw	rw

1. Available on the STM32F7x3xx and STM32F730xx devices only.

Bit 31 **OTGPHYCEN:** USB OTG HS PHY controller clock enable

This bit is set and cleared by software.

0: USBPHYC clock disabled

1: USBPHYC clock enabled

Bits 30 24 Reserved, must be kept at reset value.

Bit 23 **SAI2EN:** SAI2 clock enable

This bit is set and cleared by software.

0: SAI2 clock disabled

1: SAI2 clock enabled

Bit 22 **SAI1EN:** SAI1 clock enable

This bit is set and cleared by software.

0: SAI1 clock disabled

1: SAI1 clock enabled

Bit 21 Reserved, must be kept at reset value.

Bit 20 **SPI5EN:** SPI5 clock enable

This bit is set and cleared by software.

0: SPI5 clock disabled

1: SPI5 clock enabled

Bit 18 **TIM11EN:** TIM11 clock enable

This bit is set and cleared by software.

0: TIM11 clock disabled

1: TIM11 clock enabled

Bit 17 **TIM10EN:** TIM10 clock enable

This bit is set and cleared by software.

0: TIM10 clock disabled

1: TIM10 clock enabled

Bit 16 **TIM9EN:** TIM9 clock enable

This bit is set and cleared by software.

0: TIM9 clock disabled

1: TIM9 clock enabled

Bit 15 Reserved, must be kept at reset value.

- Bit 14 **SYSCFGEN:** System configuration controller clock enable
This bit is set and cleared by software.
0: System configuration controller clock disabled
1: System configuration controller clock enabled
- Bit 13 **SPI4EN:** SPI4 clock enable
This bit is set and cleared by software.
0: SPI4 clock disabled
1: SPI4 clock enabled
- Bit 12 **SPI1EN:** SPI1 clock enable
This bit is set and cleared by software.
0: SPI1 clock disabled
1: SPI1 clock enabled
- Bit 11 **SDMMC1EN:** SDMMC1 clock enable
This bit is set and cleared by software.
0: SDMMC1 module clock disabled
1: SDMMC1 module clock enabled
- Bit 10 **ADC3EN:** ADC3 clock enable
This bit is set and cleared by software.
0: ADC3 clock disabled
1: ADC3 clock enabled
- Bit 9 **ADC2EN:** ADC2 clock enable
This bit is set and cleared by software.
0: ADC2 clock disabled
1: ADC2 clock enabled
- Bit 8 **ADC1EN:** ADC1 clock enable
This bit is set and cleared by software.
0: ADC1 clock disabled
1: ADC1 clock enabled
- Bit 7 **SDMMC2EN:** SDMMC2 clock enable
This bit is set and cleared by software.
0: SDMMC2 clock disabled
1: SDMMC2 clock disabled
- Bit 6 Reserved, must be kept at reset value.
- Bit 5 **USART6EN:** USART6 clock enable
This bit is set and cleared by software.
0: USART6 clock disabled
1: USART6 clock enabled
- Bit 4 **USART1EN:** USART1 clock enable
This bit is set and cleared by software.
0: USART1 clock disabled
1: USART1 clock enabled

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

Bit 1 **TIM8EN:** TIM8 clock enable

This bit is set and cleared by software.

0: TIM8 clock disabled

1: TIM8 clock enabled

Bit 0 **TIM1EN:** TIM1 clock enable

This bit is set and cleared by software.

0: TIM1 clock disabled

1: TIM1 clock enabled

5.3.15 RCC AHB1 peripheral clock enable in low-power mode register (RCC_AHB1LPENR)

Address offset: 0x50

Reset value: 0x7EF7 B7FFh

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	OTGHS ULPI LPEN ⁽¹⁾	OTGHS LPEN	Res.	Res.	Res.	Res.	Res.	Res.	DMA2 LPEN	DMA1 LPEN	DTCM LPEN	Res.	BKPS RAM LPEN	SRAM2 LPEN	SRAM1 LPEN
	rw	rw							rw	rw	rw		rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FLITF LPEN	Res.	AXI LPEN	CRC LPEN	Res.	Res.	Res.	GPIOI LPEN	GPIOH LPEN	GPIOGG LPEN	GPIOF LPEN	GPIOE LPEN	GPIOD LPEN	GPIOC LPEN	GPIOB LPEN	GPIOA LPEN
	rw	rw	rw				rw	rw	rw	rw	rw	rw	rw	rw	rw

1. Available on the STM32F7x2xx devices only.

Bit 31 Reserved, must be kept at reset value.

Bit 30 **OTGHSULPILPEN:** USB OTG HS ULPI clock enable during Sleep mode

This bit is set and cleared by software.

0: USB OTG HS ULPI clock disabled during Sleep mode

1: USB OTG HS ULPI clock enabled during Sleep mode

Bit 29 **OTGHSLPEN:** USB OTG HS clock enable during Sleep mode

This bit is set and cleared by software.

0: USB OTG HS clock disabled during Sleep mode

1: USB OTG HS clock enabled during Sleep mode

Bits 28:23 Reserved, must be kept at reset value.

Bit 22 **DMA2LPEN:** DMA2 clock enable during Sleep mode

This bit is set and cleared by software.

0: DMA2 clock disabled during Sleep mode

1: DMA2 clock enabled during Sleep mode

Bit 21 **DMA1LPEN:** DMA1 clock enable during Sleep mode

This bit is set and cleared by software.

0: DMA1 clock disabled during Sleep mode

1: DMA1 clock enabled during Sleep mode

Bit 20 **DTCMLPEN:** DTCM RAM interface clock enable during Sleep mode

This bit is set and cleared by software.

0: DTCM RAM interface clock disabled during Sleep mode

1: DTCM RAM interface clock enabled during Sleep mode

Bit 19 Reserved, must be kept at reset value.

Bit 18 **BKPSRAMLPEN:** Backup SRAM interface clock enable during Sleep mode

This bit is set and cleared by software.

0: Backup SRAM interface clock disabled during Sleep mode

1: Backup SRAM interface clock enabled during Sleep mode

- Bit 17 **SRAM2LPEN:** SRAM2 interface clock enable during Sleep mode
 This bit is set and cleared by software.
 0: SRAM2 interface clock disabled during Sleep mode
 1: SRAM2 interface clock enabled during Sleep mode
- Bit 16 **SRAM1LPEN:** SRAM1 interface clock enable during Sleep mode
 This bit is set and cleared by software.
 0: SRAM1 interface clock disabled during Sleep mode
 1: SRAM1 interface clock enabled during Sleep mode
- Bit 15 **FLITFLPEN:** Flash interface clock enable during Sleep mode
 This bit is set and cleared by software.
 0: Flash interface clock disabled during Sleep mode
 1: Flash interface clock enabled during Sleep mode
- Bit 14 Reserved, must be kept at reset value.
- Bit 13 **AXILPEN:** AXI to AHB bridge clock enable during Sleep mode
 This bit is set and cleared by software.
 0: AXI to AHB bridge clock disabled during Sleep mode
 1: AXI to AHB bridge clock enabled during Sleep mode
- Bit 12 **CRCLPEN:** CRC clock enable during Sleep mode
 This bit is set and cleared by software.
 0: CRC clock disabled during Sleep mode
 1: CRC clock enabled during Sleep mode
- Bits 11:9 Reserved, must be kept at reset value.
- Bit 8 **GPIOILPEN:** IO port I clock enable during Sleep mode
 This bit is set and cleared by software.
 0: IO port I clock disabled during Sleep mode
 1: IO port I clock enabled during Sleep mode
- Bit 7 **GPIOHLPEN:** IO port H clock enable during Sleep mode
 This bit is set and cleared by software.
 0: IO port H clock disabled during Sleep mode
 1: IO port H clock enabled during Sleep mode
- Bits 6 **GPIOGLPEN:** IO port G clock enable during Sleep mode
 This bit is set and cleared by software.
 0: IO port G clock disabled during Sleep mode
 1: IO port G clock enabled during Sleep mode
- Bit 5 **GPIOFLPEN:** IO port F clock enable during Sleep mode
 This bit is set and cleared by software.
 0: IO port F clock disabled during Sleep mode
 1: IO port F clock enabled during Sleep mode
- Bit 4 **GPIOELPEN:** IO port E clock enable during Sleep mode
 Set and cleared by software.
 0: IO port E clock disabled during Sleep mode
 1: IO port E clock enabled during Sleep mode
- Bit 3 **GPIODLPEN:** IO port D clock enable during Sleep mode
 This bit is set and cleared by software.
 0: IO port D clock disabled during Sleep mode
 1: IO port D clock enabled during Sleep mode

Bit 2 **GPIOCLPEN**: IO port C clock enable during Sleep mode

This bit is set and cleared by software.

0: IO port C clock disabled during Sleep mode

1: IO port C clock enabled during Sleep mode

Bit 1 **GPIOBBLPEN**: IO port B clock enable during Sleep mode

This bit is set and cleared by software.

0: IO port B clock disabled during Sleep mode

1: IO port B clock enabled during Sleep mode

Bit 0 **GPIOALPEN**: IO port A clock enable during sleep mode

This bit is set and cleared by software.

0: IO port A clock disabled during Sleep mode

1: IO port A clock enabled during Sleep mode

5.3.16 RCC AHB2 peripheral clock enable in low-power mode register (RCC_AHB2LPENR)

Address offset: 0x54

Reset value: 0x0000 00F1

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

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

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

Bit 7 **OTGFSLPEN**: USB OTG FS clock enable during Sleep mode

This bit is set and cleared by software.

0: USB OTG FS clock disabled during Sleep mode

1: USB OTG FS clock enabled during Sleep mode

Bit 6 **RNGLPEN**: Random number generator clock enable during Sleep mode

This bit is set and cleared by software.

0: Random number generator clock disabled during Sleep mode

1: Random number generator clock enabled during Sleep mode

Bit 5 Reserved, must be kept at reset value.

Bit 4 **AESLPEN**: AES module clock enable during Sleep mode

This bit is set and cleared by software.

0: AES modules clock disabled during Sleep mode

1: AES modules clock enabled during Sleep mode

Bits 3:0 Reserved, must be kept at reset value.

5.3.17 RCC AHB3 peripheral clock enable in low-power mode register (RCC_AHB3LPENR)

Address offset: 0x58

Reset value: 0x0000 0003

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

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

Bits 31:2Reserved, must be kept at reset value.

Bit 1 **QSPILPEN**: QUADSPI memory controller clock enable during Sleep mode

This bit is set and cleared by software.

0: QUADSPI controller clock disabled during Sleep mode

1: QUADSPI controller clock enabled during Sleep mode

Bit 0 **FMCCLPEN**: Flexible memory controller module clock enable during Sleep mode

This bit is set and cleared by software.

0: FMC module clock disabled during Sleep mode

1: FMC module clock enabled during Sleep mode

5.3.18 RCC APB1 peripheral clock enable in low-power mode register (RCC_APB1LPENR)

Address offset: 0x60

Reset value: 0xFFFF CBFFh

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
UART8 LPEN	UART7 LPEN	DAC LPEN	PWR LPEN	Res.	Res.	CAN1 LPEN	Res.	I2C3 LPEN	I2C2 LPEN	I2C1 LPEN	UART5 LPEN	UART4 LPEN	USART3 LPEN	USART2 LPEN	Res.
rw	rw	rw	rw			rw		rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 LPEN	SPI2 LPEN	Res.	Res.	WWDG LPEN	RTCAPB LPEN	LPTM1 LPEN	TIM14 LPEN	TIM13 LPEN	TIM12 LPEN	TIM7 LPEN	TIM6 LPEN	TIM5 LPEN	TIM4 LPEN	TIM3 LPEN	TIM2 LPEN
rw	rw			rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw

- Bit 31 **UART8LPEN:** UART8 clock enable during Sleep mode
This bit is set and cleared by software.
0: UART8 clock disabled during Sleep mode
1: UART8 clock enabled during Sleep mode
- Bit 30 **UART7LPEN:** UART7 clock enable during Sleep mode
This bit is set and cleared by software.
0: UART7 clock disabled during Sleep mode
1: UART7 clock enabled during Sleep mode
- Bit 29 **DACLPEN:** DAC interface clock enable during Sleep mode
This bit is set and cleared by software.
0: DAC interface clock disabled during Sleep mode
1: DAC interface clock enabled during Sleep mode
- Bit 28 **PWRLPEN:** Power interface clock enable during Sleep mode
This bit is set and cleared by software.
0: Power interface clock disabled during Sleep mode
1: Power interface clock enabled during Sleep mode
- Bits 27:26 Reserved, must be kept at reset value.
- Bit 25 **CAN1LPEN:** CAN 1 clock enable during Sleep mode
This bit is set and cleared by software.
0: CAN 1 clock disabled during Sleep mode
1: CAN 1 clock enabled during Sleep mode
- Bit 24 Reserved, must be kept at reset value.
- Bit 23 **I2C3LPEN:** I2C3 clock enable during Sleep mode
This bit is set and cleared by software.
0: I2C3 clock disabled during Sleep mode
1: I2C3 clock enabled during Sleep mode
- Bit 22 **I2C2LPEN:** I2C2 clock enable during Sleep mode
This bit is set and cleared by software.
0: I2C2 clock disabled during Sleep mode
1: I2C2 clock enabled during Sleep mode
- Bit 21 **I2C1LPEN:** I2C1 clock enable during Sleep mode
This bit is set and cleared by software.
0: I2C1 clock disabled during Sleep mode
1: I2C1 clock enabled during Sleep mode
- Bit 20 **UART5LPEN:** UART5 clock enable during Sleep mode
This bit is set and cleared by software.
0: UART5 clock disabled during Sleep mode
1: UART5 clock enabled during Sleep mode
- Bit 19 **UART4LPEN:** UART4 clock enable during Sleep mode
This bit is set and cleared by software.
0: UART4 clock disabled during Sleep mode
1: UART4 clock enabled during Sleep mode
- Bit 18 **USART3LPEN:** USART3 clock enable during Sleep mode
This bit is set and cleared by software.
0: USART3 clock disabled during Sleep mode
1: USART3 clock enabled during Sleep mode

- Bit 17 **USART2LPEN:** USART2 clock enable during Sleep mode
 This bit is set and cleared by software.
 0: USART2 clock disabled during Sleep mode
 1: USART2 clock enabled during Sleep mode
- Bit 16 Reserved, must be kept at reset value.
- Bit 15 **SPI3LPEN:** SPI3 clock enable during Sleep mode
 This bit is set and cleared by software.
 0: SPI3 clock disabled during Sleep mode
 1: SPI3 clock enabled during Sleep mode
- Bit 14 **SPI2LPEN:** SPI2 clock enable during Sleep mode
 This bit is set and cleared by software.
 0: SPI2 clock disabled during Sleep mode
 1: SPI2 clock enabled during Sleep mode
- Bits 13:12 Reserved, must be kept at reset value.
- Bit 11 **WWDGLPEN:** Window watchdog clock enable during Sleep mode
 This bit is set and cleared by software.
 0: Window watchdog clock disabled during Sleep mode
 1: Window watchdog clock enabled during Sleep mode
- Bit 10 **RTCAPBLPEN:** RTCAPB clock enable during Sleep mode
 This bit is set and cleared by software.
 0: RTCAPB clock disabled during Sleep mode
 1: RTCAPB clock enabled during Sleep mode
- Bit 9 **LPTIM1LPEN:** low-power timer 1 clock enable during Sleep mode
 This bit is set and cleared by software.
 0: LPTIM1 clock disabled during Sleep mode
 1: LPTIM1 clock enabled during Sleep mode
- Bit 8 **TIM14LPEN:** TIM14 clock enable during Sleep mode
 This bit is set and cleared by software.
 0: TIM14 clock disabled during Sleep mode
 1: TIM14 clock enabled during Sleep mode
- Bit 7 **TIM13LPEN:** TIM13 clock enable during Sleep mode
 This bit is set and cleared by software.
 0: TIM13 clock disabled during Sleep mode
 1: TIM13 clock enabled during Sleep mode
- Bit 6 **TIM12LPEN:** TIM12 clock enable during Sleep mode
 This bit is set and cleared by software.
 0: TIM12 clock disabled during Sleep mode
 1: TIM12 clock enabled during Sleep mode
- Bit 5 **TIM7LPEN:** TIM7 clock enable during Sleep mode
 This bit is set and cleared by software.
 0: TIM7 clock disabled during Sleep mode
 1: TIM7 clock enabled during Sleep mode
- Bit 4 **TIM6LPEN:** TIM6 clock enable during Sleep mode
 This bit is set and cleared by software.
 0: TIM6 clock disabled during Sleep mode
 1: TIM6 clock enabled during Sleep mode

Bit 3 **TIM5LPEN:** TIM5 clock enable during Sleep mode

This bit is set and cleared by software.

0: TIM5 clock disabled during Sleep mode

1: TIM5 clock enabled during Sleep mode

Bit 2 **TIM4LPEN:** TIM4 clock enable during Sleep mode

This bit is set and cleared by software.

0: TIM4 clock disabled during Sleep mode

1: TIM4 clock enabled during Sleep mode

Bit 1 **TIM3LPEN:** TIM3 clock enable during Sleep mode

This bit is set and cleared by software.

0: TIM3 clock disabled during Sleep mode

1: TIM3 clock enabled during Sleep mode

Bit 0 **TIM2LPEN:** TIM2 clock enable during Sleep mode

This bit is set and cleared by software.

0: TIM2 clock disabled during Sleep mode

1: TIM2 clock enabled during Sleep mode

5.3.19 RCC APB2 peripheral clock enabled in low-power mode register (RCC_APB2LPENR)

Address offset: 0x64

Reset value: 0x04F7 7F33h

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SAI2 LPEN	SAI1 LPEN	Res.	SPI5 LPEN	Res.	TIM11 LPEN	TIM10 LPEN	TIM9 LPEN
								rw	rw		rw		rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SYSCFG LPEN	SPI4 LPEN	SPI1 LPEN	SDMMC1 LPEN	ADC3 LPEN	ADC2 LPEN	ADC1 LPEN	SDMMC2 LPEN	Res.	USART6 LPEN	USART1 LPEN	Res.	Res.	TIM8 LPEN	TIM1 LPEN
	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw			rw	rw

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

Bit 23 **SAI2LPEN**: SAI2 clock enable during Sleep mode

This bit is set and cleared by software.

0: SAI2 clock disabled during Sleep mode

1: SAI2 clock enabled during Sleep mode

Bit 22 **SAI1LPEN**: SAI1 clock enable during Sleep mode

This bit is set and cleared by software.

0: SAI1 clock disabled during Sleep mode

1: SAI1 clock enabled during Sleep mode

Bit 21 Reserved, must be kept at reset value.

Bit 20 **SPI5LPEN**: SPI5 clock enable during Sleep mode

This bit is set and cleared by software.

0: SPI5 clock disabled during Sleep mode

1: SPI5 clock enabled during Sleep mode

Bit 19 Reserved, must be kept at reset value.

Bit 18 **TIM11LPEN**: TIM11 clock enable during Sleep mode

This bit is set and cleared by software.

0: TIM11 clock disabled during Sleep mode

1: TIM11 clock enabled during Sleep mode

Bit 17 **TIM10LPEN**: TIM10 clock enable during Sleep mode

This bit is set and cleared by software.

0: TIM10 clock disabled during Sleep mode

1: TIM10 clock enabled during Sleep mode

Bit 16 **TIM9LPEN**: TIM9 clock enable during sleep mode

This bit is set and cleared by software.

0: TIM9 clock disabled during Sleep mode

1: TIM9 clock enabled during Sleep mode

Bit 15 Reserved, must be kept at reset value.

- Bit 14 **SYSCFGLPEN:** System configuration controller clock enable during Sleep mode
This bit is set and cleared by software.
0: System configuration controller clock disabled during Sleep mode
1: System configuration controller clock enabled during Sleep mode
- Bit 13 **SPI4LPEN:** SPI4 clock enable during Sleep mode
This bit is set and cleared by software.
0: SPI4 clock disabled during Sleep mode
1: SPI4 clock enabled during Sleep mode
- Bit 12 **SPI1LPEN:** SPI1 clock enable during Sleep mode
This bit is set and cleared by software.
0: SPI1 clock disabled during Sleep mode
1: SPI1 clock enabled during Sleep mode
- Bit 11 **SDMMC1LPEN:** SDMMC1 clock enable during Sleep mode
This bit is set and cleared by software.
0: SDMMC1 module clock disabled during Sleep mode
1: SDMMC1 module clock enabled during Sleep mode
- Bit 10 **ADC3LPEN:** ADC 3 clock enable during Sleep mode
This bit is set and cleared by software.
0: ADC 3 clock disabled during Sleep mode
1: ADC 3 clock enabled during Sleep mode
- Bit 9 **ADC2LPEN:** ADC2 clock enable during Sleep mode
This bit is set and cleared by software.
0: ADC2 clock disabled during Sleep mode
1: ADC2 clock enabled during Sleep mode
- Bit 8 **ADC1LPEN:** ADC1 clock enable during Sleep mode
This bit is set and cleared by software.
0: ADC1 clock disabled during Sleep mode
1: ADC1 clock enabled during Sleep mode
- Bit 7 **SDMMC2LPEN:** SDMMC2 clock enable during Sleep mode
This bit is set and cleared by software.
0: SDMMC2 module clock disabled during Sleep mode
1: SDMMC2 module clock enabled during Sleep mode
- Bit 6 Reserved, must be kept at reset value.
- Bit 5 **USART6LPEN:** USART6 clock enable during Sleep mode
This bit is set and cleared by software.
0: USART6 clock disabled during Sleep mode
1: USART6 clock enabled during Sleep mode
- Bit 4 **USART1LPEN:** USART1 clock enable during Sleep mode
This bit is set and cleared by software.
0: USART1 clock disabled during Sleep mode
1: USART1 clock enabled during Sleep mode

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

Bit 1 **TIM8LPEN:** TIM8 clock enable during Sleep mode

This bit is set and cleared by software.

0: TIM8 clock disabled during Sleep mode

1: TIM8 clock enabled during Sleep mode

Bit 0 **TIM1LPEN:** TIM1 clock enable during Sleep mode

This bit is set and cleared by software.

0: TIM1 clock disabled during Sleep mode

1: TIM1 clock enabled during Sleep mode

5.3.20 RCC backup domain control register (RCC_BDCR)

Address offset: 0x70

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.

The LSEON, LSEBYP, LSEDRV[1:0], RTCSEL and RTCEN bits in the [RCC 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 [PWR power control register \(PWR_CR1\)](#) has to be set before these can be modified. Refer to [Section 5.1.1: System reset on page 128](#) for further information. These bits are only reset after a Backup domain Reset (see [Section 5.1.3: Backup domain reset](#)). 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
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	BDRST
															rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTCEN	Res.	Res.	Res.	Res.	Res.	RTCSEL[1:0]	Res.	Res.	Res.	LSEDRV[1:0]	LSEBYP	LSERDY	LSEON		
rw						rw	rw			rw	rw	rw	r	rw	

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

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

This bit is set and cleared by software.

0: Reset not activated

1: Resets the entire Backup domain

Note: The BKPSRAM is not affected by this reset, the only way of resetting the BKPSRAM is through the Flash interface when a protection level change from level 1 to level 0 is requested.

Bit 15 **RTCEN:** RTC clock enable

This bit is set and cleared by software.

0: RTC clock disabled

1: RTC clock enabled

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

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

These bits are 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 the RTC clock

10: LSI oscillator clock used as the RTC clock

11: HSE oscillator clock divided by a programmable prescaler (selection through the RTCPRE[4:0] bits in the RCC clock configuration register (RCC_CFGR)) used as the RTC clock

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

Bits 4:3 **LSEDRV[1:0]:** LSE oscillator drive capability

Set by software to modulate the LSE oscillator's drive capability.

00: Low driving capability

01: Medium high driving capability

10: Medium low driving capability

11: High driving capability

Bit 2 **LSEBYP:** External low-speed oscillator bypass

This bit is set and cleared by software to bypass the oscillator. This bit can be written only when the LSE clock is disabled.

0: LSE oscillator not bypassed

1: LSE oscillator bypassed

Bit 1 **LSERDY:** External low-speed oscillator ready

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

0: LSE clock not ready

1: LSE clock ready

Bit 0 **LSEON:** External low-speed oscillator enable

This bit is set and cleared by software.

0: LSE clock OFF

1: LSE clock ON

5.3.21 RCC clock control & status register (RCC_CSR)

Address offset: 0x74

Reset value: 0x0E00 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	BOR RSTF	RMVF	Res.	Res.						
r	r	r	r	r	r	r	rw								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	LSIRDY	LSION
														r	rw

Bit 31 LPWRRSTF: Low-power reset flag

This bit is set by hardware when a Low-power management reset occurs.

Cleared by writing to the RMVF bit.

0: No Low-power management reset occurred

1: Low-power management reset occurred

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

Bit 30 WWDGRSTF: Window watchdog reset flag

This bit is set by hardware when a window watchdog reset occurs.

Cleared by writing to the RMVF bit.

0: No window watchdog reset occurred

1: Window watchdog reset occurred

Bit 29 IWDRSTF: Independent watchdog reset flag

This bit is set by hardware when an independent watchdog reset from V_{DD} domain occurs.

Cleared by writing to the RMVF bit.

0: No watchdog reset occurred

1: Watchdog reset occurred

Bit 28 SFRSTF: Software reset flag

This bit is set by hardware when a software reset occurs.

Cleared by writing to the RMVF bit.

0: No software reset occurred

1: Software reset occurred

Bit 27 PORRSTF: POR/PDR reset flag

This bit is set by hardware when a POR/PDR reset occurs.

Cleared by writing to the RMVF bit.

0: No POR/PDR reset occurred

1: POR/PDR reset occurred

Bit 26 PINRSTF: PIN reset flag

This bit is set by hardware when a reset from the NRST pin occurs.

Cleared by writing to the RMVF bit.

0: No reset from NRST pin occurred

1: Reset from NRST pin occurred

Bit 25 BORRSTF: BOR reset flag

Cleared by software by writing the RMVF bit.

This bit is set by hardware when a POR/PDR or BOR reset occurs.

0: No POR/PDR or BOR reset occurred

1: POR/PDR or BOR reset occurred

Bit 24 RMVF: Remove reset flag

This bit is set by software to clear the reset flags.

0: No effect

1: Clear the reset flags

Bits 23:2 Reserved, must be kept at reset value.

Bit 1 LSIRDY: Internal low-speed oscillator ready

This bit is set and cleared by hardware to indicate when the internal RC 40 kHz oscillator is stable. After the LSION bit is cleared, LSIRDY goes low after 3 LSI clock cycles.

0: LSI RC oscillator not ready

1: LSI RC oscillator ready

Bit 0 LSION: Internal low-speed oscillator enable

This bit is set and cleared by software.

0: LSI RC oscillator OFF

1: LSI RC oscillator ON

5.3.22 RCC spread spectrum clock generation register (RCC_SSCGR)

Address offset: 0x80

Reset value: 0x0000 0000

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

The spread spectrum clock generation is available only for the main PLL.

The RCC_SSCGR register must be written either before the main PLL is enabled or after the main PLL disabled.

Note: For full details about PLL spread spectrum clock generation (SSCG) characteristics, refer to the “Electrical characteristics” section in your device datasheet.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SSCG EN	SPR EAD SEL	Res.	Res.	INCSTEP											
rw	rw			rw	rw	rw		rw							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INCSTEP				MODPER											
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 SSCGEN: Spread spectrum modulation enable

This bit is set and cleared by software.

0: Spread spectrum modulation DISABLE. (To write after clearing CR[24]=PLLON bit)

1: Spread spectrum modulation ENABLE. (To write before setting CR[24]=PLLON bit)

Bit 30 SPREADSEL: Spread Select

This bit is set and cleared by software.

To write before to set CR[24]=PLLON bit.

0: Center spread

1: Down spread

Bits 29:28 Reserved, must be kept at reset value.

Bits 27:13 **INCSTEP**: Incrementation step

These bits are set and cleared by software. To write before setting CR[24]=PLLON bit.
Configuration input for modulation profile amplitude.

Bits 12:0 **MODPER**: Modulation period

These bits are set and cleared by software. To write before setting CR[24]=PLLON bit.
Configuration input for modulation profile period.

5.3.23 RCC PLLI2S configuration register (RCC_PLLI2SCFGR)

Address offset: 0x84

Reset value: 0x2400 3000

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

This register is used to configure the PLLI2S clock outputs according to the formulas:

$$f_{(VCO\ clock)} = f_{(PLL\ clock\ input)} \times (PLL\ SN / PLL\ M)$$

$$f_{(PLL\ S_P)} = f_{(VCO\ clock)} / PLL\ SP$$

$$f_{(PLL\ S_Q)} = f_{(VCO\ clock)} / PLL\ SQ$$

$$f_{(PLL\ S_R)} = f_{(VCO\ clock)} / PLL\ SR$$

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

Bit 31 Reserved, must be kept at reset value.

Bits 30:28 **PLL12SR[2:0]**: PLLI2S division factor for I2S clocks

These bits are set and cleared by software to control the I2S clock frequency. These bits should be written only if the PLLI2S is disabled. The factor must be chosen in accordance with the prescaler values inside the I2S peripherals, to reach 0.3% error when using standard crystals and 0% error with audio crystals. For more information about I2S clock frequency and precision, refer to [Section 28.7.4: Start-up description](#) in the I2S chapter.

Caution: The I2Ss requires a frequency lower than or equal to 192 MHz to work correctly.

I2S clock frequency = VCO frequency / PLLR with $2 \leq \text{PLLR} \leq 7$

000: PLLR = 0, wrong configuration

001: PLLR = 1, wrong configuration

010: PLLR = 2

...

111: PLLR = 7

Bits 27:24 **PLL12SQ[3:0]**: PLLI2S division factor for SAIs clock

These bits are set and cleared by software to control the SAIs clock frequency.

They should be written when the PLLI2S is disabled.

SAI clock frequency = VCO frequency / PLLI2SQ with $2 \leq \text{PLLI2SQ} \leq 15$

0000: PLLI2SQ = 0, wrong configuration

0001: PLLI2SQ = 1, wrong configuration

0010: PLLI2SQ = 2

0011: PLLI2SQ = 3

0100: PLLI2SQ = 4

0101: PLLI2SQ = 5

...

1111: PLLI2SQ = 15

Bits 23:16 Reserved, must be kept at reset value.

Bit 15 Reserved, must be kept at reset value.

Bits 14:6 **PLL2SN[8:0]**: PLLI2S multiplication factor for VCO

These bits are set and cleared by software to control the multiplication factor of the VCO. These bits can be written only when the PLLI2S is disabled. Only half-word and word accesses are allowed to write these bits.

Caution: The software has to set these bits correctly to ensure that the VCO output frequency is between 100 and 432 MHz.

VCO output frequency = VCO input frequency \times PLLI2SN with $50 \leq \text{PLLI2SN} \leq 432$

00000000: PLLI2SN = 0, wrong configuration

00000001: PLLI2SN = 1, wrong configuration

...

001100010: PLLI2SN = 50

...

001100011: PLLI2SN = 99

001100100: PLLI2SN = 100

001100101: PLLI2SN = 101

001100110: PLLI2SN = 102

...

110110000: PLLI2SN = 432

110110000: PLLI2SN = 433, wrong configuration

...

111111111: PLLI2SN = 511, wrong configuration

Note: Between 50 and 99, multiplication factors are possible for VCO input frequency higher than 1 MHz. However care must be taken to fulfill the minimum VCO output frequency as specified above.

Bits 5:0 Reserved, must be kept at reset value.

5.3.24 RCC PLLSAI configuration register (RCC_PLLSAICFGR)

Address offset: 0x88

Reset value: 0x2400 3000

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

This register is used to configure the PLLSAI clock outputs according to the formulas:

- $f_{(VCO\ clock)} = f_{(PLL\ clock\ input)} \times (PLL\ SAIN / PLLM)$
- $f_{(PLL\ SAI_P)} = f_{(VCO\ clock)} / PLL\ SAIP$
- $f_{(PLL\ SAI_Q)} = f_{(VCO\ clock)} / PLL\ SAIQ$

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

Bits 31:28 Reserved, must be kept at reset value.

Bits 27:24 **PLLSAIQ[3:0]:** PLLSAI division factor for SAI clock

Set and reset by software to control the frequency of SAI clock.

These bits should be written when the PLLSAI is disabled.

SAI1 clock frequency = VCO frequency / PLLSAIQ with $2 \leq \text{PLLSAIQ} \leq 15$

0000: PLLSAIQ = 0, wrong configuration

0001: PLLSAIQ = 1, wrong configuration

...

0010: PLLSAIQ = 2

0011: PLLSAIQ = 3

0100: PLLSAIQ = 4

0101: PLLSAIQ = 5

...

1111: PLLSAIQ = 15

Bits 23:18 Reserved, must be kept at reset value.

Bits 17:16 **PLLSAIP[1:0]:** PLLSAI division factor for 48MHz clock

Set and reset by software to control the frequency of the PLLSAI output clock

(PLLSAI48CLK). This output can be selected for USB, RNG, SDMMC1/2 (48 MHz clock).

These bits should be written only if the PLLSAI is disabled.

Only half-word and word accesses are allowed to write these bits.

PLLSAI48 output clock frequency = VCO frequency / PLLSAIP with PLLSAI P = 2, 4, 6, or 8

00: PLLSAIP = 2

01: PLLSAIP = 4

10: PLLSAIP = 6

11: PLLSAIP = 8

Bit 15 Reserved, must be kept at reset value.

Bits 14:6 **PLLSAIN[8:0]**: PLLSAI division factor for VCO

Set and reset by software to control the multiplication factor of the VCO.

These bits should be written when the PLLSAI is disabled.

Only half-word and word accesses are allowed to write these bits.

Caution: The software has to set these bits correctly to ensure that the VCO output frequency is between 100 and 432 MHz.

VCO output frequency = VCO input frequency x PLLSAIN with $50 \leq \text{PLLSAIN} \leq 432$

00000000: PLLSAIN = 0, wrong configuration

00000001: PLLSAIN = 1, wrong configuration

.....

001100010: PLLSAIN = 50

.....

001100011: PLLSAIN = 99

001100100: PLLSAIN = 100

001100101: PLLSAIN = 101

001100110: PLLSAIN = 102

.....

110110000: PLLSAIN = 432

110110000: PLLSAIN = 433, wrong configuration

.....

111111111: PLLSAIN = 511, wrong configuration

Note: Between 50 and 99, multiplication factors are possible for VCO input frequency higher than 1 MHz. However care must be taken to fulfill the minimum VCO output frequency as specified above.

Bits 5:0 Reserved, must be kept at reset value

5.3.25 RCC dedicated clocks configuration register (RCC_DKCFGR1)

Address offset: 0x8C

Reset value: 0x0000 0000

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

This register allows to configure the timer clock prescalers and the PLLSAI and PLLI2S output clock dividers for SAIs peripheral according to the following formula:

$$f_{(\text{PLLSAIDIVQ clock output})} = f_{(\text{PLLSAI_Q})} / \text{PLLSAIDIVQ}$$

$$f_{(\text{PLLI2SDIVQ clock output})} = f_{(\text{PLLI2S_Q})} / \text{PLLI2SDIVQ}$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	TIMPRE	SAI2SEL[1:0]		SAI1SEL[1:0]		Res.	Res.	Res.	Res.
							rw	rw	rw	rw	rw				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	PLLSAIDIVQ[4:0]					Res.	Res.	Res.	PLLI2SDIVQ[4:0]				
			rw	rw	rw	rw	rw				rw	rw	rw	rw	rw

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

Bit 24 **TIMPRE**: Timers clocks prescalers selection

This bit is set and reset by software to control the clock frequency of all the timers connected to APB1 and APB2 domain.

0: If the APB prescaler (PPRE1, PPRE2 in the RCC_CFGR register) is configured to a division factor of 1, TIMxCLK = PCLKx. Otherwise, the timer clock frequencies are set to twice to the frequency of the APB domain to which the timers are connected:

$\text{TIMxCLK} = 2 \times \text{PCLKx}$.

1: If the APB prescaler (PPRE1, PPRE2 in the RCC_CFGR register) is configured to a division factor of 1, 2 or 4, TIMxCLK = HCLK. Otherwise, the timer clock frequencies are set to four times to the frequency of the APB domain to which the timers are connected:

$\text{TIMxCLK} = 4 \times \text{PCLKx}$.

Bits 23:22 **SAI2SEL[1:0]**: SAI2 clock source selection:

These bits are set and cleared by software to control the SAI2 clock frequency.

They should be written when the PLLSAI and PLLI2S are disabled.

00: SAI2 clock frequency = $f(\text{PLLSAI_Q}) / \text{PLLSAIDIVQ}$

01: SAI2 clock frequency = $f(\text{PLLI2S_Q}) / \text{PLLI2SDIVQ}$

10: SAI2 clock frequency = Alternate function input frequency

11: wrong configuration

Bits 21:20 **SAI1SEL[1:0]**: SAI1 clock source selection

These bits are set and cleared by software to control the SAI1 clock frequency.

They should be written when the PLLSAI and PLLI2S are disabled.

00: SAI1 clock frequency = $f(\text{PLLSAI_Q}) / \text{PLLSAIDIVQ}$

01: SAI1 clock frequency = $f(\text{PLLI2S_Q}) / \text{PLLI2SDIVQ}$

10: SAI1 clock frequency = Alternate function input frequency

11: wrong configuration

Bits 19: 13 Reserved, must be kept at reset value.

Bits 12:8 **PLLSAIDIVQ[4:0]**: PLLSAI division factor for SAI1 clock

These bits are set and reset by software to control the SAI1 clock frequency.

They should be written only if PLLSAI is disabled.

SAI1 clock frequency = $f(\text{PLLSAI_Q}) / \text{PLLSAIDIVQ}$ with $1 \leq \text{PLLSAIDIVQ} \leq 31$

00000: PLLSAIDIVQ = /1

00001: PLLSAIDIVQ = /2

00010: PLLSAIDIVQ = /3

00011: PLLSAIDIVQ = /4

00100: PLLSAIDIVQ = /5

...

11111: PLLSAIDIVQ = /32

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

Bits 4:0 **PLL2SDIV[4:0]**: PLLI2S division factor for SAI1 clock

These bits are set and reset by software to control the SAI1 clock frequency.

They should be written only if PLLI2S is disabled.

SAI1 clock frequency = $f(\text{PLLI2S_Q}) / \text{PLLI2SDIVQ}$ with $1 \leq \text{PLLI2SDIVQ} \leq 31$

00000: PLLI2SDIVQ = /1

00001: PLLI2SDIVQ = /2

00010: PLLI2SDIVQ = /3

00011: PLLI2SDIVQ = /4

00100: PLLI2SDIVQ = /5

...

11111: PLLI2SDIVQ = /32

5.3.26 RCC dedicated clocks configuration register (DCKCFGR2)

Address: 0x90h

Reset value: 0x0000 0000h

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

This register allows to select the source clock for the 48MHz, SDMMC1/2, LPTIM1, UARTs, USARTs and I2Cs clocks.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	SDMMC2 SEL	SDMMC1 SEL	CK48M SEL	Res.	LPTIM1SEL	Res.	Res.	I2C3SEL	I2C2SEL	I2C1SEL				
		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
UART8SEL		UART7SEL		USART6SEL		UART5SEL		UART4SEL		UART3SEL		UART2SEL		UART1SEL	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30 Reserved, must be kept at reset value.

Bit 29 **SDMMC2SEL**: SDMMC2 clock source selection

Set and reset by software.

0: 48 MHz clock is selected as SDMMC2 clock

1: System clock is selected as SDMMC2 clock

Bit 28 **SDMMC1SEL**: SDMMC1 clock source selection

Set and reset by software.

0: 48 MHz clock is selected as SDMMC1 clock

1: System clock is selected as SDMMC1 clock

Bit 27 **CK48MSEL**: 48MHz clock source selection

Set and reset by software.

0: 48MHz clock from PLL is selected

1: 48MHz clock from PLLSAI is selected.

Bit 26 Reserved, must be kept at reset value.

Bits 25:24 **LPTIM1SEL**: Low-power timer 1 clock source selection

Set and reset by software.

00: APB1 clock (PCLK1) selected as LPTILM1 clock

01: LSI clock is selected as LPTILM1 clock

10: HSI clock is selected as LPTILM1 clock

11: LSE clock is selected as LPTILM1 clock

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

Bits 21:20 **I2C3SEL**: I2C3 clock source selection

Set and reset by software.

00: APB clock is selected as I2C3 clock

01: System clock is selected as I2C3 clock

10: HSI clock is selected as I2C3 clock

11: reserved

Bits 19:18 **I2C2SEL**: I2C2 clock source selection

Set and reset by software.

00: APB1 clock (PCLK1) is selected as I2C2 clock

01: System clock is selected as I2C2 clock

10: HSI clock is selected as I2C2 clock

11: reserved

Bits 17:16 **I2C1SEL**: I2C1 clock source selection

Set and reset by software.

00: APB clock (PCLK1) is selected as I2C1 clock

01: System clock is selected as I2C1 clock

10: HSI clock is selected as I2C1 clock

11: reserved

Bits 15:14 **UART8SEL[1:0]**: UART 8 clock source selection

Set and reset by software.

00: APB1 clock (PCLK1) is selected as UART 8 clock

01: System clock is selected as UART 8 clock

10: HSI clock is selected as UART 8 clock

11: LSE clock is selected as UART 8 clock

Bits 13:12 **UART7SEL[1:0]**: UART 7 clock source selection

Set and reset by software.

- 00: APB1 clock (PCLK1) is selected as UART 7 clock
- 01: System clock is selected as UART 7 clock
- 10: HSI clock is selected as UART 7 clock
- 11: LSE clock is selected as UART 7 clock

Bits 11:10 **USART6SEL[1:0]**: USART 6 clock source selection

Set and reset by software.

- 00: APB2 clock(PCLK2) is selected as USART 6 clock
- 01: System clock is selected as USART 6 clock
- 10: HSI clock is selected as USART 6 clock
- 11: LSE clock is selected as USART 6 clock

Bits 9:8 **UART5SEL[1:0]**: UART 5 clock source selection

Set and reset by software.

- 00: APB1 clock(PCLK1) is selected as UART 5 clock
- 01: System clock is selected as UART 5 clock
- 10: HSI clock is selected as UART 5 clock
- 11: LSE clock is selected as UART 5 clock

Bits 7:6 **UART4SEL[1:0]**: UART 4 clock source selection

Set and reset by software.

- 00: APB1 clock (PLCLK1) is selected as UART 4 clock
- 01: System clock is selected as UART 4 clock
- 10: HSI clock is selected as UART 4 clock
- 11: LSE clock is selected as UART 4 clock

Bits 5:4 **USART3SEL[1:0]**: USART 3 clock source selection

Set and reset by software.

- 00: APB1 clock (PCLK1) is selected as USART 3 clock
- 01: System clock is selected as USART 3 clock
- 10: HSI clock is selected as USART 3 clock
- 11: LSE clock is selected as USART 3 clock

Bits 3:2 **USART2SEL[1:0]**: USART 2 clock source selection

Set and reset by software.

- 00: APB1 clock (PCLK1) is selected as USART 2 clock
- 01: System clock is selected as USART 2 clock
- 10: HSI clock is selected as USART 2 clock
- 11: LSE clock is selected as USART 2 clock

Bits 1:0 **USART1SEL[1:0]**: USART 1 clock source selection

Set and reset by software.

- 00: APB2 clock (PCLK2) is selected as USART 1 clock
- 01: System clock is selected as USART 1 clock
- 10: HSI clock is selected as USART 1 clock
- 11: LSE clock is selected as USART 1 clock

5.3.27 RCC register map

[Table 22](#) gives the register map and reset values.

Table 22. RCC register map and reset values

Table 22. RCC register map and reset values (continued)

Table 22. RCC register map and reset values (continued)

Addr. offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x80	RCC_SSCGR	SSCGEN	SPREADSEL																														
0x84	RCC_PLLI2SC_FGR	PLL12SR[2:0]	PLL12SQ[3:0]	PLL12AQ[4:0]	PLL12SN[8:0]	PLL12SDIVQ[4:0]	PLL12SEL																										
0x88	RCC_PLLSAI_CFGR	PLLSAIQ[4:0]	PLLSAIP[1:0]	PLLSAIN[8:0]	PLLSAIDIVQ[4:0]	PLL2SDIVQ[4:0]	PLL2SEL																										
0x8C	RCC_DCKCF_GR1	LPTIM1SEL	TIMPRE	SAI1SEL[1:0]	I2C3SEL	I2C2SEL	UART8SEL	UART7SEL	UART6SEL	UART5SEL	UART4SEL	UART3SEL	UART2SEL	UART1SEL	Res.																		
0x90	RCC_DCKCF_GR2	SDMMC2SEL	SDMMC1SEL	CK4MSEL	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

6 General-purpose I/Os (GPIO)

6.1 Introduction

Each general-purpose I/O port has four 32-bit configuration registers (GPIO_x_MODER, GPIO_x_OTYPER, GPIO_x_OSPEEDR and GPIO_x_PUPDR), two 32-bit data registers (GPIO_x_IDR and GPIO_x_ODR) and a 32-bit set/reset register (GPIO_x_BSRR). In addition all GPIOs have a 32-bit locking register (GPIO_x_LCKR) and two 32-bit alternate function selection registers (GPIO_x_AFRH and GPIO_x_AFRL).

6.2 GPIO main features

- Output states: push-pull or open drain + pull-up/down
- Output data from output data register (GPIO_x_ODR) or peripheral (alternate function output)
- Speed selection for each I/O
- Input states: floating, pull-up/down, analog
- Input data to input data register (GPIO_x_IDR) or peripheral (alternate function input)
- Bit set and reset register (GPIO_x_BSRR) for bitwise write access to GPIO_x_ODR
- Locking mechanism (GPIO_x_LCKR) provided to freeze the I/O port configurations
- Analog function
- Alternate function selection registers
- Fast toggle capable of changing every two clock cycles
- Highly flexible pin multiplexing allows the use of I/O pins as GPIOs or as one of several peripheral functions

6.3 GPIO functional description

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

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

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

Figure 18 and *Figure 19* shows the basic structures of a standard and a 5-Volt tolerant I/O port bit, respectively. *Table 23* gives the possible port bit configurations.

Figure 18. Basic structure of an I/O port bit

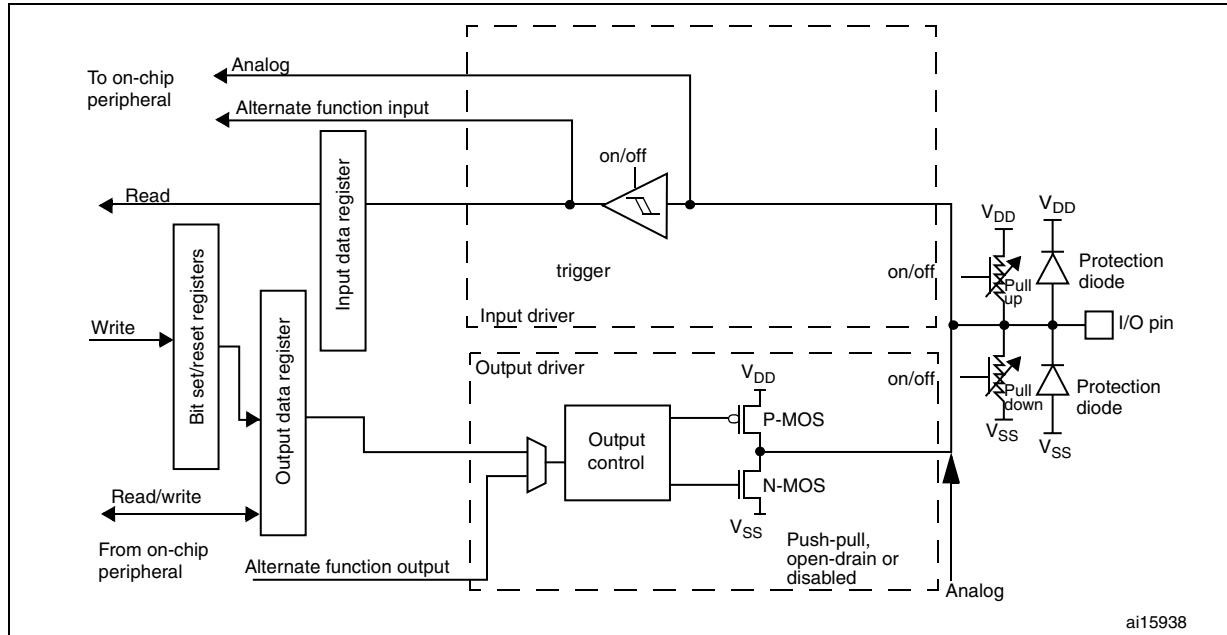
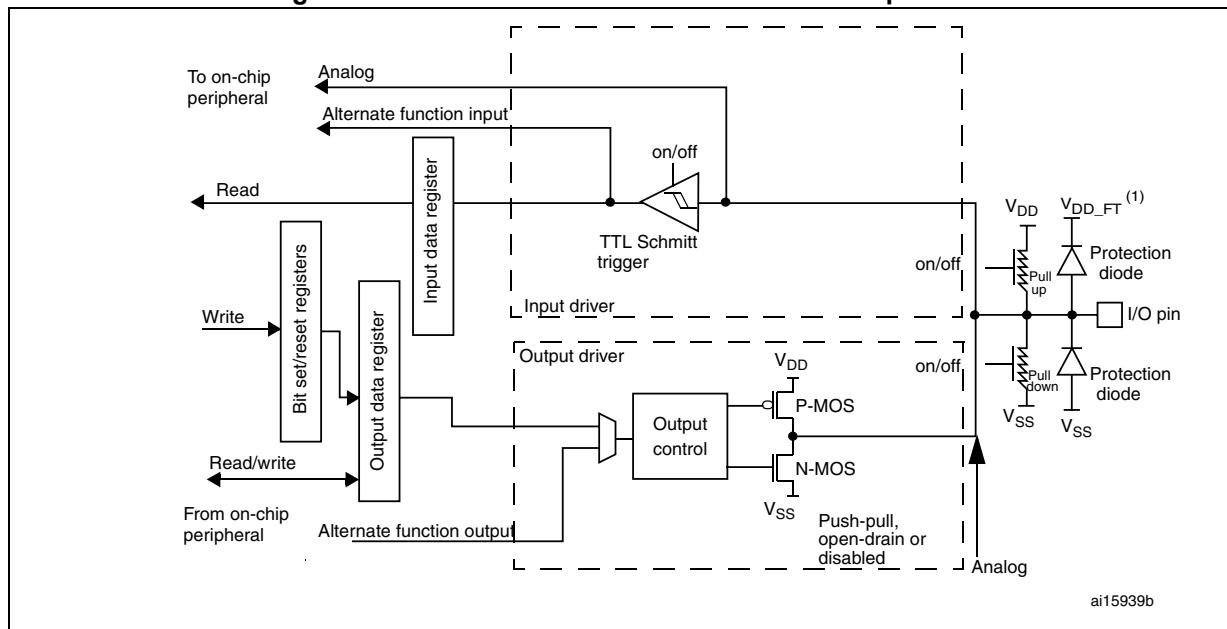


Figure 19. Basic structure of a 5-Volt tolerant I/O port bit



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

Table 23. Port bit configuration table⁽¹⁾

MODE(i) [1:0]	OTYPER(i)	OSPEED(i) [1:0]	PUPD(i) [1:0]		I/O configuration	
01	0	SPEED [1:0]	0	0	GP output	PP
	0		0	1	GP output	PP + PU
	0		1	0	GP output	PP + PD
	0		1	1	Reserved	
	1		0	0	GP output	OD
	1		0	1	GP output	OD + PU
	1		1	0	GP output	OD + PD
	1		1	1	Reserved (GP output OD)	
10	0	SPEED [1:0]	0	0	AF	PP
	0		0	1	AF	PP + PU
	0		1	0	AF	PP + PD
	0		1	1	Reserved	
	1		0	0	AF	OD
	1		0	1	AF	OD + PU
	1		1	0	AF	OD + PD
	1		1	1	Reserved	
00	x	x	x	0	0	Input
	x	x	x	0	1	Input
	x	x	x	1	0	Input
	x	x	x	1	1	Reserved (input floating)
11	x	x	x	0	0	Input/output
	x	x	x	0	1	Reserved
	x	x	x	1	0	
	x	x	x	1	1	

1. GP = general-purpose, PP = push-pull, PU = pull-up, PD = pull-down, OD = open-drain, AF = alternate function.

6.3.1 General-purpose I/O (GPIO)

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

The debug pins are in AF pull-up/pull-down after reset:

- PA15: JTDI in pull-up
- PA14: JTCK/SWCLK in pull-down
- PA13: JTMS/SWDAT in pull-up
- PB4: NJTRST in pull-up
- PB3: JTDO in floating state

When the pin is configured as output, the value written to the output data register (GPIOx_ODR) is output on the I/O pin. It is possible to use the output driver in push-pull mode or open-drain mode (only the low level is driven, high level is HI-Z).

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

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

6.3.2 I/O pin alternate function multiplexer and mapping

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

Each I/O pin has a multiplexer with up to sixteen alternate function inputs (AF0 to AF15) that can be configured through the GPIOx_AFRL (for pin 0 to 7) and GPIOx_AFRH (for pin 8 to 15) registers:

- After reset the multiplexer selection is alternate function 0 (AF0). The I/Os are configured in alternate function mode through GPIOx_MODER register.
- The specific alternate function assignments for each pin are detailed in the device datasheet.
- Cortex-M7 with FPU EVENTOUT is mapped on AF15

In addition to this flexible I/O multiplexing architecture, each peripheral has alternate functions mapped onto different I/O pins to optimize the number of peripherals available in smaller packages.

To use an I/O in a given configuration, the user has to proceed as follows:

- **Debug function:** after each device reset these pins are assigned as alternate function pins immediately usable by the debugger host
- **System function:** MCOx pins have to be configured in alternate function mode.
- **GPIO:** configure the desired I/O as output, input or analog in the GPIOx_MODER register.
- **Peripheral alternate function:**
 - Connect the I/O to the desired AFx in one of the GPIOx_AFRL or GPIOx_AFRH register.
 - Select the type, pull-up/pull-down and output speed via the GPIOx_OTYPER, GPIOx_PUPDR and GPIOx_OSPEEDER registers, respectively.

- Configure the desired I/O as an alternate function in the GPIOx_MODER register.
- **Additional functions:**
 - For the ADC and DAC, configure the desired I/O in analog mode in the GPIOx_MODER register and configure the required function in the ADC and DAC registers. For the additional functions like RTC_OUT, RTC_TS, RTC_TAMPx, WKUPx and oscillators, configure the required function in the related RTC, PWR and RCC registers. These functions have priority over the configuration in the standard GPIO registers. For details about I/O control by the RTC, refer to [Section 25.3: RTC functional description on page 777](#).
- EVENTOUT
 - Configure the I/O pin used to output the core EVENTOUT signal by connecting it to AF15.

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

6.3.3 I/O port control registers

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

6.3.4 I/O port data registers

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

See [Section 6.4.5: GPIO port input data register \(GPIOx_IDR\) \(x = A to K\)](#) and [Section 6.4.6: GPIO port output data register \(GPIOx_ODR\) \(x = A to K\)](#) for the register descriptions.

6.3.5 I/O data bitwise handling

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

To each bit in GPIOx_ODR, correspond two control bits in GPIOx_BSRR: BS(i) and BR(i). When written to 1, bit BS(i) **sets** the corresponding ODR(i) bit. When written to 1, bit BR(i) **resets** the ODR(i) corresponding bit.

Writing any bit to 0 in GPIOx_BSRR does not have any effect on the corresponding bit in GPIOx_ODR. If there is an attempt to both set and reset a bit in GPIOx_BSRR, the set action takes priority.

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

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

6.3.6 GPIO locking mechanism

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

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

The LOCK sequence (refer to [Section 6.4.8: GPIO port configuration lock register \(GPIOx_LCKR\) \(x = A to K\)](#)) can only be performed using a word (32-bit long) access to the GPIOx_LCKR register due to the fact that GPIOx_LCKR bit 16 has to be set at the same time as the [15:0] bits.

For more details refer to LCKR register description in [Section 6.4.8: GPIO port configuration lock register \(GPIOx_LCKR\) \(x = A to K\)](#).

6.3.7 I/O alternate function input/output

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

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

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

6.3.8 External interrupt/wakeup lines

All ports have external interrupt capability. To use external interrupt lines, the port must be configured in input mode. Refer to [Section 11: Extended interrupts and events controller \(EXTI\)](#)[Section 11.3: Wakeup event management](#).

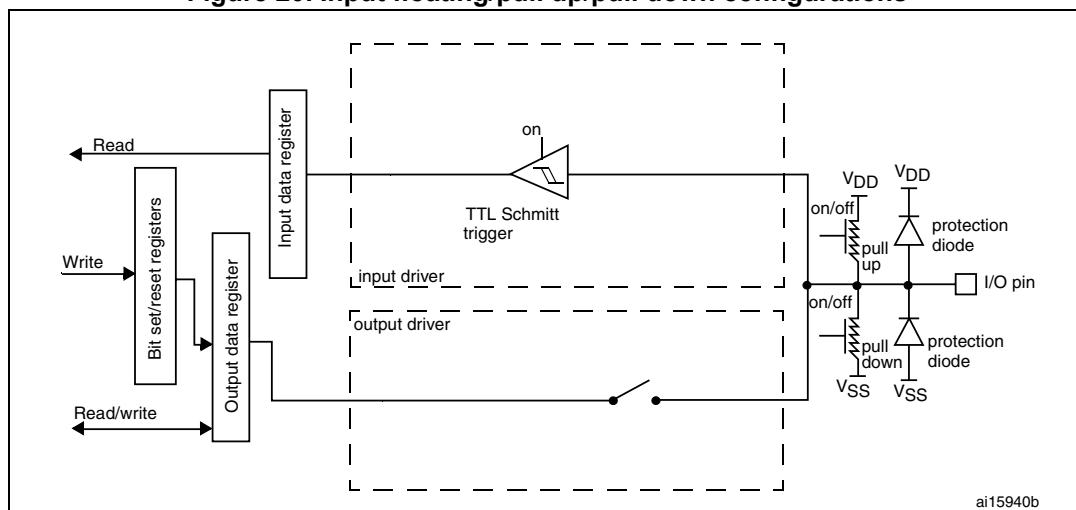
6.3.9 Input configuration

When the I/O port is programmed as input:

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

Figure 20 shows the input configuration of the I/O port bit.

Figure 20. Input floating/pull up/pull down configurations



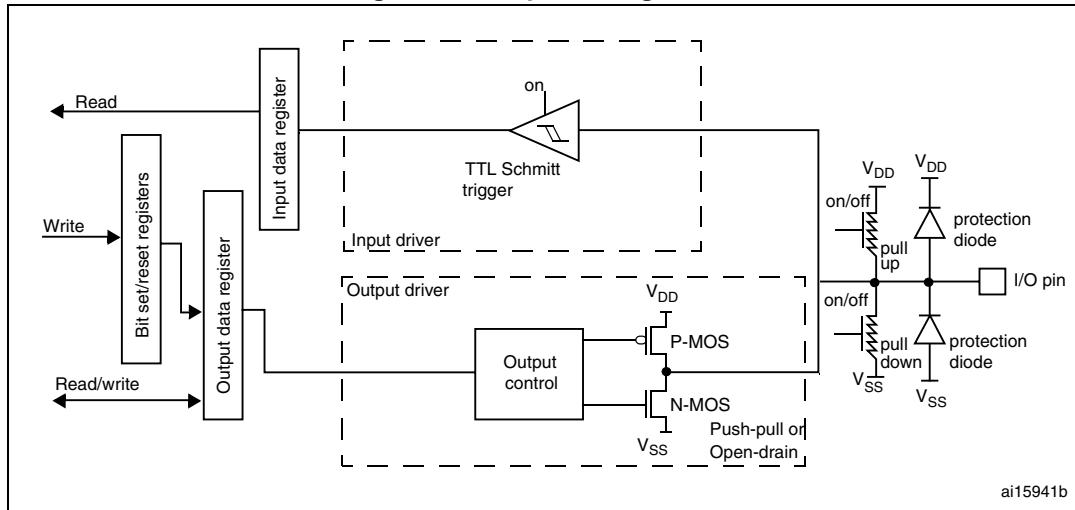
6.3.10 Output configuration

When the I/O port is programmed as output:

- The output buffer is enabled:
 - Open drain mode: A “0” in the Output register activates the N-MOS whereas a “1” in the Output register leaves the port in Hi-Z (the P-MOS is never activated)
 - Push-pull mode: A “0” in the Output register activates the N-MOS whereas a “1” in the Output register activates the P-MOS
- The Schmitt trigger input is activated
- The pull-up and pull-down resistors are activated depending on the value in the GPIOx_PUPDR register
- The data present on the I/O pin are sampled into the input data register every AHB clock cycle
- A read access to the input data register gets the I/O state
- A read access to the output data register gets the last written value

Figure 21 shows the output configuration of the I/O port bit.

Figure 21. Output configuration



ai15941b

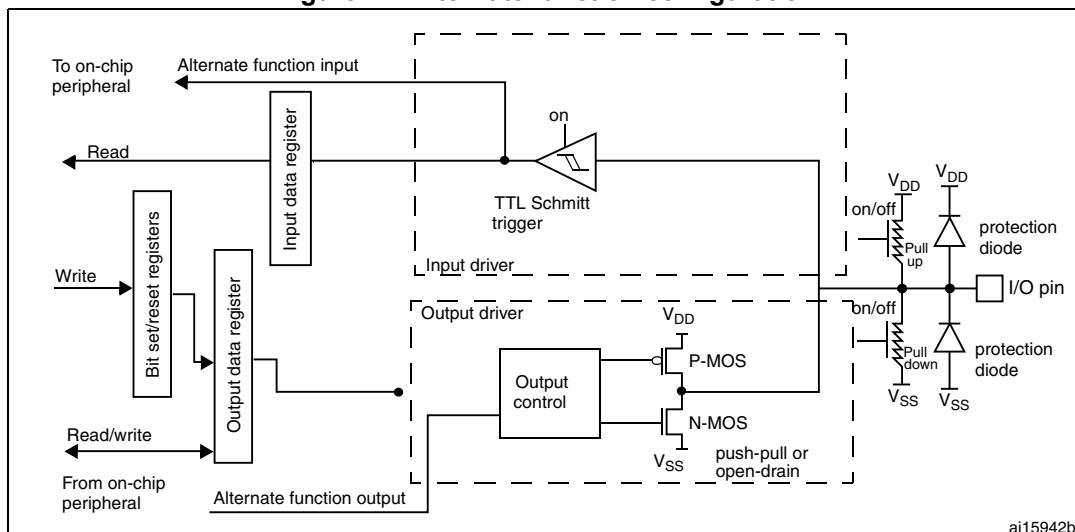
6.3.11 Alternate function configuration

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

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

Figure 22 shows the Alternate function configuration of the I/O port bit.

Figure 22. Alternate function configuration



ai15942b

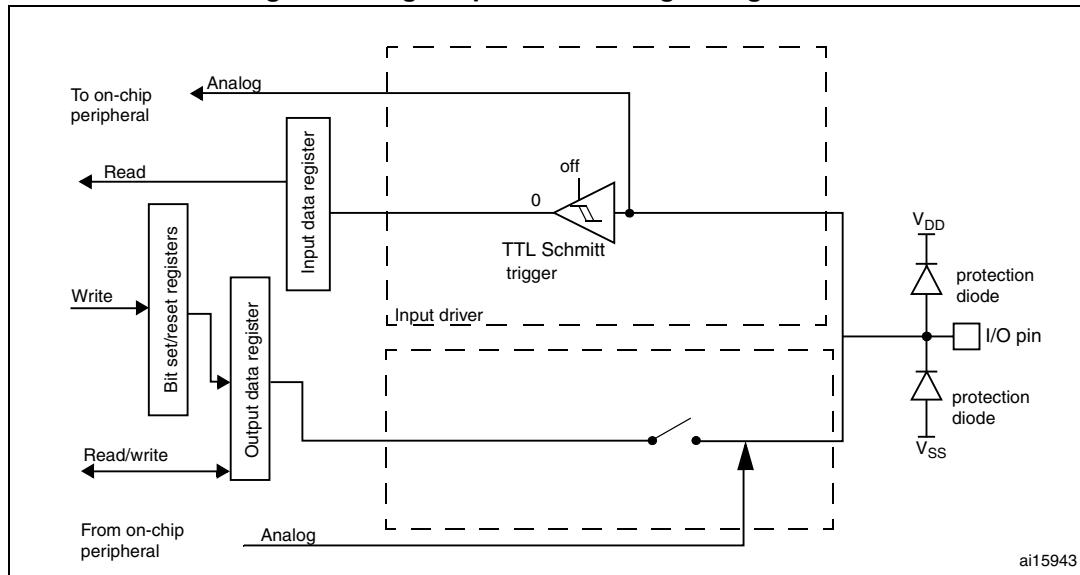
6.3.12 Analog configuration

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

- The output buffer is disabled
- The Schmitt trigger input is deactivated, providing zero consumption for every analog value of the I/O pin. The output of the Schmitt trigger is forced to a constant value (0).
- The weak pull-up and pull-down resistors are disabled by hardware
- Read access to the input data register gets the value “0”

Figure 23 shows the high-impedance, analog-input configuration of the I/O port bits.

Figure 23. High impedance-analog configuration



6.3.13 Using the HSE or LSE oscillator pins as GPIOs

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

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

When the oscillator is configured in a user external clock mode, only the OSC_IN or OSC32_IN pin is reserved for clock input and the OSC_OUT or OSC32_OUT pin can still be used as normal GPIO.

6.3.14 Using the GPIO pins in the backup supply domain

The PC13/PC14/PC15/PI8 GPIO functionality is lost when the core supply domain is powered off (when the device enters Standby mode). In this case, if their GPIO configuration is not bypassed by the RTC configuration, these pins are set in an analog input mode.

6.4 GPIO registers

This section gives a detailed description of the GPIO registers.

For a summary of register bits, register address offsets and reset values, refer to [Table 24](#).

The peripheral registers can be written in word, half word or byte mode.

6.4.1 GPIO port mode register (GPIOx_MODER) (x =A to K)

Address offset: 0x00

Reset value:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODERO[1:0]	
rw	rw	rw	rw	rw	rw										

Bits 31:0 **MODER[15:0][1:0]**: Port x configuration I/O pin y (y = 15 to 0)

These bits are written by software to configure the I/O mode.

- 00: Input mode (reset state)
- 01: General purpose output mode
- 10: Alternate function mode
- 11: Analog mode

6.4.2 GPIO port output type register (GPIOx_OTYPER) (x = A to K)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw															

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

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

These bits are written by software to configure the I/O output type.

- 0: Output push-pull (reset state)
- 1: Output open-drain

6.4.3 GPIO port output speed register (GPIOx_OSPEEDR) (x = A to K)

Address offset: 0x08

Reset value: 0x0C00 0000 (for port A)

Reset value: 0x0000 00C0 (for port B)

Reset value: 0x0000 0000 (for other ports)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [1:0]	
rw	rw	rw	rw	rw	rw										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSPEEDR7 [1:0]		OSPEEDR6 [1:0]		OSPEEDR5 [1:0]		OSPEEDR4 [1:0]		OSPEEDR3 [1:0]		OSPEEDR2 [1:0]		OSPEEDR1 [1:0]		OSPEEDR0 [1:0]	
rw	rw	rw	rw	rw	rw										

Bits 31:0 **OSPEEDR[15:0][1:0]**: Port x configuration I/O pin y (y = 15 to 0)

These bits are written by software to configure the I/O output speed.

00: Low speed

01: Medium speed

10: High speed

11: Very high speed

Note: Refer to the product datasheets for the values of OSPEEDRy bits versus V_{DD} range and external load.

6.4.4 GPIO port pull-up/pull-down register (GPIOx_PUPDR) (x = A to K)

Address offset: 0x0C

Reset value: 0x6400 0000 (for port A)

Reset value: 0x0000 0100 (for port B)

Reset value: 0x0000 0000 (for other ports)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw										

Bits 31:0 **PUPDR[15:0][1:0]**: Port x configuration I/O pin y (y = 15 to 0)

These bits are written by software to configure the I/O pull-up or pull-down

00: No pull-up, pull-down

01: Pull-up

10: Pull-down

11: Reserved

6.4.5 GPIO port input data register (GPIOx_IDR) (x = A to K)

Address offset: 0x10

Reset value: 0x0000 XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

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

Bits 15:0 **IDR[15:0]**: Port x input data I/O pin y (y = 15 to 0)

These bits are read-only. They contain the input value of the corresponding I/O port.

6.4.6 GPIO port output data register (GPIOx_ODR) (x = A to K)

Address offset: 0x14

Reset value: 0x0000 0000

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

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

Bits 15:0 **ODR[15:0]**: Port output data I/O pin y (y = 15 to 0)

These bits can be read and written by software.

Note: For atomic bit set/reset, the ODR bits can be individually set and/or reset by writing to the GPIOx_BSRR or GPIOx_BRR registers (x = A..F).

6.4.7 GPIO port bit set/reset register (GPIOx_BSRR) (x = A to K)

Address offset: 0x18

Reset value: 0x0000 0000

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

Bits 31:16 **BR[15:0]**: Port x reset I/O pin y (y = 15 to 0)

These bits are write-only. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODR_x bit

1: Resets the corresponding ODR_x bit

Note: If both BS_x and BR_x are set, BS_x has priority.

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

These bits are write-only. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODR_x bit

1: Sets the corresponding ODR_x bit

6.4.8 GPIO port configuration lock register (GPIO_x_LCKR) (x = A to K)

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

Note: A specific write sequence is used to write to the GPIO_x_LCKR register. Only word access (32-bit long) is allowed during this locking sequence.

Each lock bit freezes a specific configuration register (control and alternate function registers).

Address offset: 0x1C

Reset value: 0x0000 0000

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

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

Bit 16 **LCKK:** Lock key

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

0: Port configuration lock key not active

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

LOCK key write sequence:

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

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

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

RD LCKR

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

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

Any error in the lock sequence aborts the lock.

After the first lock sequence on any bit of the port, any read access on the LCKK bit will return '1' until the next MCU reset or peripheral reset.

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

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

0: Port configuration not locked

1: Port configuration locked

6.4.9 GPIO alternate function low register (GPIOx_AFRL) (x = A to K)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFR7[3:0]				AFR6[3:0]				AFR5[3:0]				AFR4[3:0]			
rw	rw	rw	rw												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFR3[3:0]				AFR2[3:0]				AFR1[3:0]				AFR0[3:0]			
rw	rw	rw	rw												

Bits 31:0 **AFR[7:0][3:0]**: Alternate function selection for port x I/O pin y (y = 7 to 0)

These bits are written by software to configure alternate function I/Os.

0000: AF0
0001: AF1
0010: AF2
0011: AF3
0100: AF4
0101: AF5
0110: AF6
0111: AF7
1000: AF8
1001: AF9
1010: AF10
1011: AF11
1100: AF12
1101: AF13
1110: AF14
1111: AF15

6.4.10 GPIO alternate function high register (GPIOx_AFRH) (x = A to J)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFR15[3:0]				AFR14[3:0]				AFR13[3:0]				AFR12[3:0]			
rw	rw	rw	rw												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFR11[3:0]				AFR10[3:0]				AFR9[3:0]				AFR8[3:0]			
rw	rw	rw	rw												

Bits 31:0 **AFR[15:8][3:0]**: Alternate function selection for port x I/O pin y (y = 15 to 8)

These bits are written by software to configure alternate function I/Os.

0000: AF0
0001: AF1
0010: AF2
0011: AF3
0100: AF4
0101: AF5
0110: AF6
0111: AF7
1000: AF8
1001: AF9
1010: AF10
1011: AF11
1100: AF12
1101: AF13
1110: AF14
1111: AF15

6.4.11 GPIO register map

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

Table 24. GPIO register map and reset values

Offset	Register name	Reset value
0x00	GPIOA_MODER	31
	Reset value	0 MODER15[1:0] 1 0 MODER15[1:0]
0x00	GPIOB_MODER	30
	Reset value	0 MODER14[1:0] 1 0 MODER14[1:0]
0x00	GPIOx_MODER (where x = C..K)	29
	Reset value	0 MODER13[1:0] 1 0 MODER13[1:0]
0x04	GPIOx_OTYPER (where x = A..K)	28
	Reset value	0 MODER12[1:0] 0 0 MODER12[1:0]
0x08	GPIOA_OSPEEDR	27
	Reset value	0 MODER11[1:0] 0 0 MODER11[1:0]
0x08	GPIOB_OSPEEDR	26
	Reset value	0 MODER10[1:0] 0 0 MODER10[1:0]
0x08	GPIOx_OSPEEDR (where x = C..K)	25
	Reset value	0 MODER9[1:0] 0 0 MODER9[1:0]
0x0C	GPIOA_PUPDR	24
	Reset value	0 MODER8[1:0] 0 0 MODER8[1:0]

Table 24. GPIO register map and reset values (continued)

Refer to [Section 1.5](#) for the register boundary addresses.

7 System configuration controller (SYSCFG)

The system configuration controller is mainly used to:

- Remap the memory areas
- Manage the external interrupt line connection to the GPIOs.

7.1 I/O compensation cell

By default the I/O compensation cell is not used. However when the I/O output buffer speed is configured in 50 MHz or 100 MHz mode, it is recommended to use the compensation cell for slew rate control on I/O $t_{f(IO)out}/t_{r(IO)out}$ commutation to reduce the I/O noise on power supply.

When the compensation cell is enabled, a READY flag is set to indicate that the compensation cell is ready and can be used. The I/O compensation cell can be used only when the supply voltage ranges from 2.4 to 3.6 V.

7.2 SYSCFG registers

7.2.1 SYSCFG memory remap register (SYSCFG_MEMRMP)

This register is used for specific configurations on memory mapping:

- 1bit is used to indicate which option bytes BOOT_ADD0 or BOOT_ADD1 defines the boot memory base address.
- Other bits are used to swap FMC SDRAM Banks with FMC NOR/PSRAM bank

Address offset: 0x00

Reset value: 0x0000 0000

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

Bits 31:12 Reserved, must be kept at reset value.

Bits 11:10 **SWP_FMC[1:0]**: FMC memory mapping swap

Set and cleared by software. These bits are used to swap the FMC SDRAM banks and FMC NOR/PSRAM in order to enable the code execution from SDRAM Banks without modifying the default MPU attribute

00: No FMC memory mapping swapping

SDRAM bank1 and Bank2 are mapped at 0xC000 0000 and 0xD000 0000 respectively (default mapping)

NOR/RAM is accessible @ 0x60000000 (default mapping)

01: NOR/RAM and SDRAM memory mapping swapped,

SDRAM bank1 and bank2 are mapped at 0x6000 0000 and 0x7000 0000, respectively

NOR/PSRAM bank is mapped at 0xC000 0000

10: Reserved

11: Reserved

Bits 9:1 Reserved, must be kept at reset value.

Bits 0 **MEM_BOOT**: Memory boot mapping

This bit indicates which option bytes BOOT_ADD0 or BOOT_ADD1 defines the boot memory base address.

0: Boot memory base address is defined by BOOT_ADD0 option byte
(Factory Reset value: TCM-FLASH mapped at 0x00200000).

1: Boot memory base address is defined by BOOT_ADD1 option byte
(Factory Reset value: System memory mapped at 0x001 0000).

Note: Refer to section 2.3: Memory map for details about the boot memory base address selection.

7.2.2 SYSCFG peripheral mode configuration register (SYSCFG_PMC)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	ADCxDC2[2:0]								
rw															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	PB9_FMP	PB8_FMP	PB7_FMP	PB6_FMP	Res.	I2C3_FMP	I2C2_FMP	I2C1_FMP							
rw															

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

Bits 18:16 **ADCxDC2[2:0]**:

0: No effect.

1: Refer to AN4073 on how to use this bit.

Note: These bits can be set only if the following conditions are met:

- ADC clock higher or equal to 30 MHz.

- Only one ADCxDC2 bit must be selected if ADC conversions do not start at the same time and the sampling times differ.

- These bits must not be set when the ADCDC1 bit is set in PWR_CR register.

Bits 15:8 Reserved, must be kept at reset value.

Bit 7 **PB9_FMP**: Fast Mode + Enable

Set and cleared by software.

0: Default value.

1: It forces FM+ drive capability on PB9 pin

Bit 6 **PB8_FMP**: PB8_FMP Fast Mode + Enable

Set and cleared by software.

0: Default value.

1: It forces FM+ drive capability on PB8 pin

Bit 5 **PB7_FMP**: PB7_FMP Fast Mode + Enable

Set and cleared by software.

0: Default value.

1: It forces FM+ drive capability on PB7 pin

Bit 4 **PB6_FMP**: PB6_FMP Fast Mode + Enable

Set and cleared by software.

0: Default value.

1: It forces PB6 IO pads in Fast Mode +.

Bit 3 Reserved, must be kept at reset value.

Bit 2 **I2C3_FMP**: I2C3_FMP I2C3 Fast Mode + Enable

Set and cleared by software.

0: Default value.

1: It forces FM+ drive capability on I2C3 SCL & SDA pin selected through GPIO port mode register and GPIO alternate function selection bits

Bit 1 **I2C2_FMP**: I2C2_FMP I2C2 Fast Mode + Enable

Set and cleared by software.

0: Default value.

1: It forces FM+ drive capability on I2C2 SCL & SDA pin selected through GPIO port mode register and GPIO alternate function selection bits

Bit 0 **I2C1_FMP**: I2C1_FMP I2C1 Fast Mode + Enable

Set and cleared by software.

0: Default value.

1: It forces FM+ drive capability on I2C1 SCL & SDA pin selected through GPIO port mode register and GPIO alternate function selection bits

7.2.3 SYSCFG external interrupt configuration register 1 (SYSCFG_EXTICR1)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
rw	rw	rw	rw												

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

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

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

- 0000: PA[x] pin
- 0001: PB[x] pin
- 0010: PC[x] pin
- 0011: PD[x] pin
- 0100: PE[x] pin
- 0101: PF[x] pin
- 0110: PG[x] pin
- 0111: PH[x] pin
- 1000: PI[x] pin
- 1001: PJ[x] pin
- 1010: PK[x] pin

7.2.4 SYSCFG external interrupt configuration register 2 (SYSCFG_EXTICR2)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI7[3:0]				EXTI6[3:0]				EXTI5[3:0]				EXTI4[3:0]			
rw	rw	rw	rw												

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

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

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

- 0000: PA[x] pin
- 0001: PB[x] pin
- 0010: PC[x] pin
- 0011: PD[x] pin
- 0100: PE[x] pin
- 0101: PF[x] pin
- 0110: PG[x] pin
- 0111: PH[x] pin
- 1000: PI[x] pin
- 1001: PJ[x] pin
- 1010: PK[x] pin

7.2.5 SYSCFG external interrupt configuration register 3 (SYSCFG_EXTICR3)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI11[3:0]				EXTI10[3:0]				EXTI9[3:0]				EXTI8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

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

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

- 0000: PA[x] pin
- 0001: PB[x] pin
- 0010: PC[x] pin
- 0011: PD[x] pin
- 0100: PE[x] pin
- 0101: PF[x] pin
- 0110: PG[x] pin
- 0111: PH[x] pin
- 1000: PI[x] pin
- 1001: PJ[x] pin
- 1010: PK[x] pin

Note: PK[11:8] are not used

7.2.6 SYSCFG external interrupt configuration register 4 (SYSCFG_EXTICR4)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI15[3:0]				EXTI14[3:0]				EXTI13[3:0]				EXTI12[3:0]			
rw	rw	rw	rw												

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

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

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

- 0000: PA[x] pin
- 0001: PB[x] pin
- 0010: PC[x] pin
- 0011: PD[x] pin
- 0100: PE[x] pin
- 0101: PF[x] pin
- 0110: PG[x] pin
- 0111: PH[x] pin
- 1001: PJ[x] pin
- 1010: PK[x] pin

Note: PK[15:12] are not used

7.2.7 Compensation cell control register (SYSCFG_CMPCR)

Address offset: 0x20

Reset value: 0x0000 0000

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

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

Bit 8 **READY**: Compensation cell ready flag

0: I/O compensation cell not ready

1: I/O compensation cell ready

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

Bit 0 **CMP_PD**: Compensation cell power-down

0: I/O compensation cell power-down mode

1: I/O compensation cell enabled

7.2.8 SYSCFG register maps

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

Table 25. SYSCFG 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	SYSCFG_MEMRMP	Res.	0 MEM_BOOT	0																													
	Reset value																																
0x04	SYSCFG_PMC	Res.	0 CMP_PD	0																													
	Reset value																																
0x08	SYSCFG_EXTICR1	Res.	EXTI0[3:0]	0																													
	Reset value																																
0x0C	SYSCFG_EXTICR2	Res.	EXTI1[3:0]	0																													
	Reset value																																
0x10	SYSCFG_EXTICR3	Res.	EXTI2[3:0]	0																													
	Reset value																																
0x14	SYSCFG_EXTICR4	Res.	EXTI3[3:0]	0																													
	Reset value																																
0x20	SYSCFG_CMPCR	Res.	READY 0	0																													
	Reset value																																

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

8 Direct memory access controller (DMA)

8.1 DMA introduction

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

The DMA controller combines a powerful dual AHB master bus architecture with independent FIFO to optimize the bandwidth of the system, based on a complex bus matrix architecture.

The two DMA controllers (DMA1 and DMA2) have 16 streams in total (8 for each controller), each dedicated to managing memory access requests from one or more peripherals.

Each stream can have up to 8 channels (requests) in total.

Each DMA controller has an arbiter for handling the priority between DMA requests.

8.2 DMA main features

The main DMA features are:

- Dual AHB master bus architecture, one dedicated to memory accesses and one dedicated to peripheral accesses
- AHB slave programming interface supporting only 32-bit accesses
- 8 streams for each DMA controller, up to 16 channels (requests) per stream
- Four-word depth 32 first-in, first-out memory buffers (FIFOs) per stream, that can be used in FIFO mode or direct mode:
 - FIFO mode: with threshold level software selectable between 1/4, 1/2 or 3/4 of the FIFO size
 - Direct mode: each DMA request immediately initiates a transfer from/to the memory. When it is configured in direct mode (FIFO disabled), to transfer data in memory-to-peripheral mode, the DMA preloads only one data from the memory to the internal FIFO to ensure an immediate data transfer as soon as a DMA request is triggered by a peripheral.
- Each stream can be configured to be:
 - a regular channel that supports peripheral-to-memory, memory-to-peripheral and memory-to-memory transfers
 - a double buffer channel that also supports double buffering on the memory side
- Priorities between DMA stream requests are software-programmable (4 levels consisting of very high, high, medium, low) or hardware in case of equality (for example, request 0 has priority over request 1)
- Each stream also supports software trigger for memory-to-memory transfers (only available for the DMA2 controller)
- Each stream request can be selected among up to 8 possible channel requests. This selection is software-configurable and allows several peripherals to initiate DMA requests
- The number of data items to be transferred can be managed either by the DMA controller or by the peripheral:

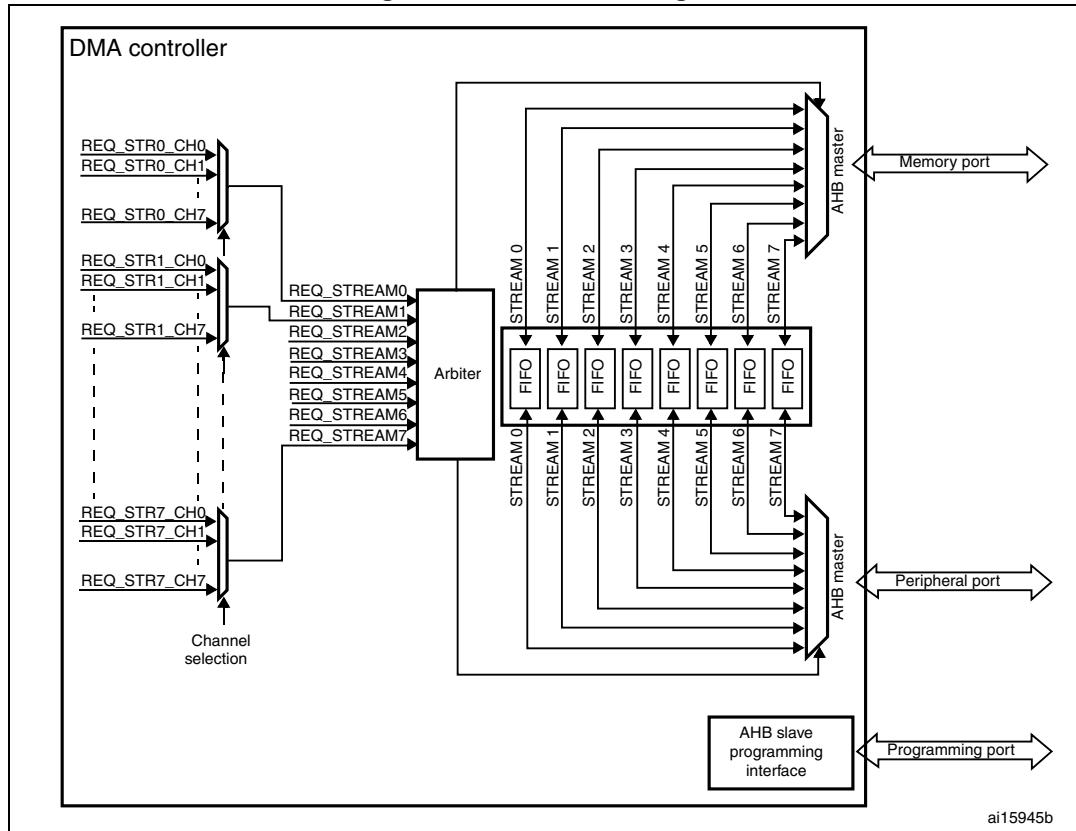
- DMA flow controller: the number of data items to be transferred is software-programmable from 1 to 65535
- Peripheral flow controller: the number of data items to be transferred is unknown and controlled by the source or the destination peripheral that signals the end of the transfer by hardware
- Independent source and destination transfer width (byte, half-word, word): when the data widths of the source and destination are not equal, the DMA automatically packs/unpacks the necessary transfers to optimize the bandwidth. This feature is only available in FIFO mode
- Incrementing or non-incrementing addressing for source and destination
- Supports incremental burst transfers of 4, 8 or 16 beats. The size of the burst is software-configurable, usually equal to half the FIFO size of the peripheral
- Each stream supports circular buffer management
- 5 event flags (DMA half transfer, DMA transfer complete, DMA transfer error, DMA FIFO error, direct mode error) logically ORed together in a single interrupt request for each stream

8.3 DMA functional description

8.3.1 DMA block diagram

Figure 24 shows the block diagram of a DMA.

Figure 24. DMA block diagram



8.3.2 DMA overview

The DMA controller performs direct memory transfer: as an AHB master, it can take the control of the AHB bus matrix to initiate AHB transactions.

It carries out the following transactions:

- peripheral-to-memory
- memory-to-peripheral
- memory-to-memory

The DMA controller provides two AHB master ports: the AHB memory port, intended to be connected to memories and the AHB peripheral port, intended to be connected to peripherals. However, to allow memory-to-memory transfers, the AHB peripheral port must also have access to the memories.

The AHB slave port is used to program the DMA controller (it supports only 32-bit accesses).

Note: The DMA1 controller AHB peripheral port is not connected to the bus matrix like in the case of the DMA2 controller, thus only DMA2 streams are able to perform memory-to-memory transfers.

See [Figure 1](#) for the implementation of the system of two DMA controllers.

8.3.3 DMA transactions

A DMA transaction consists of a sequence of a given number of data transfers. The number of data items to be transferred and their width (8-bit, 16-bit or 32-bit) are software-programmable.

Each DMA transfer consists of three operations:

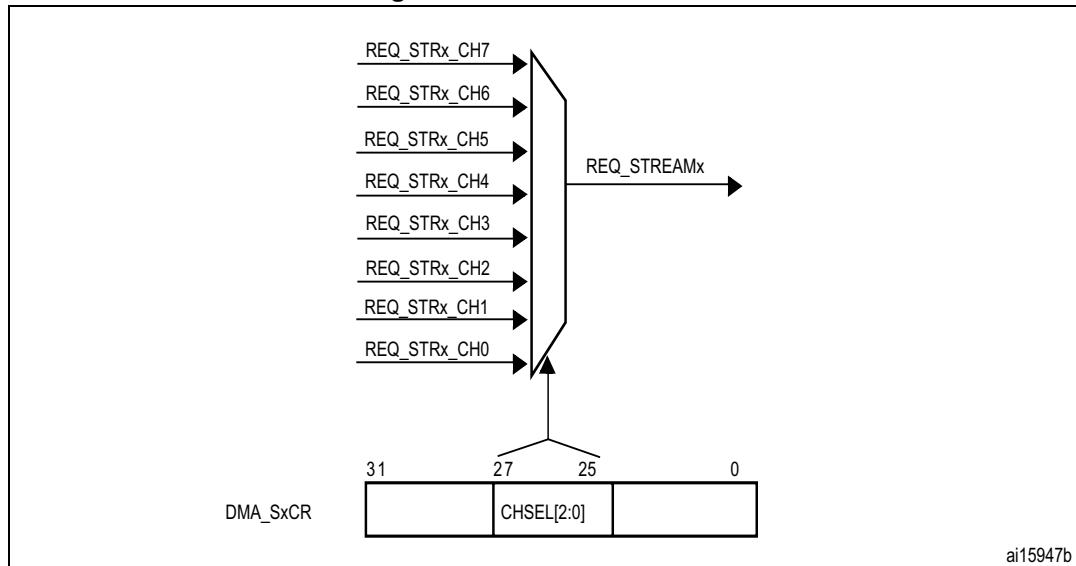
- a loading from the peripheral data register or a location in memory, addressed through the DMA_SxPAR or DMA_SxM0AR register
- a storage of the data loaded to the peripheral data register or a location in memory addressed through the DMA_SxPAR or DMA_SxM0AR register
- a post-decrement of the DMA_SxNDTR register, containing the number of transactions that still have to be performed

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 signal is sent to the peripheral by the DMA controller. The peripheral releases its request as soon as it gets the Acknowledge signal from the DMA controller. Once the request has been deasserted by the peripheral, the DMA controller releases the Acknowledge signal. If there are more requests, the peripheral can initiate the next transaction.

8.3.4 Channel selection

Each stream is associated with a DMA request that can be selected out of 8 possible channel requests. The selection is controlled by the CHSEL[2:0] bits in the DMA_SxCR register.

Figure 25. Channel selection



The 8 requests from the peripherals (such as TIM, ADC, SPI, I2C) are independently connected to each channel and their connection depends on the product implementation.

Table 26 and *Table 27* give examples of DMA request mappings.

Table 26. DMA1 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	SPI3_RX	-	SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX	-	SPI3_TX
Channel 1	I2C1_RX	I2C3_RX	TIM7_UP	-	TIM7_UP	I2C1_RX	I2C1_TX	I2C1_TX
Channel 2	TIM4_CH1	-	-	TIM4_CH2	-	-	TIM4_UP	TIM4_CH3
Channel 3	-	TIM2_UP TIM2_CH3	I2C3_RX	-	I2C3_TX	TIM2_CH1	TIM2_CH2 TIM2_CH4	TIM2_UP TIM2_CH4
Channel 4	UART5_RX	USART3_RX	UART4_RX	USART3_TX	UART4_TX	USART2_RX	USART2_TX	UART5_TX
Channel 5	UART8_TX	UART7_TX	TIM3_CH4 TIM3_UP	UART7_RX	TIM3_CH1 TIM3_TRIG	TIM3_CH2	UART8_RX	TIM3_CH3
Channel 6	TIM5_CH3 TIM5_UP	TIM5_CH4 TIM5_TRIG	TIM5_CH1	TIM5_CH4 TIM5_TRIG	TIM5_CH2	-	TIM5_UP	-
Channel 7	-	TIM6_UP	I2C2_RX	I2C2_RX	USART3_TX	DAC1	DAC2	I2C2_TX

Table 27. DMA2 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	ADC1	SAI1_A	TIM8_CH1 TIM8_CH2 TIM8_CH3	SAI1_A	ADC1	SAI1_B	TIM1_CH1 TIM1_CH2 TIM1_CH3	SAI2_B
Channel 1	-	-	ADC2	ADC2	SAI1_B	-	-	-
Channel 2	ADC3	ADC3	-	SPI5_RX	SPI5_TX	AES_OUT	AES_IN	-
Channel 3	SPI1_RX	-	SPI1_RX	SPI1_TX	SAI2_A	SPI1_TX	SAI2_B	QUADSPI
Channel 4	SPI4_RX	SPI4_TX	USART1_RX	SDMMC1	-	USART1_RX	SDMMC1	USART1_TX
Channel 5	-	USART6_RX	USART6_RX	SPI4_RX	SPI4_TX	-	USART6_TX	USART6_TX
Channel 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	-
Channel 7	-	TIM8_UP	TIM8_CH1	TIM8_CH2	TIM8_CH3	SPI5_RX	SPI5_TX	TIM8_CH4 TIM8_TRIG TIM8_COM
Channel 11	SDMMC2	-	-	-	-	SDMMC2	-	-

8.3.5 Arbiter

An arbiter manages the 8 DMA stream requests based on their priority for each of the two AHB master ports (memory and peripheral ports) and launches the peripheral/memory access sequences.

Priorities are managed in two stages:

- Software: each stream priority can be configured in the DMA_SxCR register. There are four levels:
 - Very high priority
 - High priority
 - Medium priority
 - Low priority
- Hardware: If two requests have the same software priority level, the stream with the lower number takes priority over the stream with the higher number. For example, stream 2 takes priority over stream 4.

8.3.6 DMA streams

Each of the 8 DMA controller streams provides a unidirectional transfer link between a source and a destination.

Each stream can be configured to perform:

- Regular type transactions: memory-to-peripherals, peripherals-to-memory or memory-to-memory transfers
- Double-buffer type transactions: double buffer transfers using two memory pointers for the memory (while the DMA is reading/writing from/to a buffer, the application can write/read to/from the other buffer).

The amount of data to be transferred (up to 65535) is programmable and related to the source width of the peripheral that requests the DMA transfer connected to the peripheral AHB port. The register that contains the amount of data items to be transferred is decremented after each transaction.

8.3.7 Source, destination and transfer modes

Both source and destination transfers can address peripherals and memories in the entire 4 Gbytes area, at addresses comprised between 0x0000 0000 and 0xFFFF FFFF.

The direction is configured using the DIR[1:0] bits in the DMA_SxCR register and offers three possibilities: memory-to-peripheral, peripheral-to-memory or memory-to-memory transfers. *Table 28* describes the corresponding source and destination addresses.

Table 28. Source and destination address

Bits DIR[1:0] of the DMA_SxCR register	Direction	Source address	Destination address
00	Peripheral-to-memory	DMA_SxPAR	DMA_SxM0AR
01	Memory-to-peripheral	DMA_SxM0AR	DMA_SxPAR
10	Memory-to-memory	DMA_SxPAR	DMA_SxM0AR
11	Reserved	-	-

When the data width (programmed in the PSIZE or MSIZE bits in the DMA_SxCR register) is a half-word or a word, respectively, the peripheral or memory address written into the DMA_SxPAR or DMA_SxM0AR/M1AR registers has to be aligned on a word or half-word address boundary, respectively.

Peripheral-to-memory mode

Figure 26 describes this mode.

When this mode is enabled (by setting the bit EN in the DMA_SxCR register), each time a peripheral request occurs, the stream initiates a transfer from the source to fill the FIFO.

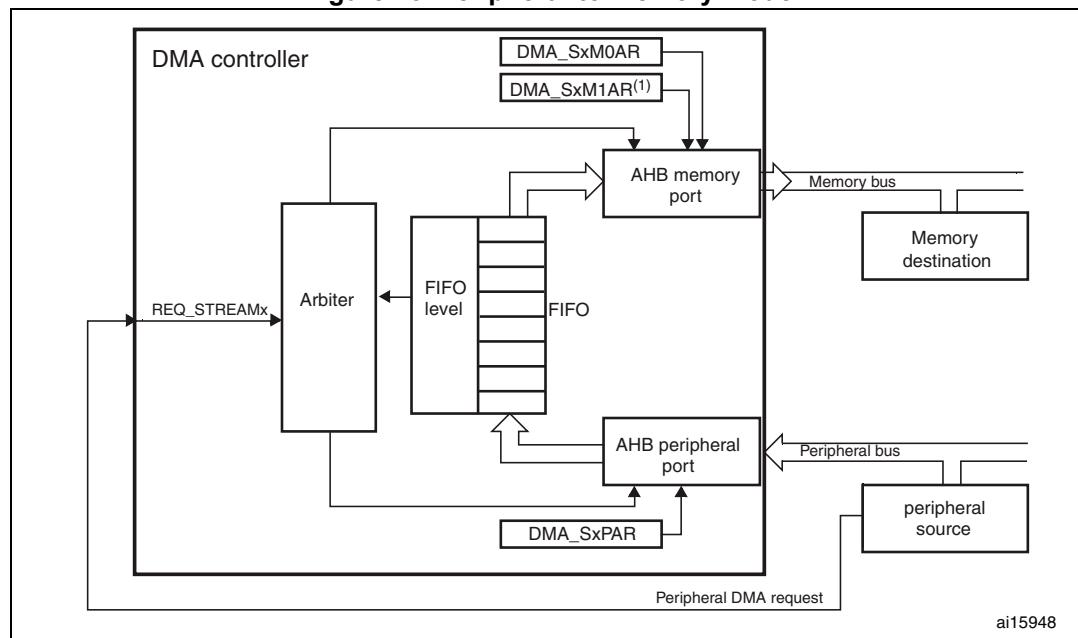
When the threshold level of the FIFO is reached, the contents of the FIFO are drained and stored into the destination.

The transfer stops once the DMA_SxNDTR register reaches zero, when the peripheral requests the end of transfers (in case of a peripheral flow controller) or when the EN bit in the DMA_SxCR register is cleared by software.

In direct mode (when the DMDIS value in the DMA_SxFCR register is ‘0’), the threshold level of the FIFO is not used: after each single data transfer from the peripheral to the FIFO, the corresponding data are immediately drained and stored into the destination.

The stream has access to the AHB source or destination port only if the arbitration of the corresponding stream is won. This arbitration is performed using the priority defined for each stream using the PL[1:0] bits in the DMA_SxCR register.

Figure 26. Peripheral-to-memory mode



1. For double-buffer mode.

Memory-to-peripheral mode

Figure 27 describes this mode.

When this mode is enabled (by setting the EN bit in the DMA_SxCR register), the stream immediately initiates transfers from the source to entirely fill the FIFO.

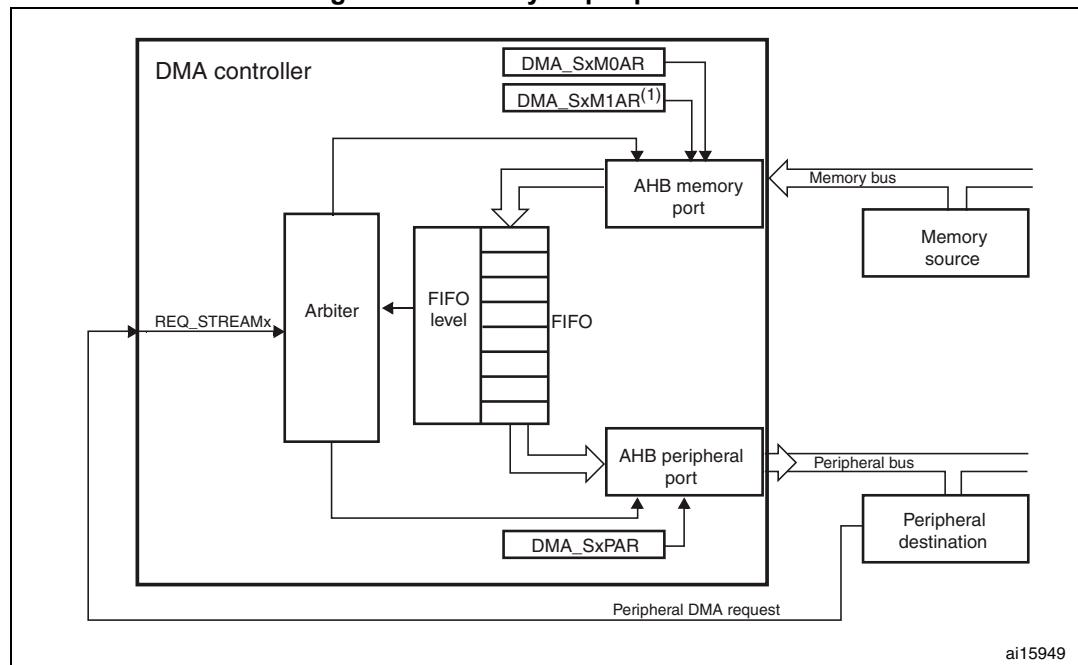
Each time a peripheral request occurs, the contents of the FIFO are drained and stored into the destination. When the level of the FIFO is lower than or equal to the predefined threshold level, the FIFO is fully reloaded with data from the memory.

The transfer stops once the DMA_SxNDTR register reaches zero, when the peripheral requests the end of transfers (in case of a peripheral flow controller) or when the EN bit in the DMA_SxCR register is cleared by software.

In direct mode (when the DMDIS value in the DMA_SxFCR register is '0'), the threshold level of the FIFO is not used. Once the stream is enabled, the DMA preloads the first data to transfer into an internal FIFO. As soon as the peripheral requests a data transfer, the DMA transfers the preloaded value into the configured destination. It then reloads again the empty internal FIFO with the next data to be transferred. The preloaded data size corresponds to the value of the PSIZE bitfield in the DMA_SxCR register.

The stream has access to the AHB source or destination port only if the arbitration of the corresponding stream is won. This arbitration is performed using the priority defined for each stream using the PL[1:0] bits in the DMA_SxCR register.

Figure 27. Memory-to-peripheral mode



1. For double-buffer mode.

Memory-to-memory mode

The DMA channels can also work without being triggered by a request from a peripheral. This is the memory-to-memory mode, described in [Figure 28](#).

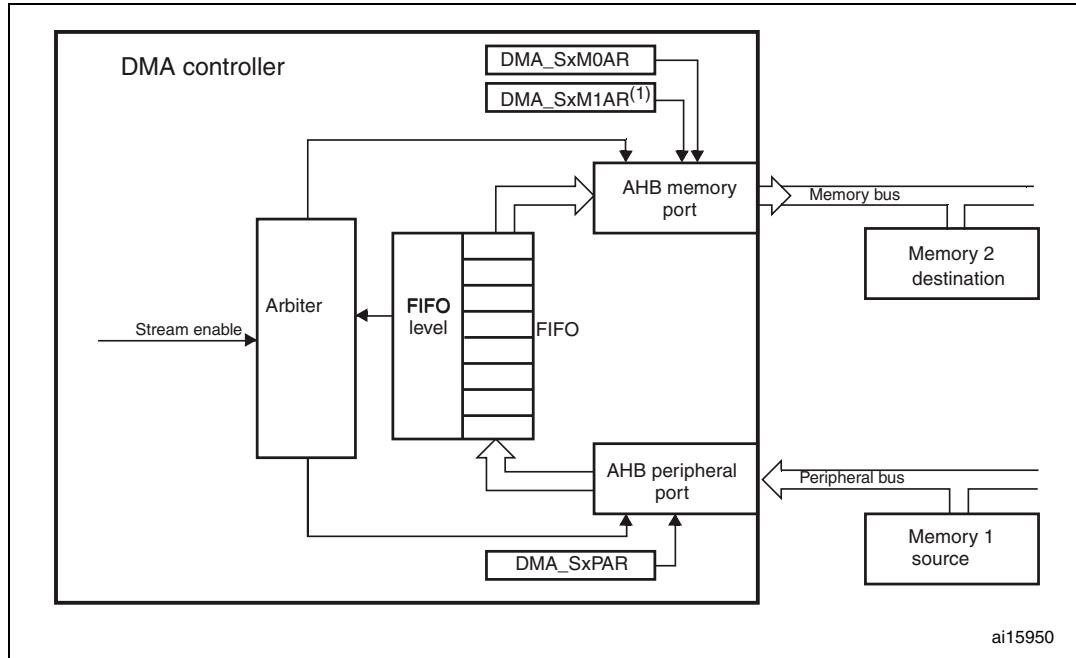
When the stream is enabled by setting the Enable bit (EN) in the DMA_SxCR register, the stream immediately starts to fill the FIFO up to the threshold level. When the threshold level is reached, the FIFO contents are drained and stored into the destination.

The transfer stops once the DMA_SxNDTR register reaches zero or when the EN bit in the DMA_SxCR register is cleared by software.

The stream has access to the AHB source or destination port only if the arbitration of the corresponding stream is won. This arbitration is performed using the priority defined for each stream using the PL[1:0] bits in the DMA_SxCR register.

Note: When memory-to-memory mode is used, the circular and direct modes are not allowed.
Only the DMA2 controller is able to perform memory-to-memory transfers.

Figure 28. Memory-to-memory mode



1. For double-buffer mode.

8.3.8 Pointer incrementation

Peripheral and memory pointers can optionally be automatically post-incremented or kept constant after each transfer depending on the PINC and MINC bits in the DMA_SxCR register.

Disabling the increment mode is useful when the peripheral source or destination data is accessed through a single register.

If the increment mode is enabled, the address of the next transfer is the address of the previous one incremented by 1 (for bytes), 2 (for half-words) or 4 (for words) depending on the data width programmed in the PSIZE or MSIZE bits in the DMA_SxCR register.

In order to optimize the packing operation, it is possible to fix the increment offset size for the peripheral address whatever the size of the data transferred on the AHB peripheral port. The PINCOS bit in the DMA_SxCR register is used to align the increment offset size with the data size on the peripheral AHB port, or on a 32-bit address (the address is then incremented by 4). The PINCOS bit has an impact on the AHB peripheral port only.

If the PINCOS bit is set, the address of the following transfer is the address of the previous one incremented by 4 (automatically aligned on a 32-bit address), whatever the PSIZE value. The AHB memory port, however, is not impacted by this operation.

8.3.9 Circular mode

The 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_SxCR register.

When the circular mode is activated, the number of data items to be transferred is automatically reloaded with the initial value programmed during the stream configuration phase, and the DMA requests continue to be served.

Note: *In the circular mode, it is mandatory to respect the following rule in case of a burst mode configured for memory:*

$$\text{DMA_SxNDTR} = \text{Multiple of } ((\text{Mburst beat}) \times (\text{Msize})/(\text{Psize})), \text{ where:}$$

- $(\text{Mburst beat}) = 4, 8 \text{ or } 16$ (depending on the MBURST bits in the DMA_SxCR register)
- $((\text{Msize})/(\text{Psize})) = 1, 2, 4, 1/2 \text{ or } 1/4$ (Msize and Psize represent the MSIZE and PSIZE bits in the DMA_SxCR register. They are byte dependent)
- $\text{DMA_SxNDTR} = \text{Number of data items to transfer on the AHB peripheral port}$

For example: Mburst beat = 8 (INCR8), MSIZE = '00' (byte) and PSIZE = '01' (half-word), in this case: DMA_SxNDTR must be a multiple of $(8 \times 1/2 = 4)$.

If this formula is not respected, the DMA behavior and data integrity are not guaranteed.

NDTR must also be a multiple of the Peripheral burst size multiplied by the peripheral data size, otherwise this could result in a bad DMA behavior.

8.3.10 Double-buffer mode

This mode is available for all the DMA1 and DMA2 streams.

The double-buffer mode is enabled by setting the DBM bit in the DMA_SxCR register.

A double-buffer stream works as a regular (single buffer) stream with the difference that it has two memory pointers. When the double-buffer mode is enabled, the circular mode is automatically enabled (CIRC bit in DMA_SxCR is not relevant) and at each end of transaction, the memory pointers are swapped.

In this mode, the DMA controller swaps from one memory target to another at each end of transaction. This allows the software to process one memory area while the second memory area is being filled/used by the DMA transfer. The double-buffer stream can work in both directions (the memory can be either the source or the destination) as described in [Table 29: Source and destination address registers in double-buffer mode \(DBM = 1\)](#).

Note: *In double-buffer mode, it is possible to update the base address for the AHB memory port on-the-fly (DMA_SxM0AR or DMA_SxM1AR) when the stream is enabled, by respecting the following conditions:*

- When the CT bit is '0' in the DMA_SxCR register, the DMA_SxM1AR register can be written. Attempting to write to this register while CT = '1' sets an error flag (TEIF) and the stream is automatically disabled.
- When the CT bit is '1' in the DMA_SxCR register, the DMA_SxM0AR register can be written. Attempting to write to this register while CT = '0', sets an error flag (TEIF) and the stream is automatically disabled.

To avoid any error condition, it is advised to change the base address as soon as the TCIF flag is asserted because, at this point, the targeted memory must have changed from

memory 0 to 1 (or from 1 to 0) depending on the value of CT in the DMA_SxCR register in accordance with one of the two above conditions.

For all the other modes (except the double-buffer mode), the memory address registers are write-protected as soon as the stream is enabled.

Table 29. Source and destination address registers in double-buffer mode (DBM = 1)

Bits DIR[1:0] of the DMA_SxCR register	Direction	Source address	Destination address
00	Peripheral-to-memory	DMA_SxPAR	DMA_SxM0AR / DMA_SxM1AR
01	Memory-to-peripheral	DMA_SxM0AR / DMA_SxM1AR	DMA_SxPAR
10	Not allowed ⁽¹⁾		
11	Reserved	-	-

- When the double-buffer mode is enabled, the circular mode is automatically enabled. Since the memory-to-memory mode is not compatible with the circular mode, when the double-buffer mode is enabled, it is not allowed to configure the memory-to-memory mode.

8.3.11 Programmable data width, packing/unpacking, endianness

The number of data items to be transferred has to be programmed into DMA_SxNDTR (number of data items to transfer bit, NDT) before enabling the stream (except when the flow controller is the peripheral, PFCTRL bit in DMA_SxCR is set).

When using the internal FIFO, the data widths of the source and destination data are programmable through the PSIZE and MSIZE bits in the DMA_SxCR register (can be 8-, 16- or 32-bit).

When PSIZE and MSIZE are not equal:

- The data width of the number of data items to transfer, configured in the DMA_SxNDTR register is equal to the width of the peripheral bus (configured by the PSIZE bits in the DMA_SxCR register). For instance, in case of peripheral-to-memory, memory-to-peripheral or memory-to-memory transfers and if the PSIZE[1:0] bits are configured for half-word, the number of bytes to be transferred is equal to $2 \times \text{NDT}$.
- The DMA controller only copes with little-endian addressing for both source and destination. This is described in [Table 30: Packing/unpacking and endian behavior \(bit PINC = MINC = 1\)](#).

This packing/unpacking procedure may present a risk of data corruption when the operation is interrupted before the data are completely packed/unpacked. So, to ensure data coherence, the stream may be configured to generate burst transfers: in this case, each group of transfers belonging to a burst are indivisible (refer to [Section 8.3.12: Single and burst transfers](#)).

In direct mode (DMDIS = 0 in the DMA_SxFCSR register), the packing/unpacking of data is not possible. In this case, it is not allowed to have different source and destination transfer data widths: both are equal and defined by the PSIZE bits in the DMA_SxCR register. MSIZE bits are not relevant.

Table 30. Packing/unpacking and endian behavior (bit PINC = MINC = 1)

AHB memory port width	AHB peripheral port width	Number of data items to transfer (NDT)	Memory transfer number	Memory port address / byte lane	Peripheral transfer number	Peripheral port address / byte lane	
						PINCOS = 1	PINCOS = 0
8	8	4	1	0x0 / B0[7:0]	1	0x0 / B0[7:0]	0x0 / B0[7:0]
			2	0x1 / B1[7:0]	2	0x4 / B1[7:0]	0x1 / B1[7:0]
			3	0x2 / B2[7:0]	3	0x8 / B2[7:0]	0x2 / B2[7:0]
			4	0x3 / B3[7:0]	4	0xC / B3[7:0]	0x3 / B3[7:0]
8	16	2	1	0x0 / B0[7:0]	1	0x0 / B1 B0[15:0]	0x0 / B1 B0[15:0]
			2	0x1 / B1[7:0]	2	0x4 / B3 B2[15:0]	0x2 / B3 B2[15:0]
			3	0x2 / B2[7:0]			
			4	0x3 / B3[7:0]			
8	32	1	1	0x0 / B0[7:0]	1	0x0 / B3 B2 B1 B0[31:0]	0x0 / B3 B2 B1 B0[31:0]
			2	0x1 / B1[7:0]			
			3	0x2 / B2[7:0]			
			4	0x3 / B3[7:0]			
16	8	4	1	0x0 / B1 B0[15:0]	1	0x0 / B0[7:0]	0x0 / B0[7:0]
			2	0x2 / B3 B2[15:0]	2	0x4 / B1[7:0]	0x1 / B1[7:0]
			3		3	0x8 / B2[7:0]	0x2 / B2[7:0]
			4		4	0xC / B3[7:0]	0x3 / B3[7:0]
16	16	2	1	0x0 / B1 B0[15:0]	1	0x0 / B1 B0[15:0]	0x0 / B1 B0[15:0]
			2	0x2 / B1 B0[15:0]	2	0x4 / B3 B2[15:0]	0x2 / B3 B2[15:0]
			3				
			4				
16	32	1	1	0x0 / B1 B0[15:0]	1	0x0 / B3 B2 B1 B0[31:0]	0x0 / B3 B2 B1 B0[31:0]
			2	0x2 / B3 B2[15:0]			
			3				
			4				
32	8	4	1	0x0 / B3 B2 B1 B0[31:0]	1	0x0 / B0[7:0]	0x0 / B0[7:0]
			2		2	0x4 / B1[7:0]	0x1 / B1[7:0]
			3		3	0x8 / B2[7:0]	0x2 / B2[7:0]
			4		4	0xC / B3[7:0]	0x3 / B3[7:0]
32	16	2	1	0x0 / B3 B2 B1 B0[31:0]	1	0x0 / B1 B0[15:0]	0x0 / B1 B0[15:0]
			2		2	0x4 / B3 B2[15:0]	0x2 / B3 B2[15:0]
32	32	1	1	0x0 / B3 B2 B1 B0[31:0]	1	0x0 / B3 B2 B1 B0[31:0]	0x0 / B3 B2 B1 B0[31:0]

Note: Peripheral port may be the source or the destination (it could also be the memory source in the case of memory-to-memory transfer).

PSIZE, MSIZE and NDT[15:0] have to be configured so as to ensure that the last transfer will not be incomplete. This can occur when the data width of the peripheral port (PSIZE bits) is lower than the data width of the memory port (MSIZE bits). This constraint is summarized in [Table 31](#).

Table 31. Restriction on NDT versus PSIZE and MSIZE

PSIZE[1:0] of DMA_SxCR	MSIZE[1:0] of DMA_SxCR	NDT[15:0] of DMA_SxNDTR
00 (8-bit)	01 (16-bit)	must be a multiple of 2
00 (8-bit)	10 (32-bit)	must be a multiple of 4
01 (16-bit)	10 (32-bit)	must be a multiple of 2

8.3.12 Single and burst transfers

The DMA controller can generate single transfers or incremental burst transfers of 4, 8 or 16 beats.

The size of the burst is configured by software independently for the two AHB ports by using the MBURST[1:0] and PBURST[1:0] bits in the DMA_SxCR register.

The burst size indicates the number of beats in the burst, not the number of bytes transferred.

To ensure data coherence, each group of transfers that form a burst are indivisible: AHB transfers are locked and the arbiter of the AHB bus matrix does not degrant the DMA master during the sequence of the burst transfer.

Depending on the single or burst configuration, each DMA request initiates a different number of transfers on the AHB peripheral port:

- When the AHB peripheral port is configured for single transfers, each DMA request generates a data transfer of a byte, half-word or word depending on the PSIZE[1:0] bits in the DMA_SxCR register
- When the AHB peripheral port is configured for burst transfers, each DMA request generates 4,8 or 16 beats of byte, half word or word transfers depending on the PBURST[1:0] and PSIZE[1:0] bits in the DMA_SxCR register.

The same as above has to be considered for the AHB memory port considering the MBURST and MSIZE bits.

In direct mode, the stream can only generate single transfers and the MBURST[1:0] and PBURST[1:0] bits are forced by hardware.

The address pointers (DMA_SxPAR or DMA_SxM0AR registers) must be chosen so as to ensure that all transfers within a burst block are aligned on the address boundary equal to the size of the transfer.

The burst configuration has to be selected in order to respect the AHB protocol, where bursts **must not** cross the 1 Kbyte address boundary because the minimum address space that can be allocated to a single slave is 1 Kbyte. This means that the 1 Kbyte address boundary **must not** be crossed by a burst block transfer, otherwise an AHB error is generated, that is not reported by the DMA registers.

8.3.13 FIFO

FIFO structure

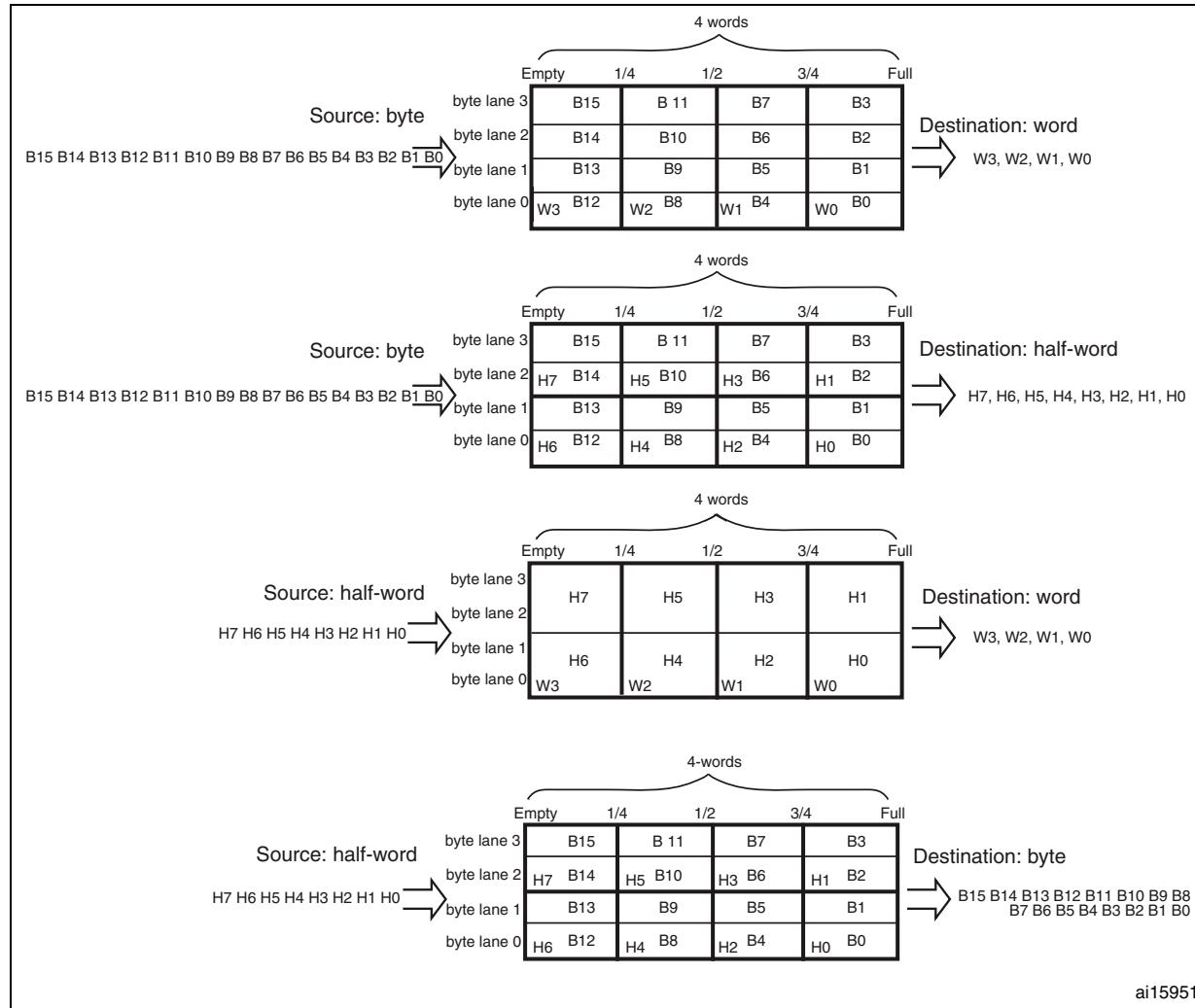
The FIFO is used to temporarily store data coming from the source before transmitting them to the destination.

Each stream has an independent 4-word FIFO and the threshold level is software-configurable between 1/4, 1/2, 3/4 or full.

To enable the use of the FIFO threshold level, the direct mode must be disabled by setting the DMDIS bit in the DMA_SxFCR register.

The structure of the FIFO differs depending on the source and destination data widths, and is described in [Figure 29: FIFO structure](#).

Figure 29. FIFO structure



ai15951

FIFO threshold and burst configuration

Caution is required when choosing the FIFO threshold (bits FTH[1:0] of the DMA_SxFCR register) and the size of the memory burst (MBURST[1:0] of the DMA_SxCR register): The content pointed by the FIFO threshold must exactly match an integer number of memory burst transfers. If this is not in the case, a FIFO error (flag FEI_F of the DMA_HISR or DMA_LISR register) is generated when the stream is enabled, then the stream is automatically disabled. The allowed and forbidden configurations are described in [Table 32](#). The forbidden configurations are highlighted in gray in the table.

Table 32. FIFO threshold configurations

MSIZE	FIFO level	MBURST = INCR4	MBURST = INCR8	MBURST = INCR16
Byte	1/4	1 burst of 4 beats	Forbidden	Forbidden
	1/2	2 bursts of 4 beats	1 burst of 8 beats	
	3/4	3 bursts of 4 beats	Forbidden	
	Full	4 bursts of 4 beats	2 bursts of 8 beats	1 burst of 16 beats

Table 32. FIFO threshold configurations (continued)

MSIZE	FIFO level	MBURST = INCR4	MBURST = INCR8	MBURST = INCR16
Half-word	1/4	Forbidden	Forbidden	Forbidden
	1/2	1 burst of 4 beats		
	3/4	Forbidden		
	Full	2 bursts of 4 beats	1 burst of 8 beats	
Word	1/4	Forbidden	Forbidden	Forbidden
	1/2			
	3/4			
	Full	1 burst of 4 beats		

In all cases, the burst size multiplied by the data size must not exceed the FIFO size (data size can be: 1 (byte), 2 (half-word) or 4 (word)).

Incomplete burst transfer at the end of a DMA transfer may happen if one of the following conditions occurs:

- For the AHB peripheral port configuration: the total number of data items (set in the DMA_SxNDTR register) is not a multiple of the burst size multiplied by the data size.
- For the AHB memory port configuration: the number of remaining data items in the FIFO to be transferred to the memory is not a multiple of the burst size multiplied by the data size.

In such cases, the remaining data to be transferred is managed in single mode by the DMA, even if a burst transaction is requested during the DMA stream configuration.

Note:

When burst transfers are requested on the peripheral AHB port and the FIFO is used (DMDIS = 1 in the DMA_SxCR register), it is mandatory to respect the following rule to avoid permanent underrun or overrun conditions, depending on the DMA stream direction:

If $(PBURST \times PSIZE) = FIFO_SIZE$ (4 words), FIFO_Threshold = 3/4 is forbidden with PSIZE = 1, 2 or 4 and PBURST = 4, 8 or 16.

This rule ensures that enough FIFO space at a time is free to serve the request from the peripheral.

FIFO flush

The FIFO can be flushed when the stream is disabled by resetting the EN bit in the DMA_SxCR register and when the stream is configured to manage peripheral-to-memory or memory-to-memory transfers. If some data are still present in the FIFO when the stream is disabled, the DMA controller continues transferring the remaining data to the destination (even though stream is effectively disabled). When this flush is completed, the transfer complete status bit (TCIFx) in the DMA_LISR or DMA_HISR register is set.

The remaining data counter DMA_SxNDTR keeps the value in this case to indicate how many data items are currently available in the destination memory.

Note that during the FIFO flush operation, if the number of remaining data items in the FIFO to be transferred to memory (in bytes) is less than the memory data width (for example 2 bytes in FIFO while MSIZE is configured to word), data is sent with the data width set in the MSIZE bit in the DMA_SxCR register. This means that memory is written with an undesired

value. The software may read the DMA_SxNDTR register to determine the memory area that contains the good data (start address and last address).

If the number of remaining data items in the FIFO is lower than a burst size (if the MBURST bits in DMA_SxCR register are set to configure the stream to manage burst on the AHB memory port), single transactions are generated to complete the FIFO flush.

Direct mode

By default, the FIFO operates in direct mode (DMDIS bit in the DMA_SxFIFO is reset) and the FIFO threshold level is not used. This mode is useful when the system requires an immediate and single transfer to or from the memory after each DMA request.

When the DMA is configured in direct mode (FIFO disabled), to transfer data in memory-to-peripheral mode, the DMA preloads one data from the memory to the internal FIFO to ensure an immediate data transfer as soon as a DMA request is triggered by a peripheral.

To avoid saturating the FIFO, it is recommended to configure the corresponding stream with a high priority.

This mode is restricted to transfers where:

- the source and destination transfer widths are equal and both defined by the PSIZE[1:0] bits in DMA_SxCR (MSIZE[1:0] bits are not relevant)
- burst transfers are not possible (PBURST[1:0] and MBURST[1:0] bits in DMA_SxCR are don't care)

Direct mode must not be used when implementing memory-to-memory transfers.

8.3.14 DMA transfer completion

Different events can generate an end of transfer by setting the TCIFx bit in the DMA_LISR or DMA_HISR status register:

- In DMA flow controller mode:
 - The DMA_SxNDTR counter has reached zero in the memory-to-peripheral mode.
 - The stream is disabled before the end of transfer (by clearing the EN bit in the DMA_SxCR register) and (when transfers are peripheral-to-memory or memory-to-memory) all the remaining data have been flushed from the FIFO into the memory.
- In Peripheral flow controller mode:
 - The last external burst or single request has been generated from the peripheral and (when the DMA is operating in peripheral-to-memory mode) the remaining data have been transferred from the FIFO into the memory
 - The stream is disabled by software, and (when the DMA is operating in peripheral-to-memory mode) the remaining data have been transferred from the FIFO into the memory

Note: The transfer completion is dependent on the remaining data in FIFO to be transferred into memory only in the case of peripheral-to-memory mode. This condition is not applicable in memory-to-peripheral mode.

If the stream is configured in noncircular mode, after the end of the transfer (that is when the number of data to be transferred reaches zero), the DMA is stopped (EN bit in DMA_SxCR register is cleared by Hardware) and no DMA request is served unless the software reprograms the stream and re-enables it (by setting the EN bit in the DMA_SxCR register).

8.3.15 DMA transfer suspension

At any time, a DMA transfer can be suspended to be restarted later on or to be definitively disabled before the end of the DMA transfer.

There are two cases:

- The stream disables the transfer with no later-on restart from the point where it was stopped. There is no particular action to do, except to clear the EN bit in the DMA_SxCR register to disable the stream. The stream may take time to be disabled (ongoing transfer is completed first). The transfer complete interrupt flag (TCIF in the DMA_LISR or DMA_HISR register) is set in order to indicate the end of transfer. The value of the EN bit in DMA_SxCR is now '0' to confirm the stream interruption. The DMA_SxNDTR register contains the number of remaining data items at the moment when the stream was stopped so that the software can determine how many data items have been transferred before the stream was interrupted.
- The stream suspends the transfer before the number of remaining data items to be transferred in the DMA_SxNDTR register reaches 0. The aim is to restart the transfer later by re-enabling the stream. In order to restart from the point where the transfer was stopped, the software has to read the DMA_SxNDTR register after disabling the stream by writing the EN bit in DMA_SxCR register (and then checking that it is at '0') to know the number of data items already collected. Then:
 - The peripheral and/or memory addresses have to be updated in order to adjust the address pointers
 - The SxNDTR register has to be updated with the remaining number of data items to be transferred (the value read when the stream was disabled)
 - The stream may then be re-enabled to restart the transfer from the point it was stopped

Note: A transfer complete interrupt flag (TCIF in DMA_LISR or DMA_HISR) is set to indicate the end of transfer due to the stream interruption.

8.3.16 Flow controller

The entity that controls the number of data to be transferred is known as the flow controller. This flow controller is configured independently for each stream using the PFCTRL bit in the DMA_SxCR register.

The flow controller can be:

- The DMA controller: in this case, the number of data items to be transferred is programmed by software into the DMA_SxNDTR register before the DMA stream is enabled.
- The peripheral source or destination: this is the case when the number of data items to be transferred is unknown. The peripheral indicates by hardware to the DMA controller when the last data are being transferred. This feature is only supported for peripherals that are able to signal the end of the transfer, that is: SDMMC1.

When the peripheral flow controller is used for a given stream, the value written into the DMA_SxNDTR has no effect on the DMA transfer. Actually, whatever the value written, it will be forced by hardware to 0xFFFF as soon as the stream is enabled, to respect the following schemes:

- Anticipated stream interruption: EN bit in DMA_SxCR register is reset to 0 by the software to stop the stream before the last data hardware signal (single or burst) is sent by the peripheral. In such a case, the stream is switched off and the FIFO flush is

triggered in the case of a peripheral-to-memory DMA transfer. The TCIFx flag of the corresponding stream is set in the status register to indicate the DMA completion. To know the number of data items transferred during the DMA transfer, read the DMA_SxNDTR register and apply the following formula:

- Number_of_data_transferred = 0xFFFF – DMA_SxNDTR
- Normal stream interruption due to the reception of a last data hardware signal: the stream is automatically interrupted when the peripheral requests the last transfer (single or burst) and when this transfer is complete. the TCIFx flag of the corresponding stream is set in the status register to indicate the DMA transfer completion. To know the number of data items transferred, read the DMA_SxNDTR register and apply the same formula as above.
- The DMA_SxNDTR register reaches 0: the TCIFx flag of the corresponding stream is set in the status register to indicate the forced DMA transfer completion. The stream is automatically switched off even though the last data hardware signal (single or burst) has not been yet asserted. The already transferred data is not lost. This means that a maximum of 65535 data items can be managed by the DMA in a single transaction, even in peripheral flow control mode.

Note: When configured in memory-to-memory mode, the DMA is always the flow controller and the PFCTRL bit is forced to 0 by hardware.

The circular mode is forbidden in the peripheral flow controller mode.

8.3.17 Summary of the possible DMA configurations

[Table 33](#) summarizes the different possible DMA configurations. The forbidden configurations are highlighted in gray in the table.

Table 33. Possible DMA configurations

DMA transfer mode	Source	Destination	Flow controller	Circular mode	Transfer type	Direct mode	Double-buffer mode
Peripheral-to-memory	AHB peripheral port	AHB memory port	DMA	Possible	single	Possible	Possible
					burst	Forbidden	
	Peripheral		Peripheral	Forbidden	single	Possible	Forbidden
					burst	Forbidden	
Memory-to-peripheral	AHB memory port	AHB peripheral port	DMA	Possible	single	Possible	Possible
					burst	Forbidden	
	Peripheral		Peripheral	Forbidden	single	Possible	Forbidden
					burst	Forbidden	
Memory-to-memory	AHB peripheral port	AHB memory port	DMA only	Forbidden	single	Forbidden	Forbidden
					burst		

8.3.18 Stream configuration procedure

The following sequence must be followed to configure a DMA stream x (where x is the stream number):

1. If the stream is enabled, disable it by resetting the EN bit in the DMA_SxCR register, then read this bit in order to confirm that there is no ongoing stream operation. Writing this bit to 0 is not immediately effective since it is actually written to 0 once all the current transfers are finished. When the EN bit is read as 0, this means that the stream is ready to be configured. It is therefore necessary to wait for the EN bit to be cleared before starting any stream configuration. All the stream dedicated bits set in the status register (DMA_LISR and DMA_HISR) from the previous data block DMA transfer must be cleared before the stream can be re-enabled.
2. Set the peripheral port register address in the DMA_SxPAR register. The data is moved from/ to this address to/ from the peripheral port after the peripheral event.
3. Set the memory address in the DMA_SxMA0R register (and in the DMA_SxMA1R register in the case of a double-buffer mode). The data is written to or read from this memory after the peripheral event.
4. Configure the total number of data items to be transferred in the DMA_SxNDTR register. After each peripheral event or each beat of the burst, this value is decremented.
5. Select the DMA channel (request) using CHSEL[2:0] in the DMA_SxCR register.
6. If the peripheral is intended to be the flow controller and if it supports this feature, set the PFCTRL bit in the DMA_SxCR register.
7. Configure the stream priority using the PL[1:0] bits in the DMA_SxCR register.
8. Configure the FIFO usage (enable or disable, threshold in transmission and reception)
9. Configure the data transfer direction, peripheral and memory incremented/fixed mode, single or burst transactions, peripheral and memory data widths, circular mode, double-buffer mode and interrupts after half and/or full transfer, and/or errors in the DMA_SxCR register.
10. Activate the stream by setting the EN bit in the DMA_SxCR register.

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

Once half the data have been transferred on the AHB destination port, 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.

Warning: To switch off a peripheral connected to a DMA stream request, it is mandatory to, first, switch off the DMA stream to which the peripheral is connected, then to wait for EN bit = 0. Only then can the peripheral be safely disabled.

8.3.19 Error management

The DMA controller can detect the following errors:

- **Transfer error:** the transfer error interrupt flag (TEIFx) is set when:
 - a bus error occurs during a DMA read or a write access
 - a write access is requested by software on a memory address register in double-buffer mode whereas the stream is enabled and the current target memory is the one impacted by the write into the memory address register (refer to [Section 8.3.10: Double-buffer mode](#))
- **FIFO error:** the FIFO error interrupt flag (FEIFx) is set if:
 - a FIFO underrun condition is detected
 - a FIFO overrun condition is detected (no detection in memory-to-memory mode because requests and transfers are internally managed by the DMA)
 - the stream is enabled while the FIFO threshold level is not compatible with the size of the memory burst (refer to [Table 32: FIFO threshold configurations](#))
- **Direct mode error:** the direct mode error interrupt flag (DMEIFx) can only be set in the peripheral-to-memory mode while operating in direct mode and when the MINC bit in the DMA_SxCR register is cleared. This flag is set when a DMA request occurs while the previous data have not yet been fully transferred into the memory (because the memory bus was not granted). In this case, the flag indicates that 2 data items were be transferred successively to the same destination address, which could be an issue if the destination is not able to manage this situation

In direct mode, the FIFO error flag can also be set under the following conditions:

- In the peripheral-to-memory mode, the FIFO can be saturated (overrun) if the memory bus is not granted for several peripheral requests.
- In the memory-to-peripheral mode, an underrun condition may occur if the memory bus has not been granted before a peripheral request occurs.

If the TEIFx or the FEIFx flag is set due to incompatibility between burst size and FIFO threshold level, the faulty stream is automatically disabled through a hardware clear of its EN bit in the corresponding stream configuration register (DMA_SxCR).

If the DMEIFx or the FEIFx flag is set due to an overrun or underrun condition, the faulty stream is not automatically disabled and it is up to the software to disable or not the stream by resetting the EN bit in the DMA_SxCR register. This is because there is no data loss when this kind of errors occur.

When the stream's error interrupt flag (TEIF, FEIF, DMEIF) in the DMA_LISR or DMA_HISR register is set, an interrupt is generated if the corresponding interrupt enable bit (TEIE, FEIE, DMIE) in the DMA_SxCR or DMA_SxFCR register is set.

Note:

When a FIFO overrun or underrun condition occurs, the data is not lost because the peripheral request is not acknowledged by the stream until the overrun or underrun condition is cleared. If this acknowledge takes too much time, the peripheral itself may detect an overrun or underrun condition of its internal buffer and data might be lost.

8.4 DMA interrupts

For each DMA stream, an interrupt can be produced on the following events:

- Half-transfer reached
- Transfer complete
- Transfer error
- FIFO error (overrun, underrun or FIFO level error)
- Direct mode error

Separate interrupt enable control bits are available for flexibility as shown in [Table 34](#).

Table 34. DMA interrupt requests

Interrupt event	Event flag	Enable control bit
Half-transfer	HTIF	HTIE
Transfer complete	TCIF	TCIE
Transfer error	TEIF	TEIE
FIFO overrun/underrun	FEIF	FEIE
Direct mode error	DMEIF	DMEIE

Note: Before setting an enable control bit EN = 1, the corresponding event flag must be cleared, otherwise an interrupt is immediately generated.

8.5 DMA registers

The DMA registers have to be accessed by words (32 bits).

8.5.1 DMA low interrupt status register (DMA_LISR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	TCIF3	HTIF3	TEIF3	DMEIF3	Res.	FEIF3	TCIF2	HTIF2	TEIF2	DMEIF2	Res.	FEIF2
				r	r	r	r		r	r	r	r	r		r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	TCIF1	HTIF1	TEIF1	DMEIF1	Res.	FEIF1	TCIF0	HTIF0	TEIF0	DMEIF0	Res.	FEIF0
				r	r	r	r		r	r	r	r	r		r

Bits 31:28, 15:12 Reserved, must be kept at reset value.

Bits 27, 21, 11, 5 **TCIF[3:0]**: stream x transfer complete interrupt flag (x = 3..0)

This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the DMA_LIFCR register.

0: no transfer complete event on stream x
1: a transfer complete event occurred on stream x

Bits 26, 20, 10, 4 **HTIF[3:0]**: stream x half transfer interrupt flag (x = 3..0)

This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the DMA_LIFCR register.

0: no half transfer event on stream x
1: a half transfer event occurred on stream x

Bits 25, 19, 9, 3 **TEIF[3:0]**: stream x transfer error interrupt flag (x = 3..0)

This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the DMA_LIFCR register.

0: no transfer error on stream x
1: a transfer error occurred on stream x

Bits 24, 18, 8, 2 **DMEIF[3:0]**: stream x direct mode error interrupt flag (x = 3..0)

This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the DMA_LIFCR register.

0: No direct mode error on stream x
1: a direct mode error occurred on stream x

Bits 23, 17, 7, 1 Reserved, must be kept at reset value.

Bits 22, 16, 6, 0 **FEIF[3:0]**: stream x FIFO error interrupt flag (x = 3..0)

This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the DMA_LIFCR register.

0: no FIFO error event on stream x
1: a FIFO error event occurred on stream x

8.5.2 DMA high interrupt status register (DMA_HISR)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	TCIF7	HTIF7	TEIF7	DMEIF7	Res.	FEIF7	TCIF6	HTIF6	TEIF6	DMEIF6	Res.	FEIF6
				r	r	r	r		r	r	r	r	r		r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	TCIF5	HTIF5	TEIF5	DMEIF5	Res.	FEIF5	TCIF4	HTIF4	TEIF4	DMEIF4	Res.	FEIF4
				r	r	r	r		r	r	r	r	r		r

Bits 31:28, 15:12 Reserved, must be kept at reset value.

Bits 27, 21, 11, 5 **TCIF[7:4]**: stream x transfer complete interrupt flag (x = 7..4)

This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the DMA_HIFCR register.

0: no transfer complete event on stream x
1: a transfer complete event occurred on stream x

Bits 26, 20, 10, 4 **HTIF[7:4]**: stream x half transfer interrupt flag (x = 7..4)

This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the DMA_HIFCR register.

0: no half transfer event on stream x
1: a half transfer event occurred on stream x

Bits 25, 19, 9, 3 **TEIF[7:4]**: stream x transfer error interrupt flag (x = 7..4)

This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the DMA_HIFCR register.

0: no transfer error on stream x
1: a transfer error occurred on stream x

Bits 24, 18, 8, 2 **DMEIF[7:4]**: stream x direct mode error interrupt flag (x = 7..4)

This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the DMA_HIFCR register.

0: no direct mode error on stream x
1: a direct mode error occurred on stream x

Bits 23, 17, 7, 1 Reserved, must be kept at reset value.

Bits 22, 16, 6, 0 **FEIF[7:4]**: stream x FIFO error interrupt flag (x = 7..4)

This bit is set by hardware. It is cleared by software writing 1 to the corresponding bit in the DMA_HIFCR register.

0: no FIFO error event on stream x
1: a FIFO error event occurred on stream x

8.5.3 DMA low interrupt flag clear register (DMA_LIFCR)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	CTCIF3	CHTIF3	CTEIF3	CDMEIF3	Res.	CFEIF3	CTCIF2	CHTIF2	CTEIF2	CDMEIF2	Res.	CFEIF2
				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
Res.	Res.	Res.	Res.	CTCIF1	CHTIF1	CTEIF1	CDMEIF1	Res.	CFEIF1	CTCIF0	CHTIF0	CTEIF0	CDMEIF0	Res.	CFEIF0
				w	w	w	w		w	w	w	w	w		w

Bits 31:28, 15:12 Reserved, must be kept at reset value.

Bits 27, 21, 11, 5 **CTCIF[3:0]**: stream x clear transfer complete interrupt flag (x = 3..0)

Writing 1 to this bit clears the corresponding TCIFx flag in the DMA_LISR register.

Bits 26, 20, 10, 4 **CHTIF[3:0]**: stream x clear half transfer interrupt flag (x = 3..0)

Writing 1 to this bit clears the corresponding HTIFx flag in the DMA_LISR register

Bits 25, 19, 9, 3 **CTEIF[3:0]**: Stream x clear transfer error interrupt flag (x = 3..0)

Writing 1 to this bit clears the corresponding TEIFx flag in the DMA_LISR register.

Bits 24, 18, 8, 2 **CDMEIF[3:0]**: stream x clear direct mode error interrupt flag (x = 3..0)

Writing 1 to this bit clears the corresponding DMEIFx flag in the DMA_LISR register.

Bits 23, 17, 7, 1 Reserved, must be kept at reset value.

Bits 22, 16, 6, 0 **CFEIF[3:0]**: stream x clear FIFO error interrupt flag (x = 3..0)

Writing 1 to this bit clears the corresponding CFEIFx flag in the DMA_LISR register.

8.5.4 DMA high interrupt flag clear register (DMA_HIFCR)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	CTCIF7	CHTIF7	CTEIF7	CDMEIF7	Res.	CFEIF7	CTCIF6	CHTIF6	CTEIF6	CDMEIF6	Res.	CFEIF6
				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
Res.	Res.	Res.	Res.	CTCIF5	CHTIF5	CTEIF5	CDMEIF5	Res.	CFEIF5	CTCIF4	CHTIF4	CTEIF4	CDMEIF4	Res.	CFEIF4
				w	w	w	w		w	w	w	w	w		w

Bits 31:28, 15:12 Reserved, must be kept at reset value.

Bits 27, 21, 11, 5 **CTCIF[7:4]**: stream x clear transfer complete interrupt flag (x = 7..4)

Writing 1 to this bit clears the corresponding TCIFx flag in the DMA_HISR register.

Bits 26, 20, 10, 4 **CHTIF[7:4]**: stream x clear half transfer interrupt flag (x = 7..4)

Writing 1 to this bit clears the corresponding HTIFx flag in the DMA_HISR register.

Bits 25, 19, 9, 3 **CTEIF[7:4]**: stream x clear transfer error interrupt flag (x = 7..4)

Writing 1 to this bit clears the corresponding TEIFx flag in the DMA_HISR register.

Bits 24, 18, 8, 2 **CDMEIF[7:4]**: stream x clear direct mode error interrupt flag (x = 7..4)

Writing 1 to this bit clears the corresponding DMEIF_x flag in the DMA_HISR register.

Bits 23, 17, 7, 1 Reserved, must be kept at reset value.

Bits 22, 16, 6, 0 **CFEIF[7:4]**: stream x clear FIFO error interrupt flag (x = 7..4)

Writing 1 to this bit clears the corresponding CFEIF_x flag in the DMA_HISR register.

8.5.5 DMA stream x configuration register (DMA_SxCR)

This register is used to configure the concerned stream.

Address offset: 0x10 + 0x18 * x, (x = 0 to 7)

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	CHSEL[3:0]				MBURST[1:0]		PBURST[1:0]		Res.	CT	DBM	PL[1:0]	
			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
PINCO	MSIZE[1:0]	PSIZE[1:0]	MINC	PINC	CIRC	DIR[1:0]		PFCTRL	TCIE	HTIE	TEIE	DMEIE	EN		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 28:25 **CHSEL[3:0]**: channel selection

These bits are set and cleared by software.

0000: channel 0 selected

0001: channel 1 selected

0010: channel 2 selected

0011: channel 3 selected

0100: channel 4 selected

0101: channel 5 selected

0110: channel 6 selected

0111: channel 7 selected

1000: channel 8 selected

1001: channel 9 selected

1010: channel 10 selected

1011: channel 11 selected

1100: channel 12 selected

1101: channel 13 selected

1110: channel 14 selected

1111: channel 15 selected

These bits are protected and can be written only if EN is '0'.

Bits 24:23 **MBURST[1:0]**: memory burst transfer configuration

These bits are set and cleared by software.

00: single transfer

01: INCR4 (incremental burst of 4 beats)

10: INCR8 (incremental burst of 8 beats)

11: INCR16 (incremental burst of 16 beats)

These bits are protected and can be written only if EN is '0'.

In direct mode, these bits are forced to 0x0 by hardware as soon as bit EN= '1'.

Bits 22:21 **PBURST[1:0]**: peripheral burst transfer configuration

These bits are set and cleared by software.

00: single transfer

01: INCR4 (incremental burst of 4 beats)

10: INCR8 (incremental burst of 8 beats)

11: INCR16 (incremental burst of 16 beats)

These bits are protected and can be written only if EN is '0'.

In direct mode, these bits are forced to 0x0 by hardware.

Bit 20 Reserved, must be kept at reset value.

Bit 19 **CT**: current target (only in double-buffer mode)

This bit is set and cleared by hardware. It can also be written by software.

0: current target memory is Memory 0 (addressed by the DMA_SxM0AR pointer)

1: current target memory is Memory 1 (addressed by the DMA_SxM1AR pointer)

This bit can be written only if EN is '0' to indicate the target memory area of the first transfer.

Once the stream is enabled, this bit operates as a status flag indicating which memory area is the current target.

Bit 18 **DBM**: double-buffer mode

This bit is set and cleared by software.

0: no buffer switching at the end of transfer

1: memory target switched at the end of the DMA transfer

This bit is protected and can be written only if EN is '0'.

Bits 17:16 **PL[1:0]**: priority level

These bits are set and cleared by software.

00: low

01: medium

10: high

11: very high

These bits are protected and can be written only if EN is '0'.

Bit 15 **PINCOS**: peripheral increment offset size

This bit is set and cleared by software

0: The offset size for the peripheral address calculation is linked to the PSIZE

1: The offset size for the peripheral address calculation is fixed to 4 (32-bit alignment).

This bit has no meaning if bit PINC = '0'.

This bit is protected and can be written only if EN = '0'.

This bit is forced low by hardware when the stream is enabled (bit EN = '1') if the direct mode is selected or if PBURST are different from "00".

Bits 14:13 **MSIZE[1:0]**: memory data size

These bits are set and cleared by software.

00: byte (8-bit)

01: half-word (16-bit)

10: word (32-bit)

11: reserved

These bits are protected and can be written only if EN is '0'.

In direct mode, MSIZE is forced by hardware to the same value as PSIZE as soon as bit EN = '1'.

Bits 12:11 **PSIZE[1:0]**: peripheral data size

These bits are set and cleared by software.
 00: byte (8-bit)
 01: half-word (16-bit)
 10: word (32-bit)
 11: reserved

These bits are protected and can be written only if EN is '0'.

Bit 10 **MINC**: memory increment mode

This bit is set and cleared by software.
 0: memory address pointer is fixed
 1: memory address pointer is incremented after each data transfer (increment is done according to MSIZE)
 This bit is protected and can be written only if EN is '0'.

Bit 9 **PINC**: peripheral increment mode

This bit is set and cleared by software.
 0: peripheral address pointer is fixed
 1: peripheral address pointer is incremented after each data transfer (increment is done according to PSIZE)
 This bit is protected and can be written only if EN is '0'.

Bit 8 **CIRC**: circular mode

This bit is set and cleared by software and can be cleared by hardware.
 0: circular mode disabled
 1: circular mode enabled
 When the peripheral is the flow controller (bit PFCTRL = 1) and the stream is enabled (bit EN = 1), then this bit is automatically forced by hardware to 0.
 It is automatically forced by hardware to 1 if the DBM bit is set, as soon as the stream is enabled (bit EN ='1').

Bits 7:6 **DIR[1:0]**: data transfer direction

These bits are set and cleared by software.
 00: peripheral-to-memory
 01: memory-to-peripheral
 10: memory-to-memory
 11: reserved
 These bits are protected and can be written only if EN is '0'.

Bit 5 **PFCTRL**: peripheral flow controller

This bit is set and cleared by software.
 0: DMA is the flow controller
 1: The peripheral is the flow controller
 This bit is protected and can be written only if EN is '0'.
 When the memory-to-memory mode is selected (bits DIR[1:0]=10), then this bit is automatically forced to 0 by hardware.

Bit 4 **TCIE**: transfer complete interrupt enable

This bit is set and cleared by software.
 0: TC interrupt disabled
 1: TC interrupt enabled

Bit 3 **HTIE**: half transfer interrupt enable

This bit is set and cleared by software.
 0: HT interrupt disabled
 1: HT interrupt enabled

Bit 2 **TEIE**: transfer error interrupt enable

This bit is set and cleared by software.

0: TE interrupt disabled

1: TE interrupt enabled

Bit 1 **DMEIE**: direct mode error interrupt enable

This bit is set and cleared by software.

0: DME interrupt disabled

1: DME interrupt enabled

Bit 0 **EN**: stream enable / flag stream ready when read low

This bit is set and cleared by software.

0: stream disabled

1: stream enabled

This bit may be cleared by hardware:

- on a DMA end of transfer (stream ready to be configured)
- if a transfer error occurs on the AHB master buses
- when the FIFO threshold on memory AHB port is not compatible with the size of the burst

When this bit is read as 0, the software is allowed to program the configuration and FIFO bits registers. It is forbidden to write these registers when the EN bit is read as 1.

Note: Before setting EN bit to '1' to start a new transfer, the event flags corresponding to the stream in DMA_LISR or DMA_HISR register must be cleared.

8.5.6 DMA stream x number of data register (DMA_SxNDTR)

Address offset: 0x14 + 0x18 * x, (x = 0 to 7)

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
NDT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 15:0 **NDT[15:0]**: number of data items to transfer (0 up to 65535)

This register can be written only when the stream is disabled. When the stream is enabled, this register is read-only, indicating the remaining data items to be transmitted. This register decrements after each DMA transfer.

Once the transfer is completed, this register can either stay at zero (when the stream is in normal mode) or be reloaded automatically with the previously programmed value in the following cases:

- when the stream is configured in circular mode.
- when the stream is enabled again by setting EN bit to '1'.

If the value of this register is zero, no transaction can be served even if the stream is enabled.

8.5.7 DMA stream x peripheral address register (DMA_SxPAR)

Address offset: $0x18 + 0x18 * x$, ($x = 0$ to 7)

Reset value: $0x0000\ 0000$

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

Bits 31:0 **PAR[31:0]**: peripheral address

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

These bits are write-protected and can be written only when bit EN = '0' in the DMA_SxCR register.

8.5.8 DMA stream x memory 0 address register (DMA_SxM0AR)

Address offset: $0x1C + 0x18 * x$, ($x = 0$ to 7)

Reset value: $0x0000\ 0000$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
M0A[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M0A[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **M0A[31:0]**: memory 0 address

Base address of memory area 0 from/to which the data is read/written.

These bits are write-protected. They can be written only if:

- the stream is disabled (bit EN= '0' in the DMA_SxCR register) or
- the stream is enabled (EN='1' in DMA_SxCR register) and bit CT = '1' in the DMA_SxCR register (in double-buffer mode).

8.5.9 DMA stream x memory 1 address register (DMA_SxM1AR)

Address offset: $0x20 + 0x18 * x$, ($x = 0$ to 7)

Reset value: $0x0000\ 0000$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
M1A[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M1A[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **M1A[31:0]**: memory 1 address (used in case of double-buffer mode)

Base address of memory area 1 from/to which the data is read/written.

This register is used only for the double-buffer mode.

These bits are write-protected. They can be written only if:

- the stream is disabled (bit EN= '0' in the DMA_SxCR register) or
- the stream is enabled (EN='1' in DMA_SxCR register) and bit CT = '0' in the DMA_SxCR register.

8.5.10 DMA stream x FIFO control register (DMA_SxFCR)

Address offset: $0x24 + 0x24 * x$, ($x = 0$ to 7)

Reset value: 0x0000 0021

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

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

Bit 7 **FEIE**: FIFO error interrupt enable

This bit is set and cleared by software.

0: FE interrupt disabled

1: FE interrupt enabled

Bit 6 Reserved, must be kept at reset value.

Bits 5:3 **FS[2:0]**: FIFO status

These bits are read-only.

000: $0 < \text{fifo_level} < 1/4$

001: $1/4 \leq \text{fifo_level} < 1/2$

010: $1/2 \leq \text{fifo_level} < 3/4$

011: $3/4 \leq \text{fifo_level} < \text{full}$

100: FIFO is empty

101: FIFO is full

others: no meaning

These bits are not relevant in the direct mode (DMDIS bit is zero).

Bit 2 **DMDIS**: direct mode disable

This bit is set and cleared by software. It can be set by hardware.

0: direct mode enabled

1: direct mode disabled

This bit is protected and can be written only if EN is '0'.

This bit is set by hardware if the memory-to-memory mode is selected (DIR bit in DMA_SxCR are "10") and the EN bit in the DMA_SxCR register is '1' because the direct mode is not allowed in the memory-to-memory configuration.

Bits 1:0 FTH[1:0]: FIFO threshold selection

These bits are set and cleared by software.

- 00: 1/4 full FIFO
- 01: 1/2 full FIFO
- 10: 3/4 full FIFO
- 11: full FIFO

These bits are not used in the direct mode when the DMIS value is zero.

These bits are protected and can be written only if EN is '0'.

8.5.11 DMA register map

Table 35 summarizes the DMA registers.

Table 35. DMA register map and reset values

Table 35. DMA register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0x0034	DMA_S1M0AR																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x0038	DMA_S1M1AR																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x003C	DMA_S1FCR	Res	FS[2:0]	MDIS	0	0																													
	Reset value																																		
0x0040	DMA_S2CR	Res	Res	Res	Res																														
	Reset value																																		
0x0044	DMA_S2NDTR	Res	Res	Res	Res																														
	Reset value																																		
0x0048	DMA_S2PAR																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x004C	DMA_S2M0AR																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0050	DMA_S2M1AR																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0054	DMA_S2FCR	Res	Res	Res	Res	Res																													
	Reset value																																		
0x0058	DMA_S3CR	Res	Res	Res	Res	Res																													
	Reset value																																		
0x005C	DMA_S3NDTR	Res	Res	Res	Res	Res																													
	Reset value																																		
0x0060	DMA_S3PAR																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0064	DMA_S3M0AR																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0068	DMA_S3M1AR																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 35. DMA register map and reset values (continued)

Table 35. DMA register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0x00A4	DMA_S6NDTR	Res																																	
	Reset value																																		
0x00A8	DMA_S6PAR																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x00AC	DMA_S6M0AR																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x00B0	DMA_S6M1AR																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x00B4	DMA_S6FCR	Res																																	
	Reset value																																		
0x00B8	DMA_S7CR	Res																																	
	Reset value				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x00BC	DMA_S7NDTR	Res																																	
	Reset value																																		
0x00C0	DMA_S7PAR																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x00C4	DMA_S7M0AR																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x00C8	DMA_S7M1AR																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x00CC	DMA_S7FCR	Res																																	
	Reset value																																		

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

9 Nested vectored interrupt controller (NVIC)

9.1 NVIC features

The nested vector interrupt controller NVIC includes the following features:

- up to 98 maskable interrupt channels for STM32F72xxx and STM32F73xxx (not including the 16 interrupt lines of Cortex®-M7 with FPU)
- 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, refer to programming manual PMxxxx.

9.1.1 SysTick calibration value register

The SysTick calibration value is fixed to 18750, which gives a reference time base of 1 ms with the SysTick clock set to 18.75 MHz (HCLK/8, with HCLK set to 150 MHz).

9.1.2 Interrupt and exception vectors

See [Table 36](#), for the vector table for the STM32F72xxx and STM32F73xxx devices.

Table 36. STM32F72xxx and STM32F73xxx 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

Table 36. STM32F72xxx and STM32F73xxx vector table (continued)

Position	Priority	Type of priority	Acronym	Description	Address
-	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
-	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	TAMP_STAMP	Tamper andTimeStamp interrupts through the EXTI line	0x0000 0048
3	10	settable	RTC_WKUP	RTC Wakeup interrupt through the EXTI line	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	DMA1_Stream0	DMA1 Stream0 global interrupt	0x0000 006C
12	19	settable	DMA1_Stream1	DMA1 Stream1 global interrupt	0x0000 0070
13	20	settable	DMA1_Stream2	DMA1 Stream2 global interrupt	0x0000 0074
14	21	settable	DMA1_Stream3	DMA1 Stream3 global interrupt	0x0000 0078
15	22	settable	DMA1_Stream4	DMA1 Stream4 global interrupt	0x0000 007C
16	23	settable	DMA1_Stream5	DMA1 Stream5 global interrupt	0x0000 0080
17	24	settable	DMA1_Stream6	DMA1 Stream6 global interrupt	0x0000 0084
18	25	settable	ADC	ADC1, ADC2 and ADC3 global interrupts	0x0000 0088
19	26	settable	CAN1_TX	CAN1 TX interrupts	0x0000 008C
20	27	settable	CAN1_RX0	CAN1 RX0 interrupts	0x0000 0090
21	28	settable	CAN1_RX1	CAN1 RX1 interrupt	0x0000 0094

Table 36. STM32F72xxx and STM32F73xxx vector table (continued)

Position	Priority	Type of priority	Acronym	Description	Address
22	29	settable	CAN1_SCE	CAN1 SCE interrupt	0x0000 0098
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000 009C
24	31	settable	TIM1_BRK_TIM9	TIM1 Break interrupt and TIM9 global interrupt	0x0000 00A0
25	32	settable	TIM1_UP_TIM10	TIM1 Update interrupt and TIM10 global interrupt	0x0000 00A4
26	33	settable	TIM1_TRG_COM_TIM11	TIM1 Trigger and Commutation interrupts and TIM11 global interrupt	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
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	RTC_Alarm	RTC Alarms (A and B) through EXTI line interrupt	0x0000 00E4
42	49	settable	OTG_FS_WKUP	USB On-The-Go FS Wakeup through EXTI line interrupt	0x0000 00E8
43	50	settable	TIM8_BRK_TIM12	TIM8 Break interrupt and TIM12 global interrupt	0x0000 00EC
44	51	settable	TIM8_UP_TIM13	TIM8 Update interrupt and TIM13 global interrupt	0x0000 00F0
45	52	settable	TIM8_TRG_COM_TIM14	TIM8 Trigger and Commutation interrupts and TIM14 global interrupt	0x0000 00F4
46	53	settable	TIM8_CC	TIM8 Capture Compare interrupt	0x0000 00F8

Table 36. STM32F72xxx and STM32F73xxx vector table (continued)

Position	Priority	Type of priority	Acronym	Description	Address
47	54	settable	DMA1_Stream7	DMA1 Stream7 global interrupt	0x0000 00FC
48	55	settable	FSMC	FSMC global interrupt	0x0000 0100
49	56	settable	SDMMC1	SDMMC1 global interrupt	0x0000 0104
50	57	settable	TIM5	TIM5 global interrupt	0x0000 0108
51	58	settable	SPI3	SPI3 global interrupt	0x0000 010C
52	59	settable	UART4	UART4 global interrupt	0x0000 0110
53	60	settable	UART5	UART5 global interrupt	0x0000 0114
54	61	settable	TIM6_DAC	TIM6 global interrupt, DAC1 and DAC2 underrun error interrupts	0x0000 0118
55	62	settable	TIM7	TIM7 global interrupt	0x0000 011C
56	63	settable	DMA2_Stream0	DMA2 Stream0 global interrupt	0x0000 0120
57	64	settable	DMA2_Stream1	DMA2 Stream1 global interrupt	0x0000 0124
58	65	settable	DMA2_Stream2	DMA2 Stream2 global interrupt	0x0000 0128
59	66	settable	DMA2_Stream3	DMA2 Stream3 global interrupt	0x0000 012C
60	67	settable	DMA2_Stream4	DMA2 Stream4 global interrupt	0x0000 0130
61	68	-	-	Reserved	0x0000 0134
62	69	-	-	Reserved	0x0000 0138
63	70	-	-	Reserved	0x0000 013C
64	71	-	-	Reserved	0x0000 0140
65	72	-	-	Reserved	0x0000 0144
66	73	-	-	Reserved	0x0000 0148
67	74	settable	OTG_FS	USB On The Go FS global interrupt	0x0000 014C
68	75	settable	DMA2_Stream5	DMA2 Stream5 global interrupt	0x0000 0150
69	76	settable	DMA2_Stream6	DMA2 Stream6 global interrupt	0x0000 0154
70	77	settable	DMA2_Stream7	DMA2 Stream7 global interrupt	0x0000 0158
71	78	settable	USART6	USART6 global interrupt	0x0000 015C
72	79	settable	I2C3_EV	I ² C3 event interrupt	0x0000 0160
73	80	settable	I2C3_ER	I ² C3 error interrupt	0x0000 0164
74	81	settable	OTG_HS_EP1_OUT	USB On The Go HS End Point 1 Out global interrupt	0x0000 0168

Table 36. STM32F72xxx and STM32F73xxx vector table (continued)

Position	Priority	Type of priority	Acronym	Description	Address
75	82	settable	OTG_HS_EP1_IN	USB On The Go HS End Point 1 In global interrupt	0x0000 016C
76	83	settable	OTG_HS_WKUP	USB On The Go HS Wakeup through EXTI interrupt	0x0000 0170
77	84	settable	OTG_HS	USB On The Go HS global interrupt	0x0000 0174
78	85	-	-	Reserved	0x0000 0178
79	86	settable	AES	AES global interrupt	0x0000 017C
80	87	settable	RNG	Rng global interrupt	0x0000 0180
81	88	settable	FPU	FPU global interrupt	0x0000 0184
82	89	settable	UART7	UART7 global interrupt	0x0000 0188
83	90	settable	UART8	UART8 global interrupt	0x0000 018C
84	91	settable	SPI4	SPI4 global interrupt	0x0000 0190
85	92	settable	SPI5	SPI5 global interrupt	0x0000 0194
86	93	-	-	Reserved	0x0000 0198
87	94	settable	SAI1	SAI1 global interrupt	0x0000 019C
88	95	-	-	Reserved	0x0000 01A0
89	96	-	-	Reserved	0x0000 01A4
90	97	-	-	Reserved	0x0000 01A8
91	98	settable	SAI2	SAI2 global interrupt	0x0000 01AC
92	99	settable	QuadSPI	QuadSPI global interrupt	0x0000 01B0
93	100	settable	LP Timer1	LP Timer1 global interrupt	0x0000 01B4
94	101	-	-	Reserved	0x0000 01B8
95	102	-	-	Reserved	0x0000 01BC
96	103	-	-	Reserved	0x0000 01C0
97	104	-	-	Reserved	0x0000 01C4
103	110	settable	SDMMC2	SDMMC2 global interrupt	0x0000 01DC

10 Extended interrupts and events controller (EXTI)

The external interrupt/event controller consists of up to 24 edge detectors for generating event/interrupt requests. Each input line can be independently configured to select the type (interrupt or event) 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.

10.1 EXTI main features

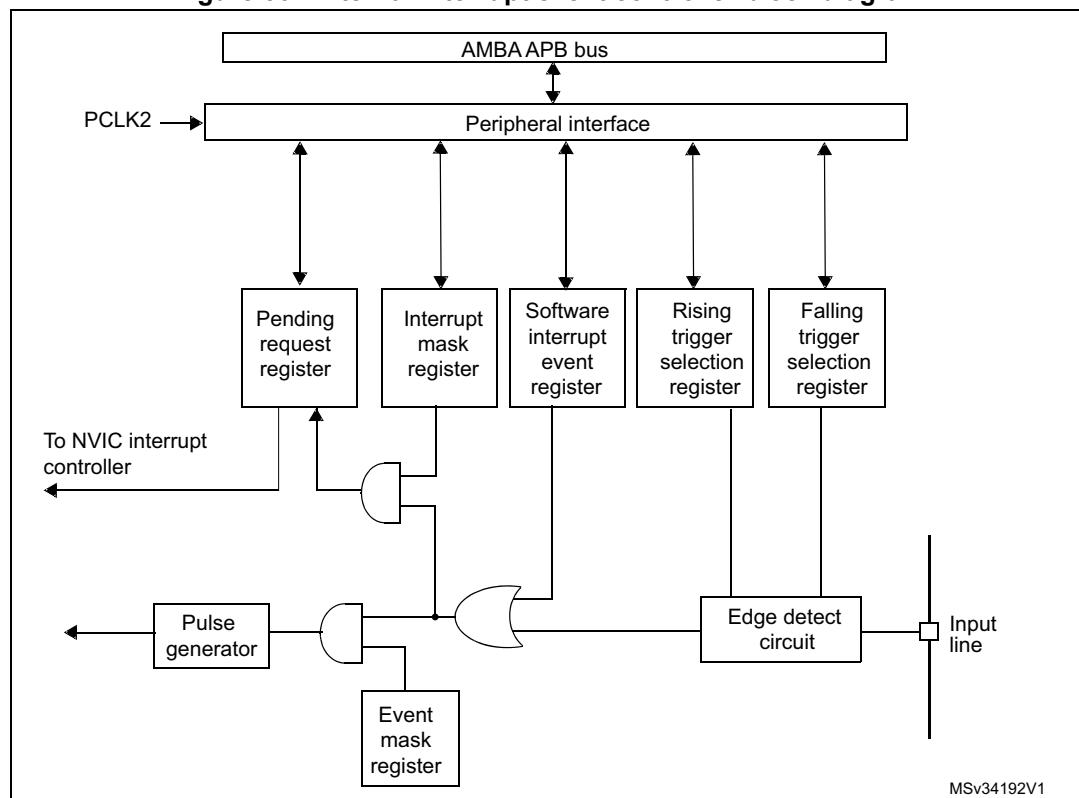
The main features of the EXTI controller are the following:

- independent trigger and mask on each interrupt/event line
- dedicated status bit for each interrupt line
- generation of up to 24 software event/interrupt requests
- detection of external signals with a pulse width lower than the APB2 clock period. Refer to the electrical characteristics section of the STM32F72xxx and STM32F73xxx datasheets for details on this parameter.

10.2 EXTI block diagram

Figure 30 shows the block diagram.

Figure 30. External interrupt/event controller block diagram



10.3 Wakeup event management

The STM32F72xxx and STM32F73xxx devices are able to handle external or internal events in order to wake up the core (WFE). The wakeup event can be generated either by:

- enabling an interrupt in the peripheral control register but not in the NVIC, and enabling the SEVONPEND bit in the Cortex®-M7 System Control register. When the MCU resumes from WFE, the peripheral interrupt pending bit and the peripheral NVIC IRQ channel pending bit (in the NVIC interrupt clear pending register) have to be cleared.
- or configuring an external or internal EXTI line in event mode. When the CPU resumes from WFE, it is not necessary to clear the peripheral interrupt pending bit or the NVIC IRQ channel pending bit as the pending bit corresponding to the event line is not set.

To use an external line as a wakeup event, refer to [Section 10.4: Functional description](#).

10.4 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.

10.5 Hardware interrupt selection

To configure a line as interrupt sources, use the following procedure:

1. Configure the corresponding mask bit (EXTI_IMR)
2. Configure the Trigger selection bits of the interrupt lines (EXTI_RTSR and EXTI_FTSR)
3. 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 24 lines can be correctly acknowledged.

10.6 Hardware event selection

To configure a line as event sources, use the following procedure:

1. Configure the corresponding mask bit (EXTI_EMR)
2. Configure the Trigger selection bits of the event line (EXTI_RTSR and EXTI_FTSR)

10.7 Software interrupt/event selection

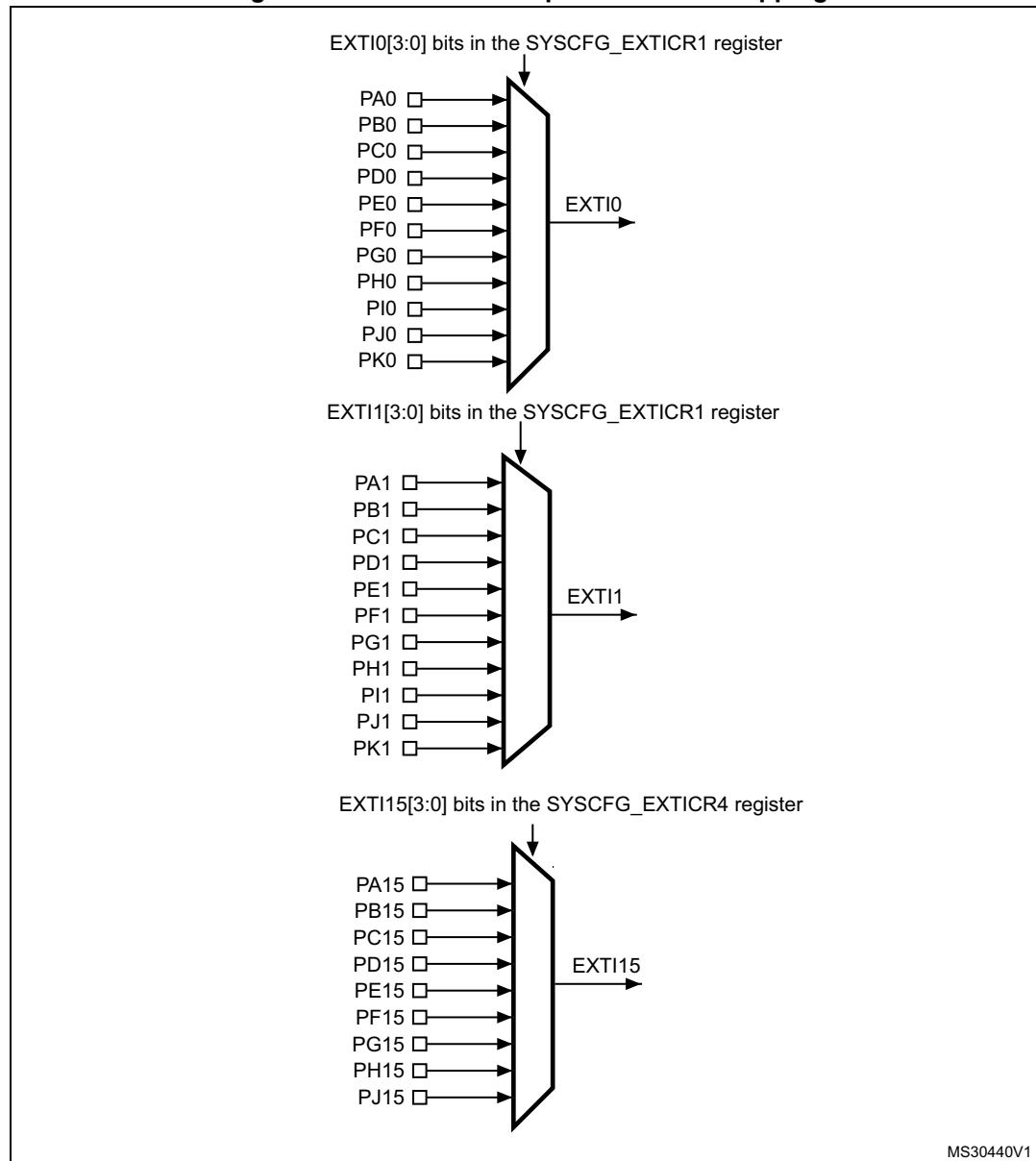
The line can be configured as software interrupt/event line. The following is the procedure to generate a software interrupt.

1. Configure the corresponding mask bit (EXTI_IMR, EXTI_EMR)
2. Set the required bit in the software interrupt register (EXTI_SWIER)

10.8 External interrupt/event line mapping

Up to 168 GPIOs are connected to the 16 external interrupt/event lines in the following manner:

Figure 31. External interrupt/event GPIO mapping



MS30440V1

The eight 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 OTG FS Wakeup event
- EXTI line 19 is reserved
- EXTI line 20 is connected to the USB OTG HS (configured in FS) Wakeup event
- EXTI line 21 is connected to the RTC Tamper and TimeStamp events
- EXTI line 22 is connected to the RTC Wakeup event
- EXTI line 23 is connected to the LPTIM1 asynchronous event

10.9 EXTI registers

Refer to [Section 1.2 on page 53](#) for a list of abbreviations used in register descriptions.

10.9.1 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
Res.	IM23	IM22	IM21	IM20	IM19	IM18	IM17	IM16							
								rw							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IM15	IM14	IM13	IM12	IM11	IM10	IM9	IM8	IM7	IM6	IM5	IM4	IM3	IM2	IM1	IM0
rw															

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

Bits 23:0 **IMx**: Interrupt mask on line x

- 0: Interrupt request from line x is masked
- 1: Interrupt request from line x is not masked

10.9.2 Event mask register (EXTI_EMR)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	EM23	EM22	EM21	EM20	EM19	EM18	EM17	EM16							
								rw							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EM15	EM14	EM13	EM12	EM11	EM10	EM9	EM8	EM7	EM6	EM5	EM4	EM3	EM2	EM1	EM0
rw															

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

Bits 23:0 **EMx**: Event mask on line x

- 0: Event request from line x is masked
- 1: Event request from line x is not masked

10.9.3 Rising trigger selection register (EXTI_RTSR)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	TR23	TR22	TR21	TR20	TR19	TR18	TR17	TR16							
								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															

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

Bits 23:0 **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 glitch must be generated on these lines. If a rising edge occurs on the external interrupt line while writing to the EXTI_RTSR register, the pending bit is set. Rising and falling edge triggers can be set for the same interrupt line. In this configuration, both generate a trigger condition.*

10.9.4 Falling trigger selection register (EXTI_FTSR)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	TR23	TR22	TR21	TR20	TR19	TR18	TR17	TR16							
								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															

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

Bits 23:0 **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 glitch must be generated on these lines. If a falling edge occurs on the external interrupt line while writing to the EXTI_FTSR register, the pending bit is not set.

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

10.9.5 Software interrupt event register (EXTI_SWIER)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SWIER 23	SWIER 22	SWIER 21	SWIER 20	SWIER 19	SWIER 18	SWIER 17	SWIER 16
								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 31:24 Reserved, must be kept at reset value.

Bits 23:0 **SWIER_x:** Software Interrupt on line x

If interrupt are enabled on line x in the EXTI_IMR register, writing '1' to SWIER_x bit when it is set at '0' sets the corresponding pending bit in the EXTI_PR register, thus resulting in an interrupt request generation.

This bit is cleared by clearing the corresponding bit in EXTI_PR (by writing a 1 to the bit).

10.9.6 Pending register (EXTI_PR)

Address offset: 0x14

Reset value: undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	PR23	PR22	PR21	PR20	PR19	PR18	PR17	PR16							
								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															

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

Bits 23:0 **PR_x:** 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 programming it to '1'.

10.9.7 EXTI register map

[Table 37](#) gives the EXTI register map and the reset values.

Table 37. External interrupt/event controller register map and reset values

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

11 Cyclic redundancy check calculation unit (CRC)

11.1 Introduction

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

Among other applications, CRC-based techniques are used to verify data transmission or storage integrity. In the scope of the functional safety standards, they offer a means of verifying the Flash memory integrity. The CRC calculation unit helps compute a signature of the software during runtime, to be compared with a reference signature generated at link time and stored at a given memory location.

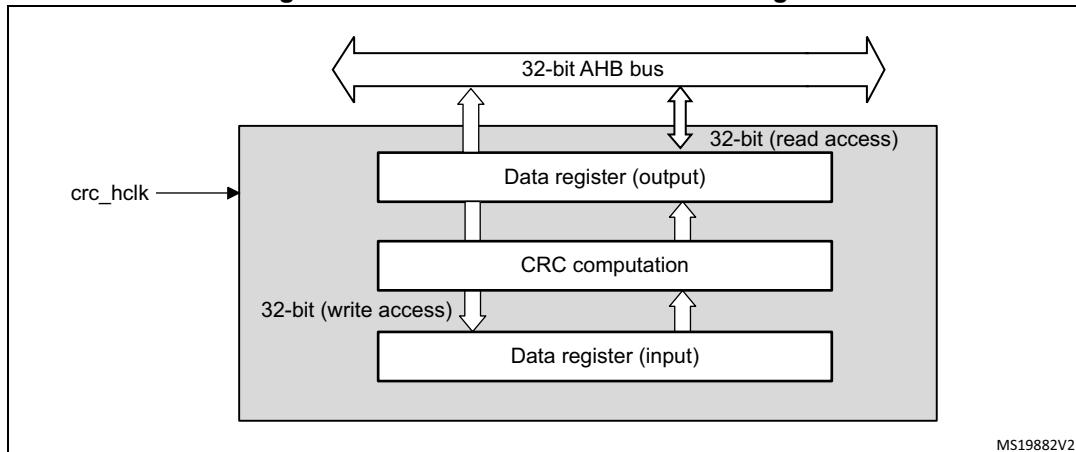
11.2 CRC main features

- Uses CRC-32 (Ethernet) polynomial: 0x4C11DB7
$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$
- Alternatively, uses fully programmable polynomial with programmable size (7, 8, 16, 32 bits)
- Handles 8-, 16-, 32-bit data size
- Programmable CRC initial value
- Single input/output 32-bit data register
- Input buffer to avoid bus stall during calculation
- CRC computation done in 4 AHB clock cycles (HCLK) for the 32-bit data size
- General-purpose 8-bit register (can be used for temporary storage)
- Reversibility option on I/O data

11.3 CRC functional description

11.3.1 CRC block diagram

Figure 32. CRC calculation unit block diagram



11.3.2 CRC internal signals

Table 38. CRC internal input/output signals

Signal name	Signal type	Description
crc_hclk	Digital input	AHB clock

11.3.3 CRC operation

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

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

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

The duration of the computation depends on data width:

- 4 AHB clock cycles for 32-bit
- 2 AHB clock cycles for 16-bit
- 1 AHB clock cycles for 8-bit

An input buffer allows to immediately write a second data without waiting for any wait states due to the previous CRC calculation.

The data size can be dynamically adjusted to minimize the number of write accesses for a given number of bytes. For instance, a CRC for 5 bytes can be computed with a word write followed by a byte write.

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

For example: input data 0x1A2B3C4D is used for CRC calculation as:

- 0x58D43CB2 with bit-reversal done by byte
- 0xD458B23C with bit-reversal done by half-word
- 0xB23CD458 with bit-reversal done on the full word

The output data can also be reversed by setting the REV_OUT bit in the CRC_CR register.

The operation is done at bit level: for example, output data 0x11223344 is converted into 0x22CC4488.

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

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

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

Polynomial programmability

The polynomial coefficients are fully programmable through the CRC_POL register, and the polynomial size can be configured to be 7, 8, 16 or 32 bits by programming the POLYSIZE[1:0] bits in the CRC_CR register. Even polynomials are not supported.

If the CRC data is less than 32-bit, its value can be read from the least significant bits of the CRC_DR register.

To obtain a reliable CRC calculation, the change on-fly of the polynomial value or size can not be performed during a CRC calculation. As a result, if a CRC calculation is ongoing, the application must either reset it or perform a CRC_DR read before changing the polynomial.

The default polynomial value is the CRC-32 (Ethernet) polynomial: 0x4C11DB7.

11.4 CRC registers

11.4.1 Data register (CRC_DR)

Address offset: 0x00

Reset value: 0xFFFF FFFF

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

Bits 31:0 **DR[31:0]:** Data register bits

This register is used to write new data to the CRC calculator.

It holds the previous CRC calculation result when it is read.

If the data size is less than 32 bits, the least significant bits are used to write/read the correct value.

11.4.2 Independent data register (CRC_IDR)

Address offset: 0x04

Reset value: 0x0000 0000

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

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

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

These bits can be used as a temporary storage location for one byte.

This register is not affected by CRC resets generated by the RESET bit in the CRC_CR register

11.4.3 Control register (CRC_CR)

Address offset: 0x08

Reset value: 0x0000 0000

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

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

Bit 7 **REV_OUT**: Reverse output data

This bit controls the reversal of the bit order of the output data.

0: Bit order not affected

1: Bit-reversed output format

Bits 6:5 **REV_IN[1:0]**: Reverse input data

These bits control the reversal of the bit order of the input data

00: Bit order not affected

01: Bit reversal done by byte

10: Bit reversal done by half-word

11: Bit reversal done by word

Bits 4:3 **POLYSIZE[1:0]**: Polynomial size

These bits control the size of the polynomial.

00: 32 bit polynomial

01: 16 bit polynomial

10: 8 bit polynomial

11: 7 bit polynomial

Bits 2:1 Reserved, must be kept at reset value.

Bit 0 **RESET**: RESET bit

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

11.4.4 Initial CRC value (CRC_INIT)

Address offset: 0x10

Reset value: 0xFFFF FFFF

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

Bits 31:0 **CRC_INIT[31:0]**: Programmable initial CRC value

This register is used to write the CRC initial value.

11.4.5 CRC polynomial (CRC_POL)

Address offset: 0x14

Reset value: 0x04C1 1DB7

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
POL[31:16]															
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
POL[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **POL[31:0]**: Programmable polynomial

This register is used to write the coefficients of the polynomial to be used for CRC calculation.

If the polynomial size is less than 32 bits, the least significant bits have to be used to program the correct value.

11.4.6 CRC register map

Table 39. CRC register map and reset values

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	CRC_DR	DR[31:0]																															
		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1			
0x04	CRC_IDR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
0x08	CRC_CR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
0x10	CRC_INIT	CRC_INIT[31:0]																															
		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		
0x14	CRC_POL	Polynomial coefficients																															
		0x04C1 1DB7																															

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

12 Flexible memory controller (FMC)

The Flexible memory controller (FMC) includes three memory controllers:

- The NOR/PSRAM memory controller
- The NAND memory controller
- The Synchronous DRAM (SDRAM/Mobile LPDDR SDRAM) controller

12.1 FMC main features

The FMC functional block makes the interface with: synchronous and asynchronous static memories, SDRAM memories, and NAND Flash memory. Its main purposes are:

- to translate AHB transactions into the appropriate external device protocol
- to meet the access time requirements of the external memory devices

All external memories share the addresses, data and control signals with the controller. Each external device is accessed by means of a unique chip select. The FMC performs only one access at a time to an external device.

The main features of the FMC controller are the following:

- Interface with static-memory mapped devices including:
 - Static random access memory (SRAM)
 - NOR Flash memory/OneNAND Flash memory
 - PSRAM (4 memory banks)
 - NAND Flash memory with ECC hardware to check up to 8 Kbytes of data
- Interface with synchronous DRAM (SDRAM/Mobile LPDDR SDRAM) memories
- Interface with parallel LCD modules, supporting Intel 8080 and Motorola 6800 modes.
- Burst mode support for faster access to synchronous devices such as NOR Flash memory, PSRAM and SDRAM)
- Programmable continuous clock output for asynchronous and synchronous accesses
- 8-, 16- or 32-bit wide data bus
- Independent chip select control for each memory bank
- Independent configuration for each memory bank
- Write enable and byte lane select outputs for use with PSRAM, SRAM and SDRAM devices
- External asynchronous wait control
- Write FIFO with 16 x32-bit depth
- Cacheable Read FIFO with 6 x32-bit depth (6 x14-bit address tag) for SDRAM controller.

The Write FIFO is common to all memory controllers and consists of:

- a Write Data FIFO which stores the AHB data to be written to the memory (up to 32 bits) plus one bit for the AHB transfer (burst or not sequential mode)
- a Write Address FIFO which stores the AHB address (up to 28 bits) plus the AHB data size (up to 2 bits). When operating in burst mode, only the start address is stored except when crossing a page boundary (for PSRAM and SDRAM). In this case, the AHB burst is broken into two FIFO entries.

The Write FIFO can be disabled by setting the WFDIS bit in the FMC_BCR1 register.

At startup the FMC pins must be configured by the user application. The FMC I/O pins which are not used by the application can be used for other purposes.

The FMC registers that define the external device type and associated characteristics are usually set at boot time and do not change until the next reset or power-up. However, the settings can be changed at any time.

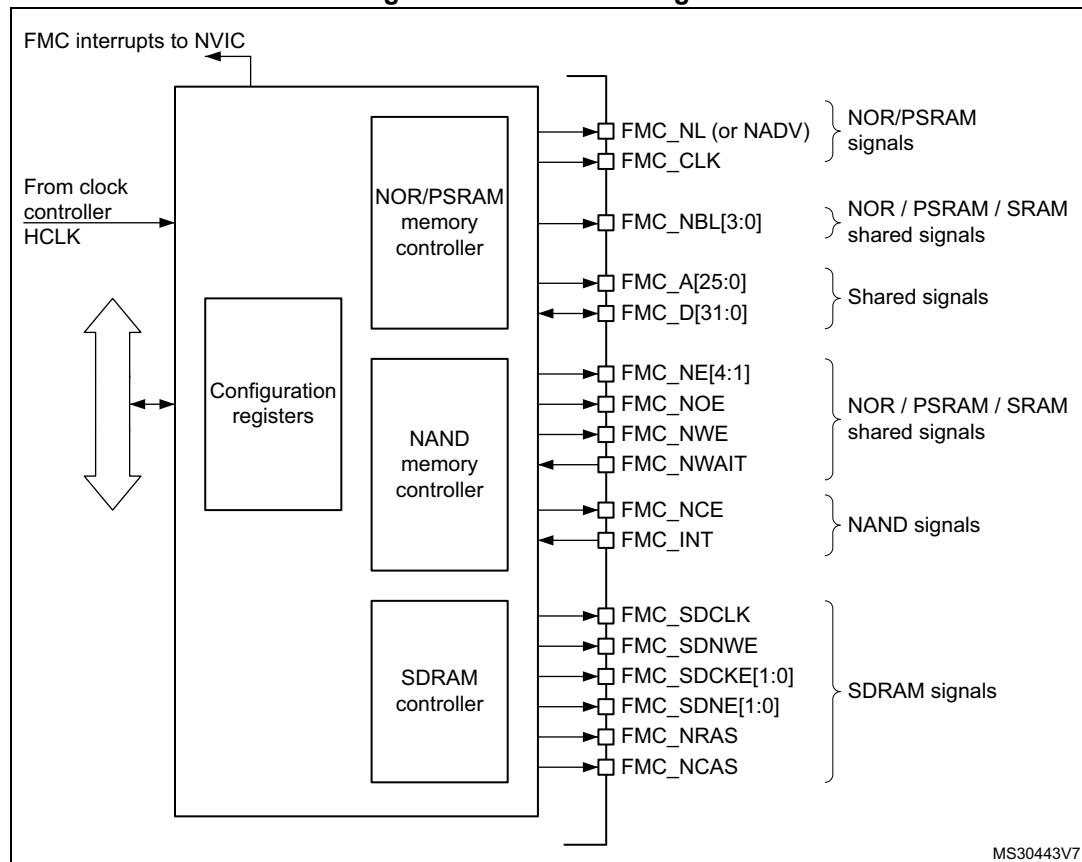
12.2 FMC block diagram

The FMC consists of the following main blocks:

- The AHB interface (including the FMC configuration registers)
- The NOR Flash/PSRAM/SRAM controller
- The SDRAM controller
- The NAND Flash controller

The block diagram is shown in the figure below.

Figure 33. FMC block diagram



MS30443V7

12.3 AHB interface

The AHB slave interface allows internal CPUs and other bus master peripherals to access the external memories.

AHB transactions are translated into the external device protocol. In particular, if the selected external memory is 16- or 8-bit wide, 32-bit wide transactions on the AHB are split into consecutive 16- or 8-bit accesses. The FMC chip select (FMC_NEx) does not toggle between the consecutive accesses except in case of Access mode D when the Extended mode is enabled.

The FMC generates an AHB error in the following conditions:

- When reading or writing to an FMC bank (Bank 1 to 4) which is not enabled.
- When reading or writing to the NOR Flash bank while the FACCEN bit is reset in the FMC_BCRx register.
- When writing to a write protected SDRAM bank (WP bit set in the SDRAM_SDCRx register).
- When the SDRAM address range is violated (access to reserved address range)

The effect of an AHB error depends on the AHB master which has attempted the R/W access:

- If the access has been attempted by the Cortex®-M7 CPU, a hard fault interrupt is generated.
- If the access has been performed by a DMA controller, a DMA transfer error is generated and the corresponding DMA channel is automatically disabled.

The AHB clock (HCLK) is the reference clock for the FMC.

12.3.1 Supported memories and transactions

General transaction rules

The requested AHB transaction data size can be 8-, 16- or 32-bit wide whereas the accessed external device has a fixed data width. This may lead to inconsistent transfers.

Therefore, some simple transaction rules must be followed:

- AHB transaction size and memory data size are equal
There is no issue in this case.
- AHB transaction size is greater than the memory size:
In this case, the FMC splits the AHB transaction into smaller consecutive memory accesses to meet the external data width. The FMC chip select (FMC_NEx) does not toggle between the consecutive accesses.
- AHB transaction size is smaller than the memory size:
The transfer may or not be consistent depending on the type of external device:
 - Accesses to devices that have the byte select feature (SRAM, ROM, PSRAM, SDRAM)
In this case, the FMC allows read/write transactions and accesses the right data through its byte lanes NBL[3:0].
Bytes to be written are addressed by NBL[3:0].
All memory bytes are read (NBL[3:0] are driven low during read transaction) and the useless ones are discarded.
 - Accesses to devices that do not have the byte select feature (NOR and NAND Flash memories)
This situation occurs when a byte access is requested to a 16-bit wide Flash memory. Since the device cannot be accessed in Byte mode (only 16-bit words can be read/written from/to the Flash memory), Write transactions and Read transactions are allowed (the controller reads the entire 16-bit memory word and uses only the required byte).

Wrap support for NOR Flash/PSRAM and SDRAM

The synchronous memories must be configured in Linear burst mode of undefined length as not all masters can issue a wrap transactions.

If a master generates an AHB wrap transaction:

- The read is split into two linear burst transactions.
- The write is split into two linear burst transactions if the write fifo is enabled and into several linear burst transactions if the write fifo is disabled.

Configuration registers

The FMC can be configured through a set of registers. Refer to [Section 12.5.6](#), for a detailed description of the NOR Flash/PSRAM controller registers. Refer to [Section 12.6.7](#), for a detailed description of the NAND Flash registers and to [Section 12.7.5](#) for a detailed description of the SDRAM controller registers.

12.4 External device address mapping

From the FMC point of view, the external memory is divided into fixed-size banks of 256 Mbytes each (see [Figure 34](#)):

- Bank 1 used to address up to 4 NOR Flash memory or PSRAM devices. This bank is split into 4 NOR/PSRAM subbanks with 4 dedicated chip selects, as follows:
 - Bank 1 - NOR/PSRAM 1
 - Bank 1 - NOR/PSRAM 2
 - Bank 1 - NOR/PSRAM 3
 - Bank 1 - NOR/PSRAM 4
- Bank 3 used to address NAND Flash memory devices. The MPU memory attribute for this space must be reconfigured by software to Device.
- Bank 4 and 5 used to address SDRAM devices (1 device per bank).

For each bank the type of memory to be used can be configured by the user application through the Configuration register.

Figure 34. FMC memory banks

Address	Bank	Supported memory type
0x6000 0000	Bank 1 4 x 64 MB	NOR/PSRAM/ SRAM
0x6FFF FFFF		
0x7000 0000	Bank 2 Not used	
0x7FFF FFFF		
0x8000 0000	Bank 3 4 x 64 MB	NAND Flash memory
0x8FFF FFFF		
0x9000 0000	Bank 4 Not used	
0x9FFF FFFF		
0xC000 0000	SDRAM Bank 1 4 x 64 MB	
0xCFFF FFFF		
0xD000 0000	SDRAM Bank 2 4 x 64 MB	
0xFFFF FFFF		

MSv30444V5

12.4.1 NOR/PSRAM address mapping

HADDR[27:26] bits are used to select one of the four memory banks as shown in [Table 40](#).

Table 40. NOR/PSRAM bank selection

HADDR[27:26] ⁽¹⁾	Selected bank
00	Bank 1 - NOR/PSRAM 1
01	Bank 1 - NOR/PSRAM 2
10	Bank 1 - NOR/PSRAM 3
11	Bank 1 - NOR/PSRAM 4

1. HADDR are internal AHB address lines that are translated to external memory.

The HADDR[25:0] bits contain the external memory address. Since HADDR is a byte address whereas the memory is addressed at word level, the address actually issued to the memory varies according to the memory data width, as shown in the following table.

Table 41. NOR/PSRAM External memory address

Memory width ⁽¹⁾	Data address issued to the memory	Maximum memory capacity (bits)
8-bit	HADDR[25:0]	64 Mbytes x 8 = 512 Mbit
16-bit	HADDR[25:1] >> 1	64 Mbytes/2 x 16 = 512 Mbit
32-bit	HADDR[25:2] >> 2	64 Mbytes/4 x 32 = 512 Mbit

1. In case of a 16-bit external memory width, the FMC will internally use HADDR[25:1] to generate the address for external memory FMC_A[24:0]. In case of a 32-bit memory width, the FMC will internally use HADDR[25:2] to generate the external address.
Whatever the external memory width, FMC_A[0] should be connected to external memory address A[0].

12.4.2 NAND Flash memory address mapping

The NAND bank is divided into memory areas as indicated in [Table 42](#).

Table 42. NAND memory mapping and timing registers

Start address	End address	FMC bank	Memory space	Timing register
0x8800 0000	0x8BFF FFFF	Bank 3 - NAND Flash	Attribute	FMC_PATT (0x8C)
0x8000 0000	0x83FF FFFF		Common	FMC_PMEM (0x88)

For NAND Flash memory, the common and attribute memory spaces are subdivided into three sections (see in [Table 43](#) below) located in the lower 256 Kbytes:

- Data section (first 64 Kbytes in the common/attribute memory space)
- Command section (second 64 Kbytes in the common / attribute memory space)
- Address section (next 128 Kbytes in the common / attribute memory space)

Table 43. NAND bank selection

Section name	HADDR[17:16]	Address range
Address section	1X	0x020000-0x03FFFF
Command section	01	0x010000-0x01FFFF
Data section	00	0x000000-0x0FFFF

The application software uses the 3 sections to access the NAND Flash memory:

- **To sending a command to NAND Flash memory**, the software must write the command value to any memory location in the command section.
- **To specify the NAND Flash address that must be read or written**, the software must write the address value to any memory location in the address section. Since an address can be 4 or 5 bytes long (depending on the actual memory size), several consecutive write operations to the address section are required to specify the full address.
- **To read or write data**, the software reads or writes the data from/to any memory location in the data section.

Since the NAND Flash memory automatically increments addresses, there is no need to increment the address of the data section to access consecutive memory locations.

12.4.3 SDRAM address mapping

The HADDR[28] bit (internal AHB address line 28) is used to select one of the two memory banks as indicated in [Table 44](#).

Table 44. SDRAM bank selection

HADDR[28]	Selected bank	Control register	Timing register
0	SDRAM Bank1	FMC_SDCR1	FMC_SDTR1
1	SDRAM Bank2	FMC_SDCR2	FMC_SDTR2

The following table shows SDRAM mapping for a 13-bit row, a 11-bit column and a 4 internal bank configuration.

Table 45. SDRAM address mapping

Memory width ⁽¹⁾	Internal bank	Row address	Column address ⁽²⁾	Maximum memory capacity (Mbytes)
8-bit	HADDR[25:24]	HADDR[23:11]	HADDR[10:0]	64 Mbytes: 4 x 8K x 2K
16-bit	HADDR[26:25]	HADDR[24:12]	HADDR[11:1]	128 Mbytes: 4 x 8K x 2K x 2
32-bit	HADDR[27:26]	HADDR[25:13]	HADDR[12:2]	256 Mbytes: 4 x 8K x 2K x 4

1. When interfacing with a 16-bit memory, the FMC internally uses the HADDR[11:1] internal AHB address lines to generate the external address. Whatever the memory width, FMC_A[0] has to be connected to the external memory address A[0].
2. The AutoPrecharge is not supported. FMC_A[10] must be connected to the external memory address A[10] but it will be always driven 'low'.

The HADDR[27:0] bits are translated to external SDRAM address depending on the SDRAM controller configuration:

- Data size:8, 16 or 32 bits
- Row size:11, 12 or 13 bits
- Column size: 8, 9, 10 or 11 bits
- Number of internal banks: two or four internal banks

The following tables show the SDRAM address mapping versus the SDRAM controller configuration.

Table 46. SDRAM address mapping with 8-bit data bus width⁽¹⁾⁽²⁾

Row size configuration	HADDR(AHB Internal Address Lines)																																	
	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						
11-bit row size configuration	Res.						Bank [1:0]		Row[10:0]										Column[7:0]															
	Res.						Bank [1:0]		Row[10:0]										Column[8:0]															
	Res.				Bank [1:0]		Row[10:0]										Column[9:0]																	
	Res.			Bank [1:0]		Row[10:0]										Column[10:0]																		
12-bit row size configuration	Res.						Bank [1:0]		Row[11:0]										Column[7:0]															
	Res.						Bank [1:0]		Row[11:0]										Column[8:0]															
	Res.				Bank [1:0]		Row[11:0]										Column[9:0]																	
	Res.			Bank [1:0]		Row[11:0]										Column[10:0]																		
13-bit row size configuration	Res.						Bank [1:0]		Row[12:0]										Column[7:0]															
	Res.						Bank [1:0]		Row[12:0]										Column[8:0]															
	Res.				Bank [1:0]		Row[12:0]										Column[9:0]																	
	Res.			Bank [1:0]		Row[12:0]										Column[10:0]																		

1. BANK[1:0] are the Bank Address BA[1:0]. When only 2 internal banks are used, BA1 must always be set to '0'.

2. Access to Reserved (Res.) address range generates an AHB error.

Table 47. SDRAM address mapping with 16-bit data bus width⁽¹⁾⁽²⁾

Row size Configuration	HADDR(AHB address Lines)																												
	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	
11-bit row size configuration	Res.			Bank [1:0]		Row[10:0]														Column[7:0]				BM0 ⁽³⁾					
	Res.			Bank [1:0]		Row[10:0]														Column[8:0]				BM0					
	Res.			Bank [1:0]		Row[10:0]														Column[9:0]				BM0					
	Res.	Bank [1:0]		Row[10:0]														Column[10:0]				BM0							
12-bit row size configuration	Res.			Bank [1:0]		Row[11:0]														Column[7:0]				BM0					
	Res.			Bank [1:0]		Row[11:0]														Column[8:0]				BM0					
	Res.			Bank [1:0]		Row[11:0]														Column[9:0]				BM0					
	Res.	Bank [1:0]		Row[11:0]														Column[10:0]				BM0							
13-bit row size configuration	Res.			Bank [1:0]		Row[12:0]														Column[7:0]				BM0					
	Res.			Bank [1:0]		Row[12:0]														Column[8:0]				BM0					
	Res.			Bank [1:0]		Row[12:0]														Column[9:0]				BM0					
	Res.	Bank s. [1:0]		Row[12:0]														Column[10:0]				BM0							

1. BANK[1:0] are the Bank Address BA[1:0]. When only 2 internal banks are used, BA1 must always be set to '0'.

2. Access to Reserved space (Res.) generates an AHB error.

3. BM0: is the byte mask for 16-bit access.

Table 48. SDRAM address mapping with 32-bit data bus width⁽¹⁾⁽²⁾

Row size configuration	HADDR(AHB address Lines)																										
	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
11-bit row size configuration	Res.			Bank [1:0]		Row[10:0]														Column[7:0]				BM[1:0] ⁽³⁾			
	Res.			Bank [1:0]		Row[10:0]														Column[8:0]				BM[1:0]			
	Res.			Bank [1:0]		Row[10:0]														Column[9:0]				BM[1:0]			
	Res.	Bank [1:0]		Row[10:0]														Column[10:0]				BM[1:0]					

Table 48. SDRAM address mapping with 32-bit data bus width⁽¹⁾⁽²⁾ (continued)

Row size configuration	HADDR(AHB address Lines)																											
	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
12-bit row size configuration	Res.		Bank [1:0]		Row[11:0]																Column[7:0]				BM[1:0]			
	Res.		Bank [1:0]		Row[11:0]																Column[8:0]				BM[1:0]			
	Res.		Bank [1:0]		Row[11:0]																Column[9:0]				BM[1:0]			
	Res.	Bank [1:0]		Row[11:0]																	Column[10:0]				BM[1:0]			
13-bit row size configuration	Res.		Bank [1:0]		Row[12:0]																Column[7:0]				BM[1:0]			
	Res.		Bank [1:0]		Row[12:0]																Column[8:0]				BM[1:0]			
	Res.		Bank [1:0]		Row[12:0]																Column[9:0]				BM[1:0]			
	Bank [1:0]		Row[12:0]																	Column[10:0]				BM[1:0]				

1. BANK[1:0] are the Bank Address BA[1:0]. When only 2 internal banks are used, BA1 must always be set to '0'.

2. Access to Reserved space (Res.) generates an AHB error.

3. BM[1:0]: is the byte mask for 32-bit access.

12.5 NOR Flash/PSRAM controller

The FMC generates the appropriate signal timings to drive the following types of memories:

- Asynchronous SRAM and ROM
 - 8 bits
 - 16 bits
 - 32 bits
- PSRAM (CellularRAM™)
 - Asynchronous mode
 - Burst mode for synchronous accesses
 - Multiplexed or non-multiplexed
- NOR Flash memory
 - Asynchronous mode
 - Burst mode for synchronous accesses
 - Multiplexed or non-multiplexed

The FMC outputs a unique chip select signal, NE[4:1], per bank. All the other signals (addresses, data and control) are shared.

The FMC supports a wide range of devices through a programmable timings among which:

- Programmable wait states (up to 15)
- Programmable bus turnaround cycles (up to 15)
- Programmable output enable and write enable delays (up to 15)
- Independent read and write timings and protocol to support the widest variety of memories and timings
- Programmable continuous clock (FMC_CLK) output.

The FMC Clock (FMC_CLK) is a submultiple of the HCLK clock. It can be delivered to the selected external device either during synchronous accesses only or during asynchronous and synchronous accesses depending on the CCKEN bit configuration in the FMC_BCR1 register:

- If the CCLKEN bit is reset, the FMC generates the clock (CLK) only during synchronous accesses (Read/write transactions).
- If the CCLKEN bit is set, the FMC generates a continuous clock during asynchronous and synchronous accesses. To generate the FMC_CLK continuous clock, Bank 1 must be configured in Synchronous mode (see [Section 12.5.6: NOR/PSRAM controller registers](#)). Since the same clock is used for all synchronous memories, when a continuous output clock is generated and synchronous accesses are performed, the AHB data size has to be the same as the memory data width (MWID) otherwise the FMC_CLK frequency will be changed depending on AHB data transaction (refer to [Section 12.5.5: Synchronous transactions](#) for FMC_CLK divider ratio formula).

The size of each bank is fixed and equal to 64 Mbytes. Each bank is configured through dedicated registers (see [Section 12.5.6: NOR/PSRAM controller registers](#)).

The programmable memory parameters include access times (see [Table 49](#)) and support for wait management (for PSRAM and NOR Flash accessed in Burst mode).

Table 49. Programmable NOR/PSRAM access parameters

Parameter	Function	Access mode	Unit	Min.	Max.
Address setup	Duration of the address setup phase	Asynchronous	AHB clock cycle (HCLK)	0	15
Address hold	Duration of the address hold phase	Asynchronous, muxed I/Os	AHB clock cycle (HCLK)	1	15
Data setup	Duration of the data setup phase	Asynchronous	AHB clock cycle (HCLK)	1	256
Bust turn	Duration of the bus turnaround phase	Asynchronous and synchronous read / write	AHB clock cycle (HCLK)	0	15
Clock divide ratio	Number of AHB clock cycles (HCLK) to build one memory clock cycle (CLK)	Synchronous	AHB clock cycle (HCLK)	2	16
Data latency	Number of clock cycles to issue to the memory before the first data of the burst	Synchronous	Memory clock cycle (CLK)	2	17

12.5.1 External memory interface signals

Table 50, *Table 51* and *Table 52* list the signals that are typically used to interface with NOR Flash memory, SRAM and PSRAM.

Note: The prefix “N” identifies the signals that are active low.

NOR Flash memory, non-multiplexed I/Os

Table 50. Non-multiplexed I/O NOR Flash memory

FMC signal name	I/O	Function
CLK	O	Clock (for synchronous access)
A[25:0]	O	Address bus
D[31:0]	I/O	Bidirectional data bus
NE[x]	O	Chip select, x = 1..4
NOE	O	Output enable
NWE	O	Write enable
NL(=NADV)	O	Latch enable (this signal is called address valid, NADV, by some NOR Flash devices)
NWAIT	I	NOR Flash wait input signal to the FMC

The maximum capacity is 512 Mbits (26 address lines).

NOR Flash memory, 16-bit multiplexed I/Os

Table 51. 16-bit multiplexed I/O NOR Flash memory

FMC signal name	I/O	Function
CLK	O	Clock (for synchronous access)
A[25:16]	O	Address bus
AD[15:0]	I/O	16-bit multiplexed, bidirectional address/data bus (the 16-bit address A[15:0] and data D[15:0] are multiplexed on the databus)
NE[x]	O	Chip select, x = 1..4
NOE	O	Output enable
NWE	O	Write enable
NL(=NADV)	O	Latch enable (this signal is called address valid, NADV, by some NOR Flash devices)
NWAIT	I	NOR Flash wait input signal to the FMC

The maximum capacity is 512 Mbits.

PSRAM/SRAM, non-multiplexed I/Os

Table 52. Non-multiplexed I/Os PSRAM/SRAM

FMC signal name	I/O	Function
CLK	O	Clock (only for PSRAM synchronous access)
A[25:0]	O	Address bus
D[31:0]	I/O	Data bidirectional bus
NE[x]	O	Chip select, x = 1..4 (called NCE by PSRAM (CellularRAM™ i.e. CRAM))
NOE	O	Output enable
NWE	O	Write enable
NL(= NADV)	O	Address valid only for PSRAM input (memory signal name: NADV)
NWAIT	I	PSRAM wait input signal to the FMC
NBL[3:0]	O	Byte lane output. Byte 0 to Byte 3 control (Upper and lower byte enable)

The maximum capacity is 512 Mbits.

PSRAM, 16-bit multiplexed I/Os

Table 53. 16-Bit multiplexed I/O PSRAM

FMC signal name	I/O	Function
CLK	O	Clock (for synchronous access)
A[25:16]	O	Address bus
AD[15:0]	I/O	16-bit multiplexed, bidirectional address/data bus (the 16-bit address A[15:0] and data D[15:0] are multiplexed on the databus)
NE[x]	O	Chip select, x = 1..4 (called NCE by PSRAM (CellularRAM™ i.e. CRAM))
NOE	O	Output enable
NWE	O	Write enable
NL(= NADV)	O	Address valid PSRAM input (memory signal name: NADV)
NWAIT	I	PSRAM wait input signal to the FMC
NBL[1:0]	O	Byte lane output. Byte 0 and Byte 1 control (upper and lower byte enable)

The maximum capacity is 512 Mbits (26 address lines).

12.5.2 Supported memories and transactions

Table 54 below shows an example of the supported devices, access modes and transactions when the memory data bus is 16-bit wide for NOR Flash memory, PSRAM and SRAM. The transactions not allowed (or not supported) by the FMC are shown in gray in this example.

Table 54. NOR Flash/PSRAM: example of supported memories and transactions

Device	Mode	R/W	AHB data size	Memory data size	Allowed/not allowed	Comments
NOR Flash (muxed I/Os and nonmuxed I/Os)	Asynchronous	R	8	16	Y	-
	Asynchronous	W	8	16	N	-
	Asynchronous	R	16	16	Y	-
	Asynchronous	W	16	16	Y	-
	Asynchronous	R	32	16	Y	Split into 2 FMC accesses
	Asynchronous	W	32	16	Y	Split into 2 FMC accesses
	Asynchronous page	R	-	16	N	Mode is not supported
	Synchronous	R	8	16	N	-
	Synchronous	R	16	16	Y	-
	Synchronous	R	32	16	Y	-
PSRAM (multiplexed I/Os and non- multiplexed I/Os)	Asynchronous	R	8	16	Y	-
	Asynchronous	W	8	16	Y	Use of byte lanes NBL[1:0]
	Asynchronous	R	16	16	Y	-
	Asynchronous	W	16	16	Y	-
	Asynchronous	R	32	16	Y	Split into 2 FMC accesses
	Asynchronous	W	32	16	Y	Split into 2 FMC accesses
	Asynchronous page	R	-	16	N	Mode is not supported
	Synchronous	R	8	16	N	-
	Synchronous	R	16	16	Y	-
	Synchronous	R	32	16	Y	-
	Synchronous	W	8	16	Y	Use of byte lanes NBL[1:0]
SRAM and ROM	Asynchronous	R	8 / 16	16	Y	-
	Asynchronous	W	8 / 16	16	Y	Use of byte lanes NBL[1:0]
	Asynchronous	R	32	16	Y	Split into 2 FMC accesses
	Asynchronous	W	32	16	Y	Split into 2 FMC accesses Use of byte lanes NBL[1:0]

12.5.3 General timing rules

Signals synchronization

- All controller output signals change on the rising edge of the internal clock (HCLK)
- In Synchronous mode (read or write), all output signals change on the rising edge of HCLK. Whatever the CLKDIV value, all outputs change as follows:
 - NOEL/NWEL/ NEL/NADVL/ NADVH /NBLL/ Address valid outputs change on the falling edge of FMC_CLK clock.
 - NOEH/ NWEH / NEH/ NOEH/NBLH/ Address invalid outputs change on the rising edge of FMC_CLK clock.

12.5.4 NOR Flash/PSRAM controller asynchronous transactions

Asynchronous static memories (NOR Flash, PSRAM, SRAM)

- Signals are synchronized by the internal clock HCLK. This clock is not issued to the memory
- The FMC always samples the data before de-asserting the NOE signal. This guarantees that the memory data hold timing constraint is met (minimum Chip Enable high to data transition is usually 0 ns)
- If the Extended mode is enabled (EXTMOD bit is set in the FMC_BCRx register), up to four extended modes (A, B, C and D) are available. It is possible to mix A, B, C and D modes for read and write operations. For example, read operation can be performed in mode A and write in mode B.
- If the Extended mode is disabled (EXTMOD bit is reset in the FMC_BCRx register), the FMC can operate in Mode1 or Mode2 as follows:
 - Mode 1 is the default mode when SRAM/PSRAM memory type is selected (MTYP = 0x0 or 0x01 in the FMC_BCRx register)
 - Mode 2 is the default mode when NOR memory type is selected (MTYP = 0x10 in the FMC_BCRx register).

Mode 1 - SRAM/PSRAM (CRAM)

The next figures show the read and write transactions for the supported modes followed by the required configuration of FMC_BCRx, and FMC_BTRx/FMC_BWTRx registers.

Figure 35. Mode1 read access waveforms

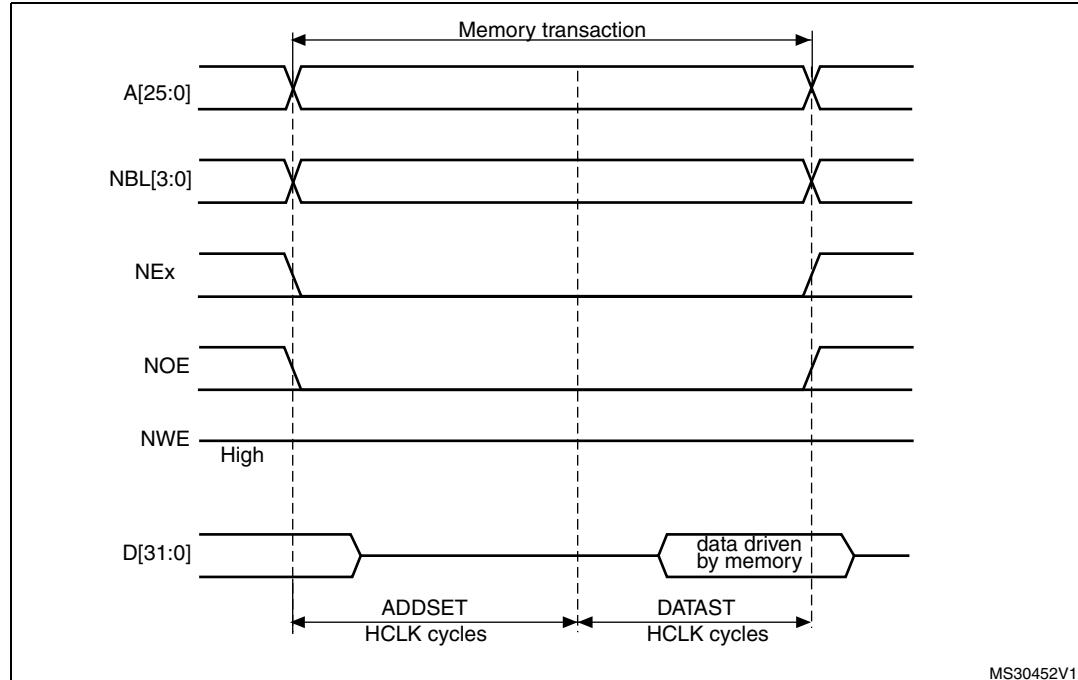
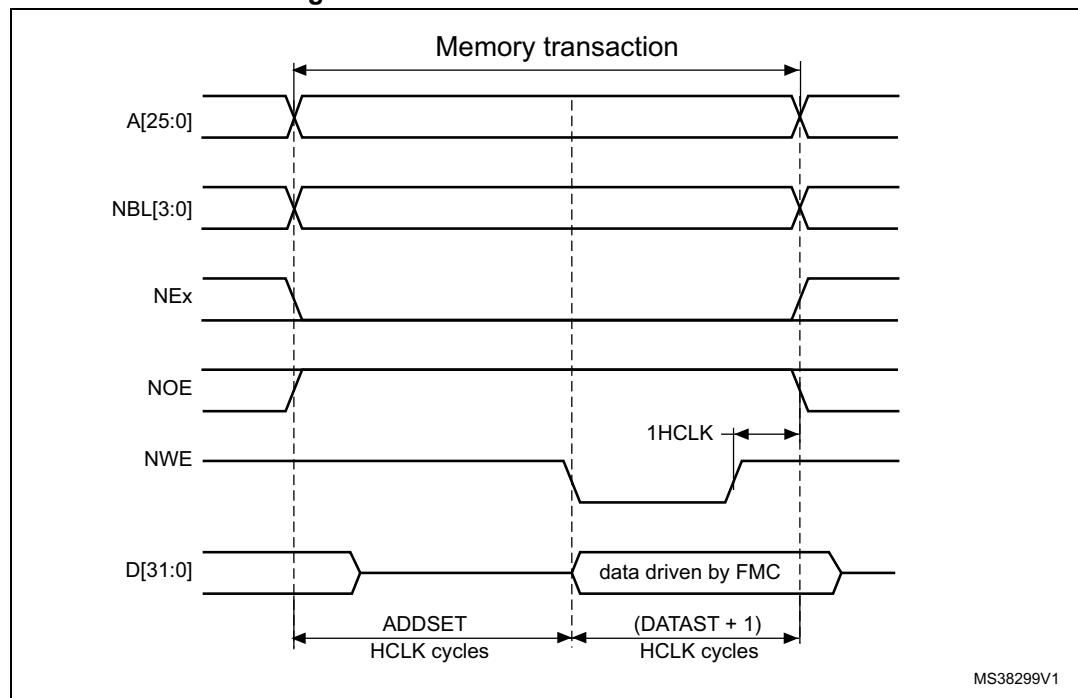


Figure 36. Mode1 write access waveforms



The one HCLK cycle at the end of the write transaction helps guarantee the address and data hold time after the NWE rising edge. Due to the presence of this HCLK cycle, the DATAST value must be greater than zero (DATAST > 0).

Table 55. FMC_BCRx bit fields

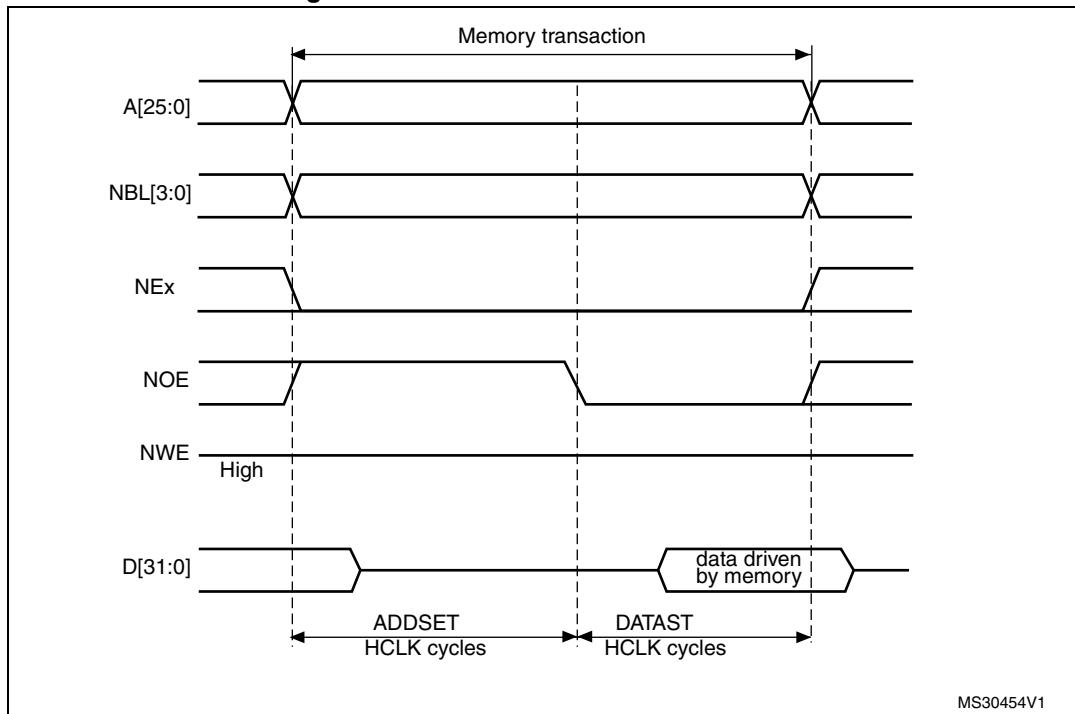
Bit number	Bit name	Value to set
31:22	Reserved	0x000
21	WFDIS	As needed
20	CCLKEN	As needed
19	CBURSTRW	0x0 (no effect in Asynchronous mode)
18:16	CPSIZE	0x0 (no effect in Asynchronous mode)
15	ASYNCWAIT	Set to 1 if the memory supports this feature. Otherwise keep at 0.
14	EXTMOD	0x0
13	WAITEN	0x0 (no effect in Asynchronous mode)
12	WREN	As needed
11	WAITCFG	Don't care
10	Reserved	0x0
9	WAITPOL	Meaningful only if bit 15 is 1
8	BURSTEN	0x0
7	Reserved	0x1
6	FACCEN	Don't care
5:4	MWID	As needed
3:2	MTYP	As needed, exclude 0x2 (NOR Flash memory)
1	MUXE	0x0
0	MBKEN	0x1

Table 56. FMC_BTRx bit fields

Bit number	Bit name	Value to set
31:30	Reserved	0x0
29:28	ACCMOD	Don't care
27:24	DATLAT	Don't care
23:20	CLKDIV	Don't care
19:16	BUSTURN	Time between NEx high to NEx low (BUSTURN HCLK).
15:8	DATAST	Duration of the second access phase (DATAST+1 HCLK cycles for write accesses, DATAST HCLK cycles for read accesses).
7:4	ADDHLD	Don't care
3:0	ADDSET	Duration of the first access phase (ADDSET HCLK cycles). Minimum value for ADDSET is 0.

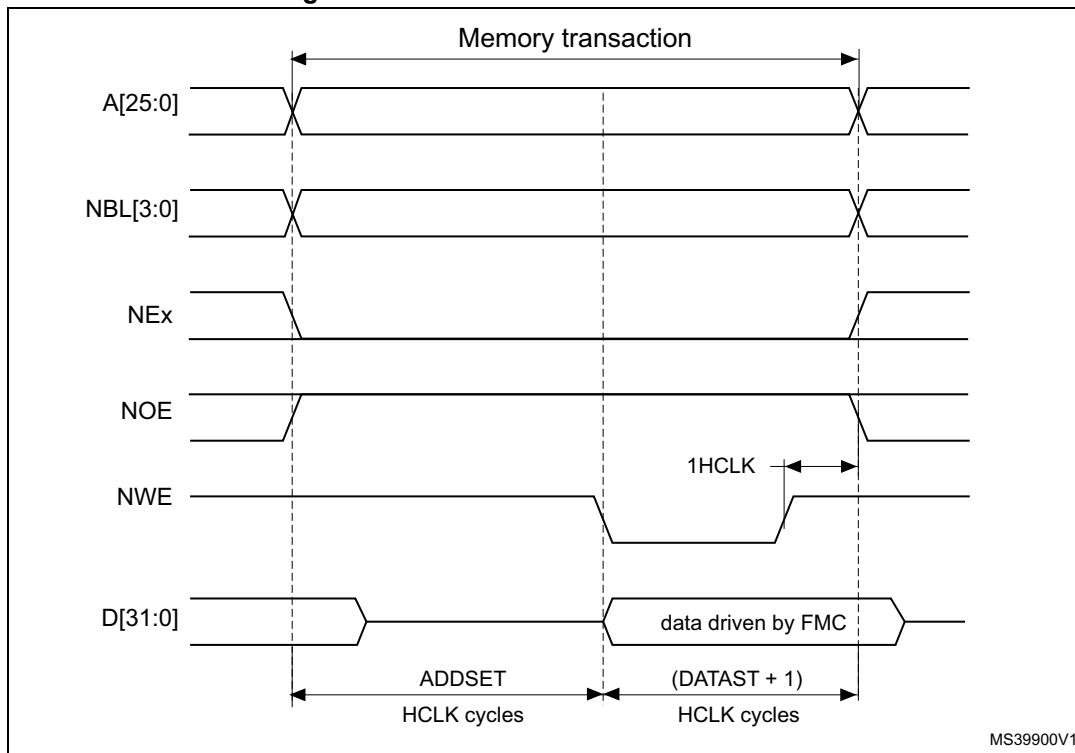
Mode A - SRAM/PSRAM (CRAM) OE toggling

Figure 37. ModeA read access waveforms



1. NBL[3:0] are driven low during the read access

Figure 38. ModeA write access waveforms



The differences compared with Mode1 are the toggling of NOE and the independent read and write timings.

Table 57. FMC_BCRx bit fields

Bit number	Bit name	Value to set
31:22	Reserved	0x000
21	WFDIS	As needed
20	CCLKEN	As needed
19	CBURSTRW	0x0 (no effect in Asynchronous mode)
18:16	CPSIZE	0x0 (no effect in Asynchronous mode)
15	ASYNCWAIT	Set to 1 if the memory supports this feature. Otherwise keep at 0.
14	EXTMOD	0x1
13	WAITEN	0x0 (no effect in Asynchronous mode)
12	WREN	As needed
11	WAITCFG	Don't care
10	Reserved	0x0
9	WAITPOL	Meaningful only if bit 15 is 1
8	BURSTEN	0x0
7	Reserved	0x1
6	FACCEN	Don't care
5:4	MWID	As needed
3:2	MTYP	As needed, exclude 0x2 (NOR Flash memory)
1	MUXEN	0x0
0	MBKEN	0x1

Table 58. FMC_BTRx bit fields

Bit number	Bit name	Value to set
31:30	Reserved	0x0
29:28	ACCMOD	0x0
27:24	DATLAT	Don't care
23:20	CLKDIV	Don't care
19:16	BUSTURN	Time between NEx high to NEx low (BUSTURN HCLK).
15:8	DATAST	Duration of the second access phase (DATAST HCLK cycles) for read accesses.
7:4	ADDHLD	Don't care
3:0	ADDSET	Duration of the first access phase (ADDSET HCLK cycles) for read accesses. Minimum value for ADDSET is 0.

Table 59. FMC_BWTRx bit fields

Bit number	Bit name	Value to set
31:30	Reserved	0x0
29:28	ACCMOD	0x0
27:24	DATLAT	Don't care
23:20	CLKDIV	Don't care
19:16	BUSTURN	Time between NEx high to NEx low (BUSTURN HCLK).
15:8	DATAST	Duration of the second access phase (DATAST HCLK cycles) for write accesses.
7:4	ADDHLD	Don't care
3:0	ADDSET	Duration of the first access phase (ADDSET HCLK cycles) for write accesses. Minimum value for ADDSET is 0.

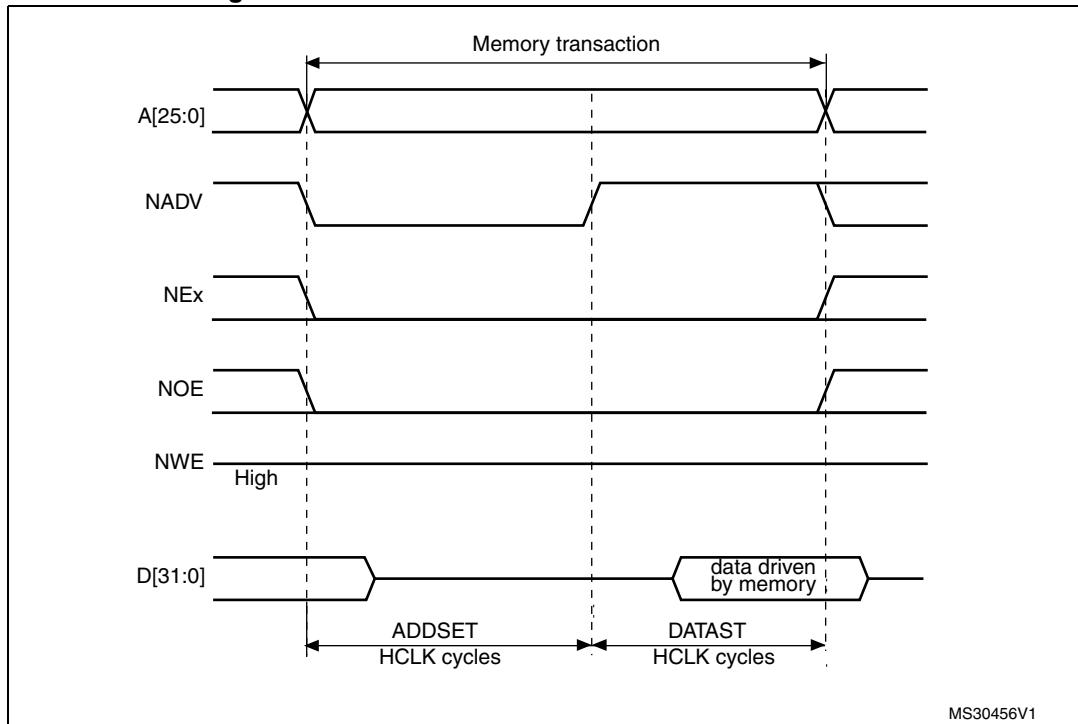
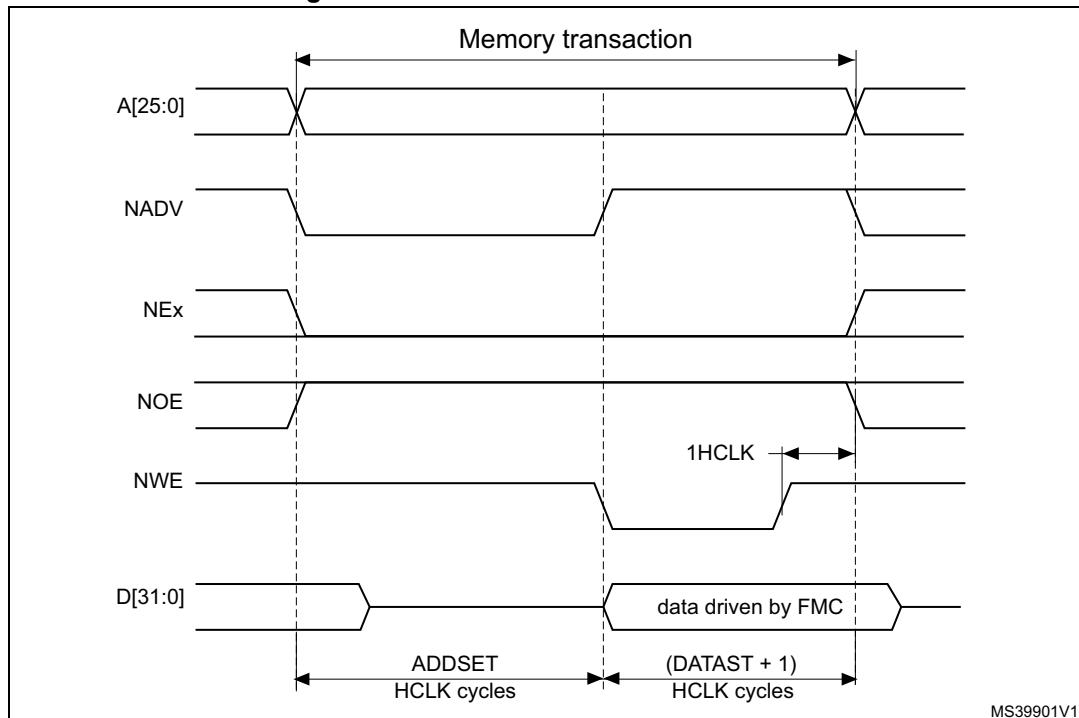
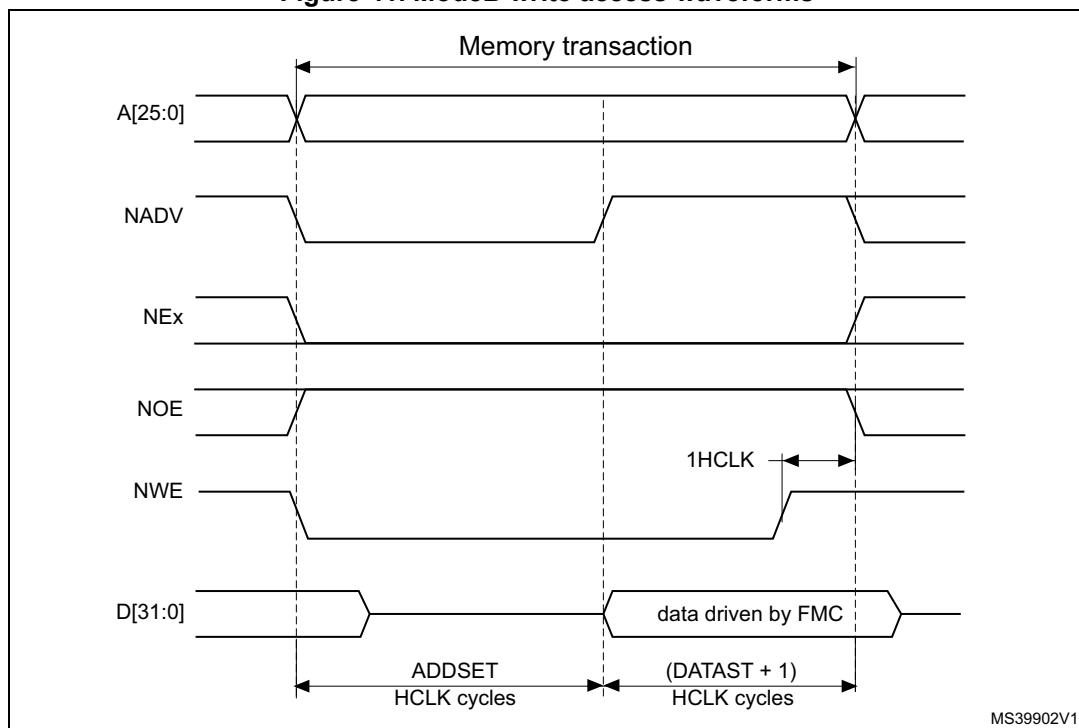
Mode 2/B - NOR Flash**Figure 39. Mode2 and mode B read access waveforms**

Figure 40. Mode2 write access waveforms**Figure 41. ModeB write access waveforms**

The differences with Mode1 are the toggling of NWE and the independent read and write timings when extended mode is set (Mode B).

Table 60. FMC_BCRx bit fields

Bit number	Bit name	Value to set
31:22	Reserved	0x000
21	WFDIS	As needed
20	CCLKEN	As needed
19	CBURSTRW	0x0 (no effect in Asynchronous mode)
18:16	CPSIZE	0x0 (no effect in Asynchronous mode)
15	ASYNCPWAIT	Set to 1 if the memory supports this feature. Otherwise keep at 0.
14	EXTMOD	0x1 for mode B, 0x0 for mode 2
13	WAITEN	0x0 (no effect in Asynchronous mode)
12	WREN	As needed
11	WAITCFG	Don't care
10	Reserved	0x0
9	WAITPOL	Meaningful only if bit 15 is 1
8	BURSTEN	0x0
7	Reserved	0x1
6	FACCEN	0x1
5:4	MWID	As needed
3:2	MTYP	0x2 (NOR Flash memory)
1	MUXEN	0x0
0	MBKEN	0x1

Table 61. FMC_BTRx bit fields

Bit number	Bit name	Value to set
31:30	Reserved	0x0
29:28	ACCMOD	0x1 if Extended mode is set
27:24	DATLAT	Don't care
23:20	CLKDIV	Don't care
19:16	BUSTURN	Time between NEx high to NEx low (BUSTURN HCLK).
15:8	DATAST	Duration of the access second phase (DATAST HCLK cycles) for read accesses.
7:4	ADDHLD	Don't care
3:0	ADDSET	Duration of the access first phase (ADDSET HCLK cycles) for read accesses. Minimum value for ADDSET is 0.

Table 62. FMC_BWTRx bit fields

Bit number	Bit name	Value to set
31:30	Reserved	0x0
29:28	ACCMOD	0x1 if Extended mode is set
27:24	DATLAT	Don't care
23:20	CLKDIV	Don't care
19:16	BUSTURN	Time between NEx high to NEx low (BUSTURN HCLK).
15:8	DATAST	Duration of the access second phase (DATAST HCLK cycles) for write accesses.
7:4	ADDHLD	Don't care
3:0	ADDSET	Duration of the access first phase (ADDSET HCLK cycles) for write accesses. Minimum value for ADDSET is 0.

Note: The FMC_BWTRx register is valid only if the Extended mode is set (mode B), otherwise its content is don't care.

Mode C - NOR Flash - OE toggling

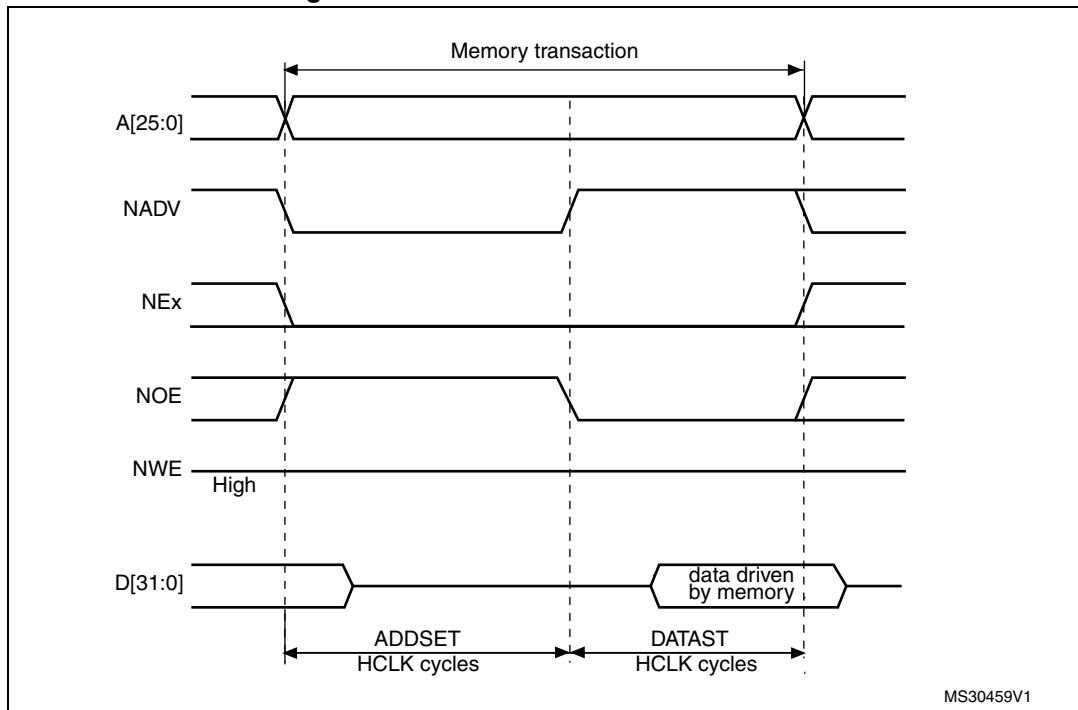
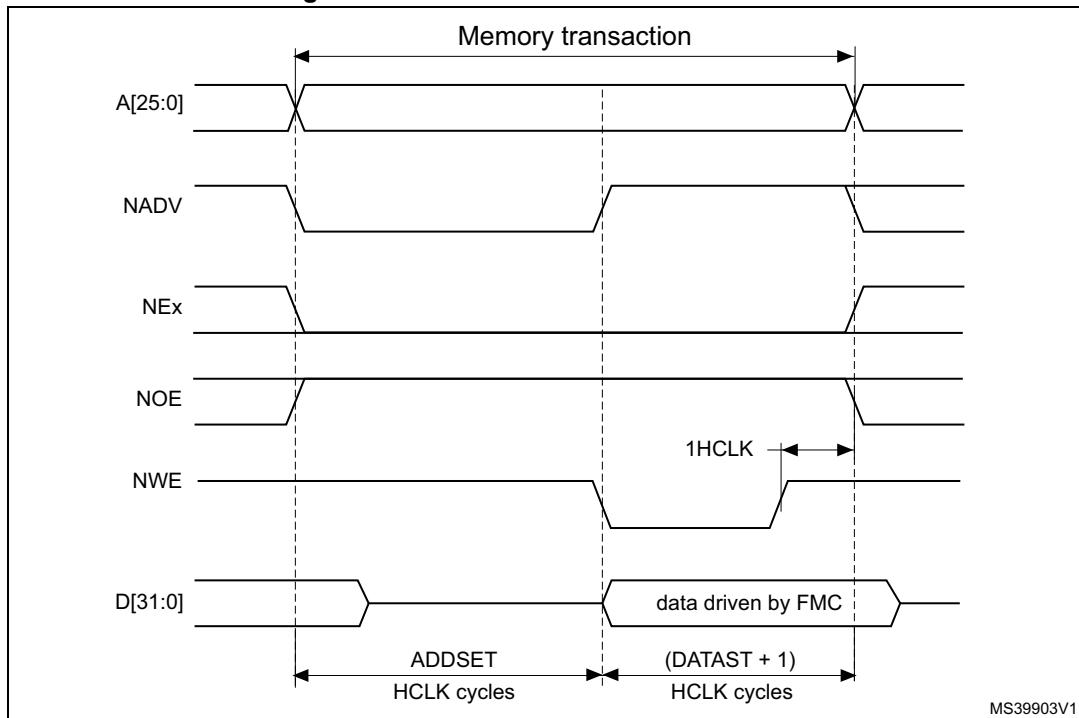
Figure 42. ModeC read access waveforms

Figure 43. ModeC write access waveforms

The differences compared with Mode1 are the toggling of NOE and the independent read and write timings.

Table 63. FMC_BCRx bit fields

Bit number	Bit name	Value to set
31:22	Reserved	0x000
21	WFDIS	As needed
20	CCLKEN	As needed
19	CBURSTRW	0x0 (no effect in Asynchronous mode)
18:16	CPSIZE	0x0 (no effect in Asynchronous mode)
15	ASYNCFWAIT	Set to 1 if the memory supports this feature. Otherwise keep at 0.
14	EXTMOD	0x1
13	WAITEN	0x0 (no effect in Asynchronous mode)
12	WREN	As needed
11	WAITCFG	Don't care
10	Reserved	0x0
9	WAITPOL	Meaningful only if bit 15 is 1
8	BURSTEN	0x0
7	Reserved	0x1
6	FACCEN	0x1
5:4	MWID	As needed

Table 63. FMC_BCRx bit fields (continued)

Bit number	Bit name	Value to set
3:2	MTYP	0x02 (NOR Flash memory)
1	MUXEN	0x0
0	MBKEN	0x1

Table 64. FMC_BTRx bit fields

Bit number	Bit name	Value to set
31:30	Reserved	0x0
29:28	ACCMOD	0x2
27:24	DATLAT	0x0
23:20	CLKDIV	0x0
19:16	BUSTURN	Time between NEx high to NEx low (BUSTURN HCLK).
15:8	DATAST	Duration of the second access phase (DATAST HCLK cycles) for read accesses.
7:4	ADDHLD	Don't care
3:0	ADDSET	Duration of the first access phase (ADDSET HCLK cycles) for read accesses. Minimum value for ADDSET is 0.

Table 65. FMC_BWTRx bit fields

Bit number	Bit name	Value to set
31:30	Reserved	0x0
29:28	ACCMOD	0x2
27:24	DATLAT	Don't care
23:20	CLKDIV	Don't care
19:16	BUSTURN	Time between NEx high to NEx low (BUSTURN HCLK).
15:8	DATAST	Duration of the second access phase (DATAST HCLK cycles) for write accesses.
7:4	ADDHLD	Don't care
3:0	ADDSET	Duration of the first access phase (ADDSET HCLK cycles) for write accesses. Minimum value for ADDSET is 0.

Mode D - asynchronous access with extended address

Figure 44. ModeD read access waveforms

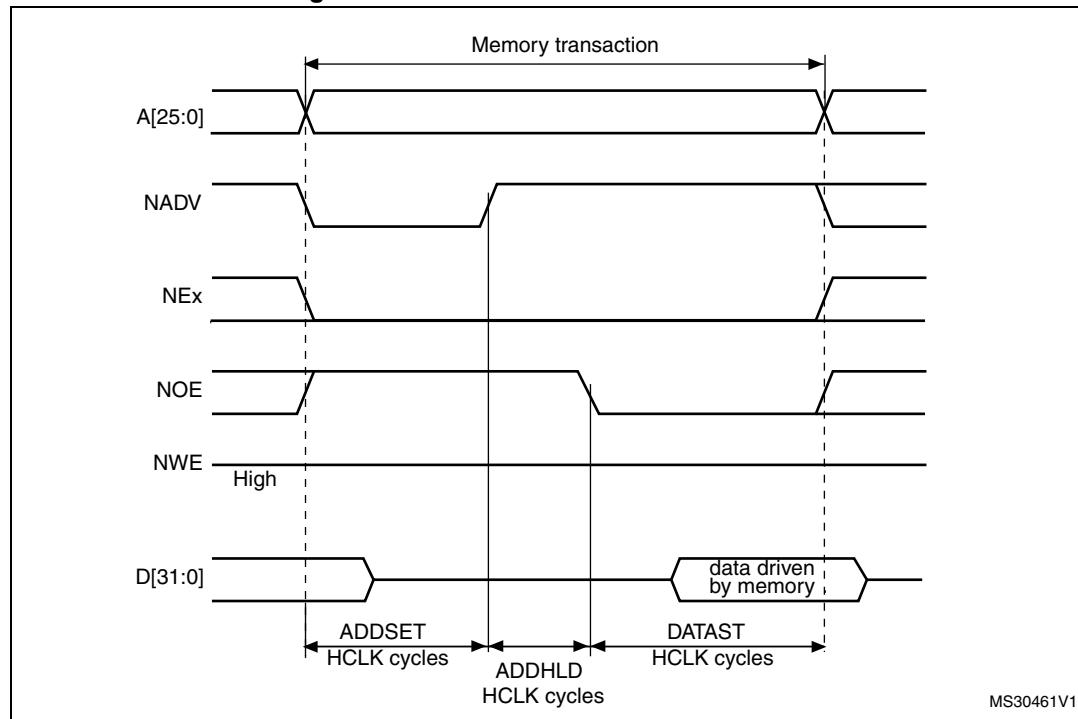
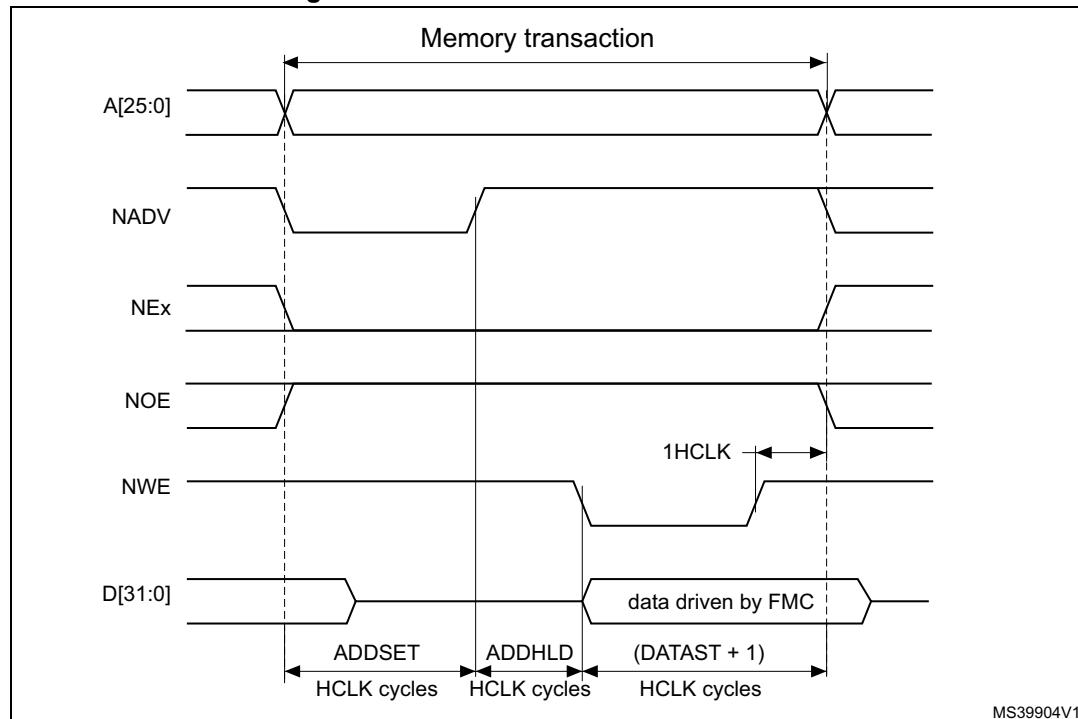


Figure 45. ModeD write access waveforms



The differences with Mode1 are the toggling of NOE that goes on toggling after NADV changes and the independent read and write timings.

Table 66. FMC_BCRx bit fields

Bit number	Bit name	Value to set
31:22	Reserved	0x000
21	WFDIS	As needed
20	CCLKEN	As needed
19	CBURSTRW	0x0 (no effect in Asynchronous mode)
18:16	CPSIZE	0x0 (no effect in Asynchronous mode)
15	ASYNCPWAIT	Set to 1 if the memory supports this feature. Otherwise keep at 0.
14	EXTMOD	0x1
13	WAITEN	0x0 (no effect in Asynchronous mode)
12	WREN	As needed
11	WAITCFG	Don't care
10	Reserved	0x0
9	WAITPOL	Meaningful only if bit 15 is 1
8	BURSTEN	0x0
7	Reserved	0x1
6	FACCEN	Set according to memory support
5:4	MWID	As needed
3:2	MTYP	As needed
1	MUXEN	0x0
0	MBKEN	0x1

Table 67. FMC_BTRx bit fields

Bit number	Bit name	Value to set
31:30	Reserved	0x0
29:28	ACCMOD	0x3
27:24	DATLAT	Don't care
23:20	CLKDIV	Don't care
19:16	BUSTURN	Time between NEx high to NEx low (BUSTURN HCLK).
15:8	DATAST	Duration of the second access phase (DATAST HCLK cycles) for read accesses.
7:4	ADDHLD	Duration of the middle phase of the read access (ADDHLD HCLK cycles)
3:0	ADDSET	Duration of the first access phase (ADDSET HCLK cycles) for read accesses. Minimum value for ADDSET is 1.

Table 68. FMC_BWTRx bit fields

Bit number	Bit name	Value to set
31:30	Reserved	0x0
29:28	ACCMOD	0x3
27:24	DATLAT	Don't care
23:20	CLKDIV	Don't care
19:16	BUSTURN	Time between NEx high to NEx low (BUSTURN HCLK).
15:8	DATAST	Duration of the second access phase (DATAST + 1 HCLK cycles) for write accesses.
7:4	ADDHLD	Duration of the middle phase of the write access (ADDHLD HCLK cycles)
3:0	ADDSET	Duration of the first access phase (ADDSET HCLK cycles) for write accesses. Minimum value for ADDSET is 1.

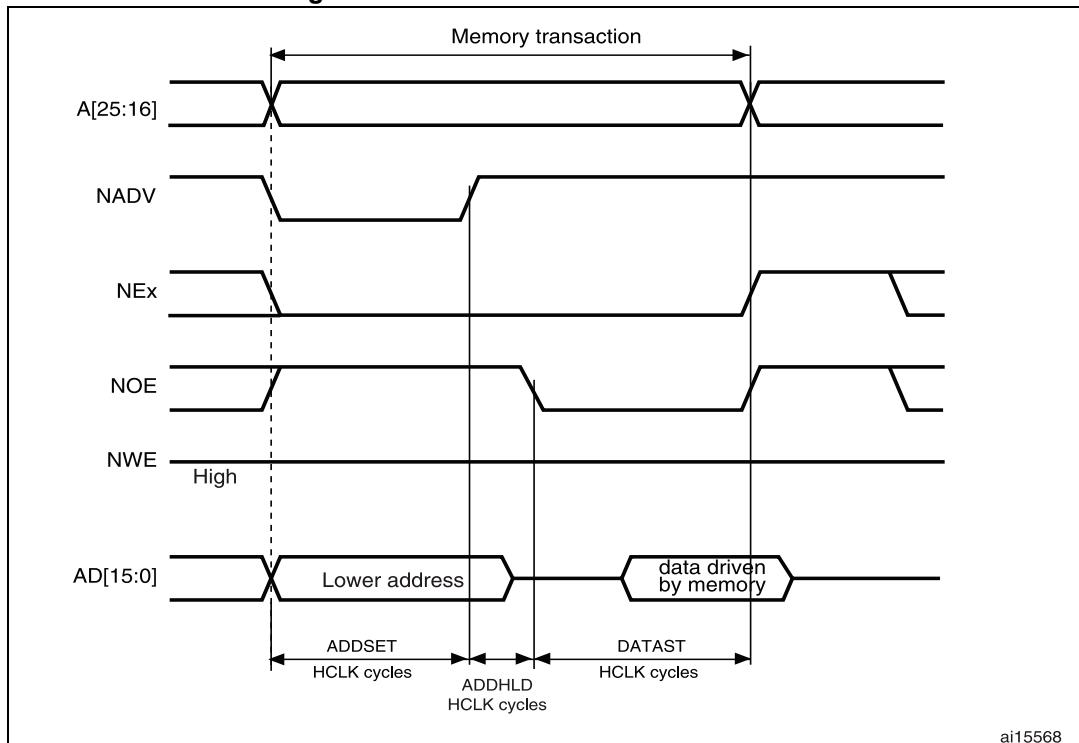
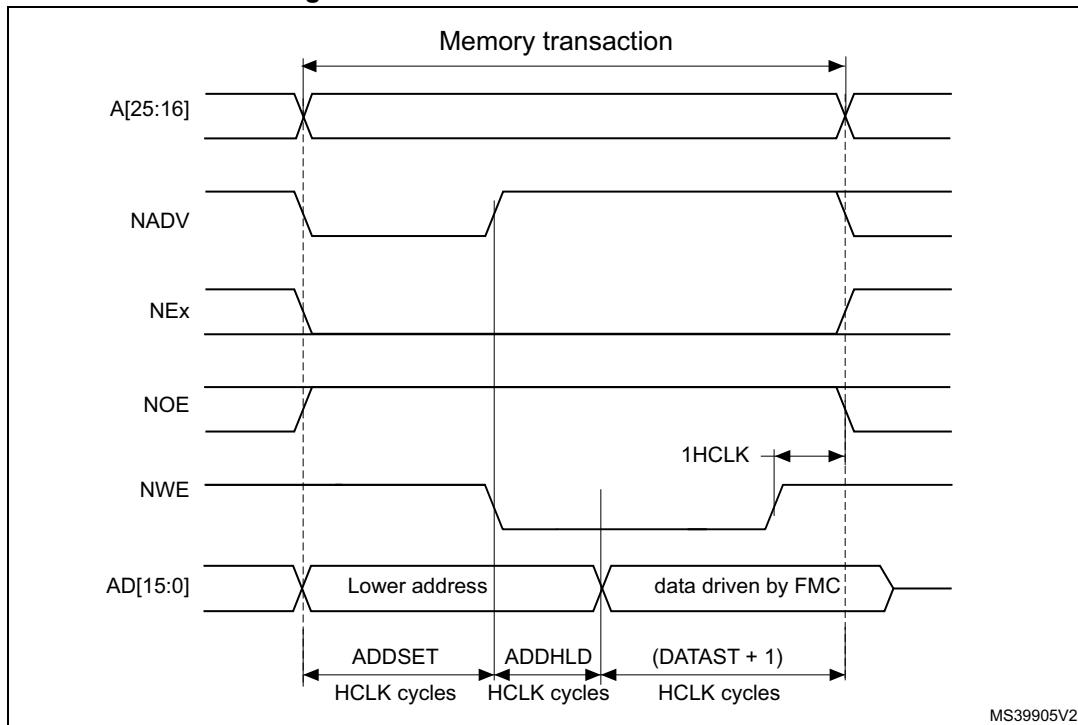
Muxed mode - multiplexed asynchronous access to NOR Flash memory**Figure 46. Muxed read access waveforms**

Figure 47. Muxed write access waveforms

The difference with ModeD is the drive of the lower address byte(s) on the data bus.

Table 69. FMC_BCRx bit fields

Bit number	Bit name	Value to set
31:22	Reserved	0x000
21	WFDIS	As needed
20	CCLKEN	As needed
19	CBURSTRW	0x0 (no effect in Asynchronous mode)
18:16	CPSIZE	0x0 (no effect in Asynchronous mode)
15	ASYNCWAIT	Set to 1 if the memory supports this feature. Otherwise keep at 0.
14	EXTMOD	0x0
13	WAITEN	0x0 (no effect in Asynchronous mode)
12	WREN	As needed
11	WAITCFG	Don't care
10	Reserved	0x0
9	WAITPOL	Meaningful only if bit 15 is 1
8	BURSTEN	0x0
7	Reserved	0x1
6	FACCEN	0x1
5:4	MWID	As needed

Table 69. FMC_BCRx bit fields (continued)

Bit number	Bit name	Value to set
3:2	MTYP	0x2 (NOR Flash memory) or 0x1(PSRAM)
1	MUXEN	0x1
0	MBKEN	0x1

Table 70. FMC_BTRx bit fields

Bit number	Bit name	Value to set
31:30	Reserved	0x0
29:28	ACCMOD	0x0
27:24	DATLAT	Don't care
23:20	CLKDIV	Don't care
19:16	BUSTURN	Time between NEx high to NEx low (BUSTURN HCLK).
15:8	DATAST	Duration of the second access phase (DATAST HCLK cycles for read accesses and DATAST+1 HCLK cycles for write accesses).
7:4	ADDHLD	Duration of the middle phase of the access (ADDHLD HCLK cycles).
3:0	ADDSET	Duration of the first access phase (ADDSET HCLK cycles). Minimum value for ADDSET is 1.

WAIT management in asynchronous accesses

If the asynchronous memory asserts the WAIT signal to indicate that it is not yet ready to accept or to provide data, the ASYNCWAIT bit has to be set in FMC_BCRx register.

If the WAIT signal is active (high or low depending on the WAITPOL bit), the second access phase (Data setup phase), programmed by the DATAST bits, is extended until WAIT becomes inactive. Unlike the data setup phase, the first access phases (Address setup and Address hold phases), programmed by the ADDSET and ADDHLD bits, are not WAIT sensitive and so they are not prolonged.

The data setup phase must be programmed so that WAIT can be detected 4 HCLK cycles before the end of the memory transaction. The following cases must be considered:

1. The memory asserts the WAIT signal aligned to NOE/NWE which toggles:

$$\text{DATAST} \geq (4 \times \text{HCLK}) + \text{max_wait_assertion_time}$$

2. The memory asserts the WAIT signal aligned to NEx (or NOE/NWE not toggling):
if

$$\text{max_wait_assertion_time} > \text{address_phase} + \text{hold_phase}$$

then:

$$\text{DATAST} \geq (4 \times \text{HCLK}) + (\text{max_wait_assertion_time} - \text{address_phase} - \text{hold_phase})$$

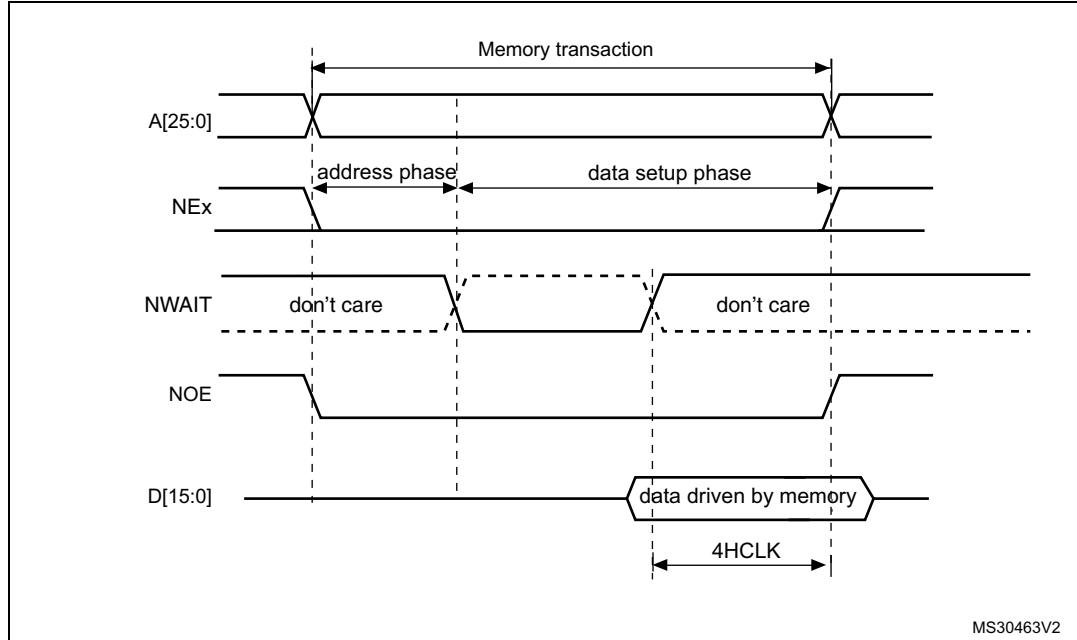
otherwise

$$\text{DATAST} \geq 4 \times \text{HCLK}$$

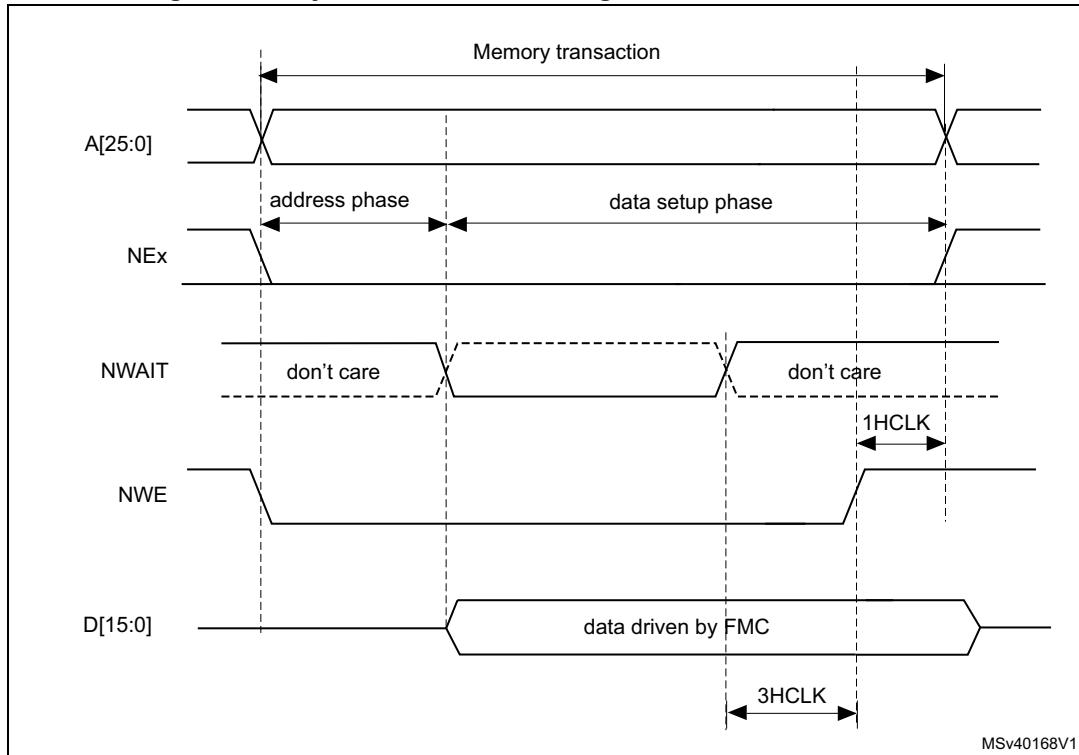
where max_wait_assertion_time is the maximum time taken by the memory to assert the WAIT signal once NEx/NOE/NWE is low.

Figure 48 and *Figure 49* show the number of HCLK clock cycles that are added to the memory access phase after WAIT is released by the asynchronous memory (independently of the above cases).

Figure 48. Asynchronous wait during a read access waveforms



1. NWAIT polarity depends on WAITPOL bit setting in FMC_BCRx register.

Figure 49. Asynchronous wait during a write access waveforms

1. NWAIT polarity depends on WAITPOL bit setting in FMC_BCRx register.

12.5.5 Synchronous transactions

The memory clock, FMC_CLK, is a submultiple of HCLK. It depends on the value of CLKDIV and the MWID/ AHB data size, following the formula given below:

$$\text{FMC_CLK divider ratio} = \max(\text{CLKDIV} + 1, \text{MWID(AHB data size)})$$

If MWID is 16 or 8-bit, the FMC_CLK divider ratio is always defined by the programmed CLKDIV value.

If MWID is 32-bit, the FMC_CLK divider ratio depends also on AHB data size.

Example:

- If CLKDIV=1, MWID = 32 bits, AHB data size=8 bits, FMC_CLK=HCLK/4.
- If CLKDIV=1, MWID = 16 bits, AHB data size=8 bits, FMC_CLK=HCLK/2.

NOR Flash memories specify a minimum time from NADV assertion to CLK high. To meet this constraint, the FMC does not issue the clock to the memory during the first internal clock cycle of the synchronous access (before NADV assertion). This guarantees that the rising edge of the memory clock occurs in the middle of the NADV low pulse.

Data latency versus NOR memory latency

The data latency is the number of cycles to wait before sampling the data. The DATLAT value must be consistent with the latency value specified in the NOR Flash configuration register. The FMC does not include the clock cycle when NADV is low in the data latency count.

Caution: Some NOR Flash memories include the NADV Low cycle in the data latency count, so that the exact relation between the NOR Flash latency and the FMC DATLAT parameter can be either:

- NOR Flash latency = (DATLAT + 2) CLK clock cycles
- or NOR Flash latency = (DATLAT + 3) CLK clock cycles

Some recent memories assert NWAIT during the latency phase. In such cases DATLAT can be set to its minimum value. As a result, the FMC samples the data and waits long enough to evaluate if the data are valid. Thus the FMC detects when the memory exits latency and real data are processed.

Other memories do not assert NWAIT during latency. In this case the latency must be set correctly for both the FMC and the memory, otherwise invalid data are mistaken for good data, or valid data are lost in the initial phase of the memory access.

Single-burst transfer

When the selected bank is configured in Burst mode for synchronous accesses, if for example an AHB single-burst transaction is requested on 16-bit memories, the FMC performs a burst transaction of length 1 (if the AHB transfer is 16 bits), or length 2 (if the AHB transfer is 32 bits) and de-assert the chip select signal when the last data is strobed.

Such transfers are not the most efficient in terms of cycles compared to asynchronous read operations. Nevertheless, a random asynchronous access would first require to re-program the memory access mode, which would altogether last longer.

Cross boundary page for CellularRAM™ 1.5

CellularRAM™ 1.5 does not allow burst access to cross the page boundary. The FMC controller allows to split automatically the burst access when the memory page size is reached by configuring the CPSIZE bits in the FMC_BCR1 register following the memory page size.

Wait management

For synchronous NOR Flash memories, NWAIT is evaluated after the programmed latency period, which corresponds to (DATLAT+2) CLK clock cycles.

If NWAIT is active (low level when WAITPOL = 0, high level when WAITPOL = 1), wait states are inserted until NWAIT is inactive (high level when WAITPOL = 0, low level when WAITPOL = 1).

When NWAIT is inactive, the data is considered valid either immediately (bit WAITCFG = 1) or on the next clock edge (bit WAITCFG = 0).

During wait-state insertion via the NWAIT signal, the controller continues to send clock pulses to the memory, keeping the chip select and output enable signals valid. It does not consider the data as valid.

In Burst mode, there are two timing configurations for the NOR Flash NWAIT signal:

- The Flash memory asserts the NWAIT signal one data cycle before the wait state (default after reset).
- The Flash memory asserts the NWAIT signal during the wait state

The FMC supports both NOR Flash wait state configurations, for each chip select, thanks to the WAITCFG bit in the FMC_BCRx registers (x = 0..3).

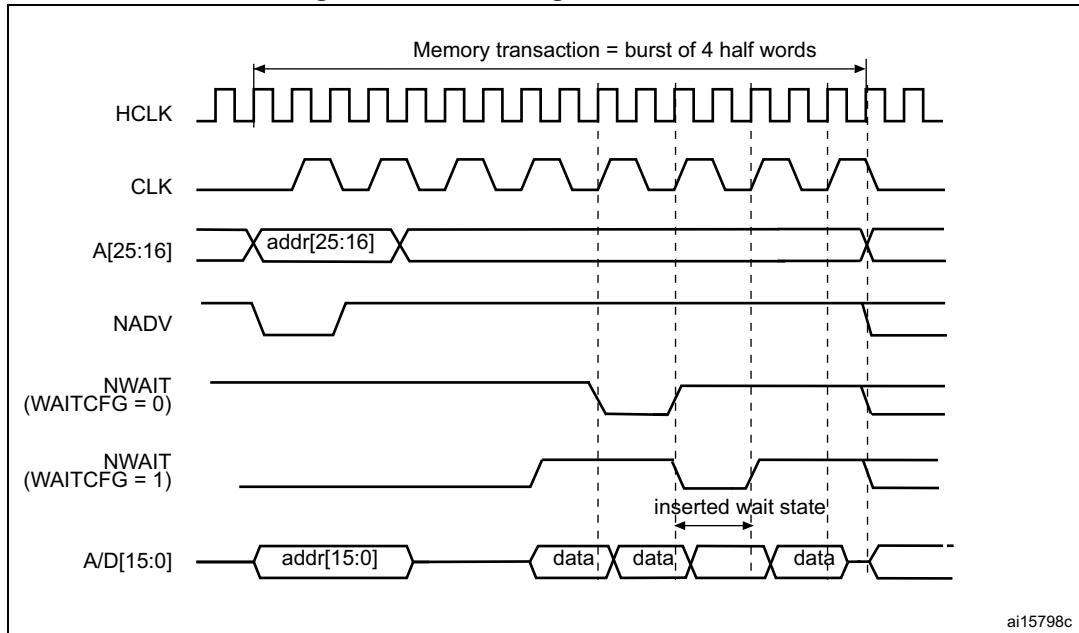
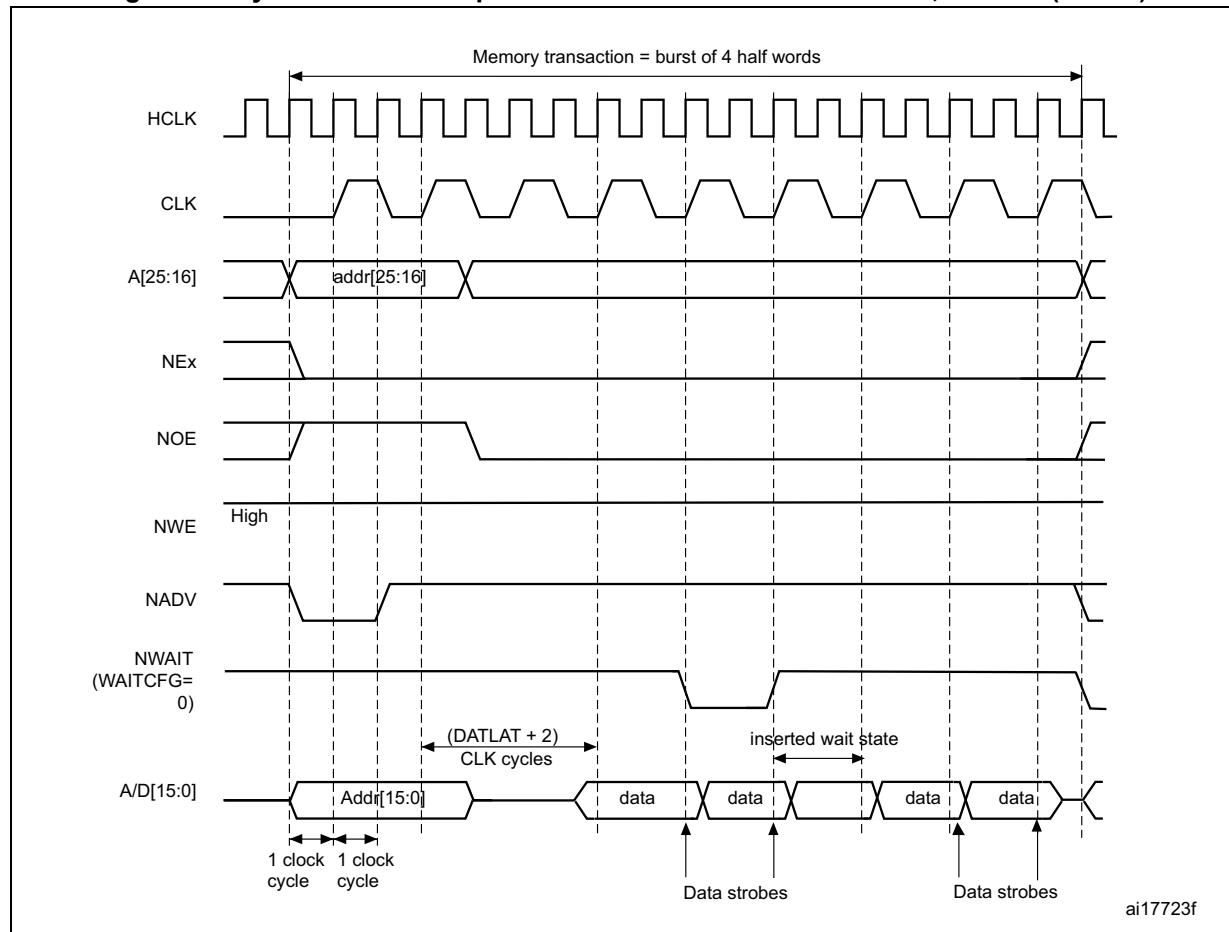
Figure 50. Wait configuration waveforms

Figure 51. Synchronous multiplexed read mode waveforms - NOR, PSRAM (CRAM)



1. Byte lane outputs (NBL are not shown; for NOR access, they are held high, and, for PSRAM (CRAM) access, they are held low.

Table 71. FMC_BCRx bit fields

Bit number	Bit name	Value to set
31:22	Reserved	0x000
21	WFDIS	As needed
20	CCLKEN	As needed
19	CBURSTRW	No effect on synchronous read
18:16	CPSIZE	0x0 (no effect in Asynchronous mode)
15	ASYNCWAIT	0x0
14	EXTMOD	0x0
13	WAITEN	To be set to 1 if the memory supports this feature, to be kept at 0 otherwise
12	WREN	No effect on synchronous read
11	WAITCFG	To be set according to memory
10	Reserved	0x0

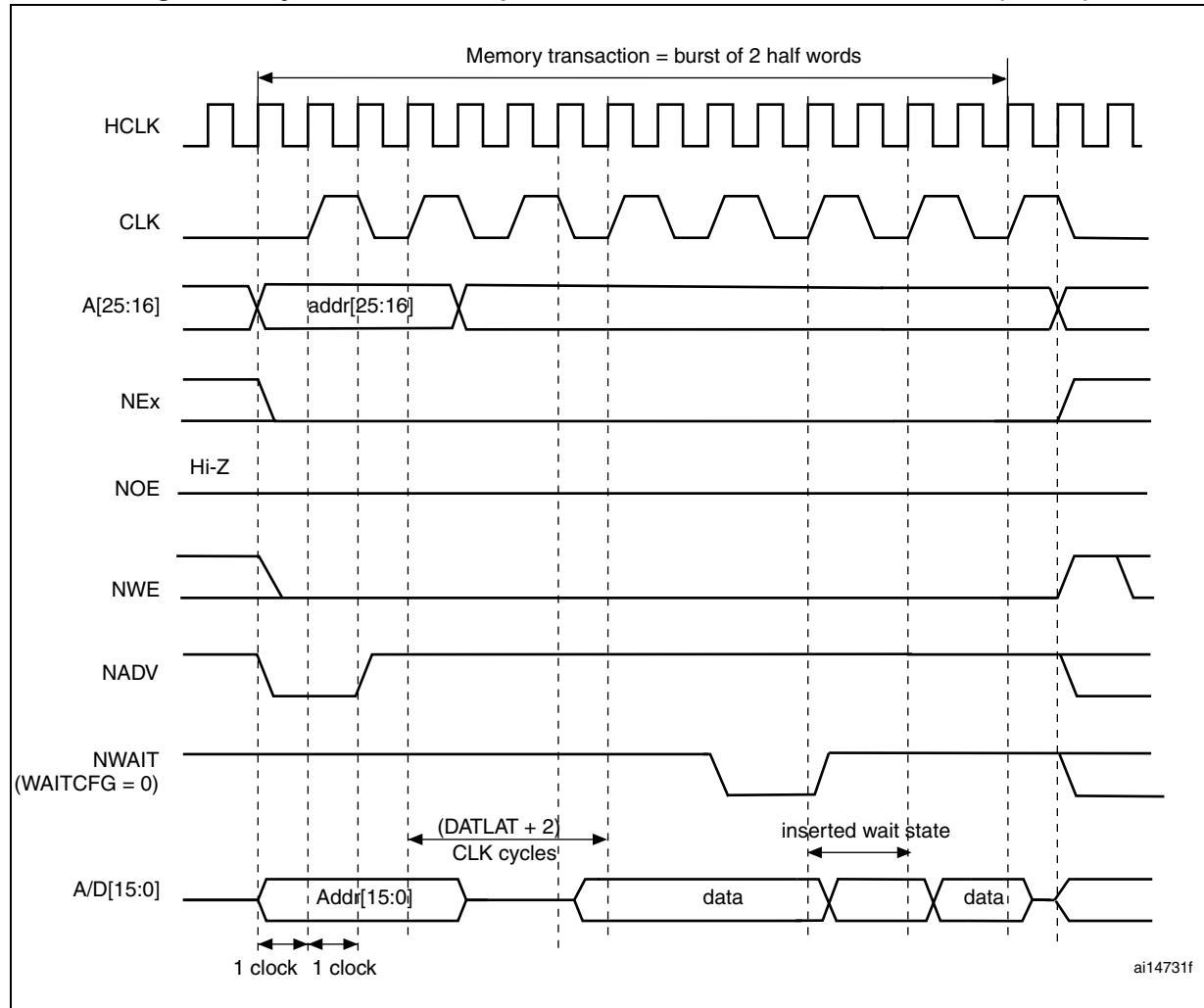
Table 71. FMC_BCRx bit fields (continued)

Bit number	Bit name	Value to set
9	WAITPOL	To be set according to memory
8	BURSTEN	0x1
7	Reserved	0x1
6	FACCEN	Set according to memory support (NOR Flash memory)
5-4	MWID	As needed
3-2	MTYP	0x1 or 0x2
1	MUXEN	As needed
0	MBKEN	0x1

Table 72. FMC_BTRx bit fields

Bit number	Bit name	Value to set
31:30	Reserved	0x0
29:28	ACCMOD	0x0
27-24	DATLAT	Data latency
27-24	DATLAT	Data latency
23-20	CLKDIV	0x0 to get CLK = HCLK (Not supported) 0x1 to get CLK = 2 × HCLK ..
19-16	BUSTURN	Time between NEx high to NEx low (BUSTURN HCLK).
15-8	DATAST	Don't care
7-4	ADDHLD	Don't care
3-0	ADDSET	Don't care

Figure 52. Synchronous multiplexed write mode waveforms - PSRAM (CRAM)



1. The memory must issue NWAIT signal one cycle in advance, accordingly WAITCFG must be programmed to 0.
2. Byte Lane (NBL) outputs are not shown, they are held low while NEx is active.

Table 73. FMC_BCRx bit fields

Bit number	Bit name	Value to set
31:22	Reserved	0x000
21	WFDIS	As needed
20	CCLKEN	As needed
19	CBURSTRW	0x1
18:16	CPSIZE	As needed (0x1 for CRAM 1.5)
15	ASYNCWAIT	0x0
14	EXTMOD	0x0
13	WAITEN	To be set to 1 if the memory supports this feature, to be kept at 0 otherwise.

Table 73. FMC_BCRx bit fields (continued)

Bit number	Bit name	Value to set
12	WREN	0x1
11	WAITCFG	0x0
10	Reserved	0x0
9	WAITPOL	to be set according to memory
8	BURSTEN	no effect on synchronous write
7	Reserved	0x1
6	FACCEN	Set according to memory support
5-4	MWID	As needed
3-2	MTYP	0x1
1	MUXEN	As needed
0	MBKEN	0x1

Table 74. FMC_BTRx bit fields

Bit number	Bit name	Value to set
31-30	Reserved	0x0
29:28	ACCMOD	0x0
27-24	DATLAT	Data latency
23-20	CLKDIV	0x0 to get CLK = HCLK (not supported) 0x1 to get CLK = 2 × HCLK
19-16	BUSTURN	Time between NEx high to NEx low (BUSTURN HCLK).
15-8	DATAST	Don't care
7-4	ADDHLD	Don't care
3-0	ADDSET	Don't care

12.5.6 NOR/PSRAM controller registers

SRAM/NOR-Flash chip-select control register for bank x (FMC_BCRx) (x = 1 to 4)

Address offset: $8 * (x - 1)$, ($x = 1$ to 4)

Reset value: Bank 1: 0x0000 30DB

Reset value: Bank 2: 0x0000 30D2

Reset value: Bank 3: 0x0000 30D2

Reset value: Bank 4: 0x0000 30D2

This register contains the control information of each memory bank, used for SRAMs, PSRAM and NOR Flash memories.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	WFDIS	CCLK EN	CBURST RW	CPSIZE[2:0]		
										rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ASYNC WAIT	EXT MOD	WAIT EN	WREN	WAIT CFG	Res.	WAIT POL	BURST EN	Res.	FACC EN	MWID[1:0]		MTYP[1:0]		MUX EN	MBK EN
rw	rw	rw	rw	rw		rw	rw		rw	rw	rw	rw	rw	rw	rw

Bits 31:22 Reserved, must be kept at reset value.

Bit 21 **WFDIS**: Write FIFO Disable

This bit disables the Write FIFO used by the FMC controller.

0: Write FIFO enabled (Default after reset)

1: Write FIFO disabled

Note: The WFDIS bit of the FMC_BCR2..4 registers is don't care. It is only enabled through the FMC_BCR1 register.

Bit 20 **CCLKEN**: Continuous Clock Enable.

This bit enables the FMC_CLK clock output to external memory devices.

0: The FMC_CLK is only generated during the synchronous memory access (read/write transaction). The FMC_CLK clock ratio is specified by the programmed CLKDIV value in the FMC_BCRx register (default after reset).

1: The FMC_CLK is generated continuously during asynchronous and synchronous access. The FMC_CLK clock is activated when the CCLKEN is set.

Note: The CCLKEN bit of the FMC_BCR2..4 registers is don't care. It is only enabled through the FMC_BCR1 register. Bank 1 must be configured in Synchronous mode to generate the FMC_CLK continuous clock.

Note: If CCLKEN bit is set, the FMC_CLK clock ratio is specified by CLKDIV value in the FMC_BTR1 register. CLKDIV in FMC_BWTR1 is don't care.

Note: If the Synchronous mode is used and CCLKEN bit is set, the synchronous memories connected to other banks than Bank 1 are clocked by the same clock (the CLKDIV value in the FMC_BTR2..4 and FMC_BWTR2..4 registers for other banks has no effect.)

Bit 19 **CBURSTRW**: Write burst enable.

For PSRAM (CRAM) operating in Burst mode, the bit enables synchronous accesses during write operations. The enable bit for synchronous read accesses is the BURSTEN bit in the FMC_BCRx register.

0: Write operations are always performed in Asynchronous mode

1: Write operations are performed in Synchronous mode.

Bits 18:16 **CPSIZE[2:0]**: CRAM page size.

These are used for CellularRAM™ 1.5 which does not allow burst access to cross the address boundaries between pages. When these bits are configured, the FMC controller splits automatically the burst access when the memory page size is reached (refer to memory datasheet for page size).

000: No burst split when crossing page boundary (default after reset)

001: 128 bytes

010: 256 bytes

011: 512 bytes

100: 1024 bytes

Others: reserved

Bit 15 **ASYNCWAIT**: Wait signal during asynchronous transfers

This bit enables/disables the FMC to use the wait signal even during an asynchronous protocol.

0: NWAIT signal is not taken in to account when running an asynchronous protocol (default after reset)

1: NWAIT signal is taken in to account when running an asynchronous protocol

Bit 14 **EXTMOD**: Extended mode enable.

This bit enables the FMC to program the write timings for non multiplexed asynchronous accesses inside the FMC_BWTR register, thus resulting in different timings for read and write operations.

0: values inside FMC_BWTR register are not taken into account (default after reset)

1: values inside FMC_BWTR register are taken into account

Note: When the Extended mode is disabled, the FMC can operate in Mode1 or Mode2 as follows:

- *Mode 1 is the default mode when the SRAM/PSRAM memory type is selected (MTYP =0x0 or 0x01)*
- *Mode 2 is the default mode when the NOR memory type is selected (MTYP = 0x10).*

Bit 13 **WAITEN**: Wait enable bit.

This bit enables/disables wait-state insertion via the NWAIT signal when accessing the memory in Synchronous mode.

0: NWAIT signal is disabled (its level not taken into account, no wait state inserted after the programmed Flash latency period)

1: NWAIT signal is enabled (its level is taken into account after the programmed latency period to insert wait states if asserted) (default after reset)

Bit 12 **WREN**: Write enable bit.

This bit indicates whether write operations are enabled/disabled in the bank by the FMC:

0: Write operations are disabled in the bank by the FMC, an AHB error is reported,

1: Write operations are enabled for the bank by the FMC (default after reset).

Bit 11 **WAITCFG**: Wait timing configuration.

The NWAIT signal indicates whether the data from the memory are valid or if a wait state must be inserted when accessing the memory in Synchronous mode. This configuration bit determines if NWAIT is asserted by the memory one clock cycle before the wait state or during the wait state:

0: NWAIT signal is active one data cycle before wait state (default after reset),

1: NWAIT signal is active during wait state (not used for PSRAM).

Bit 10 Reserved, must be kept at reset value.

Bit 9 **WAITPOL:** Wait signal polarity bit.

Defines the polarity of the wait signal from memory used for either in Synchronous or Asynchronous mode:

- 0: NWAIT active low (default after reset),
- 1: NWAIT active high.

Bit 8 **BURSTEN:** Burst enable bit.

This bit enables/disables synchronous accesses during read operations. It is valid only for synchronous memories operating in Burst mode:

- 0: Burst mode disabled (default after reset). Read accesses are performed in Asynchronous mode.
- 1: Burst mode enable. Read accesses are performed in Synchronous mode.

Bit 7 Reserved, must be kept at reset value.

Bit 6 **FACCEN:** Flash access enable

Enables NOR Flash memory access operations.

- 0: Corresponding NOR Flash memory access is disabled
- 1: Corresponding NOR Flash memory access is enabled (default after reset)

Bits 5:4 **MWID[1:0]:** Memory data bus width.

Defines the external memory device width, valid for all type of memories.

- 00: 8 bits
- 01: 16 bits (default after reset)
- 10: 32 bits
- 11: reserved

Bits 3:2 **MTYP[1:0]:** Memory type.

Defines the type of external memory attached to the corresponding memory bank:

- 00: SRAM (default after reset for Bank 2...4)
- 01: PSRAM (CRAM)
- 10: NOR Flash/OneNAND Flash (default after reset for Bank 1)
- 11: reserved

Bit 1 **MUXEN:** Address/data multiplexing enable bit.

When this bit is set, the address and data values are multiplexed on the data bus, valid only with NOR and PSRAM memories:

- 0: Address/Data non multiplexed
- 1: Address/Data multiplexed on databus (default after reset)

Bit 0 **MBKEN:** Memory bank enable bit.

Enables the memory bank. After reset Bank1 is enabled, all others are disabled. Accessing a disabled bank causes an ERROR on AHB bus.

- 0: Corresponding memory bank is disabled
- 1: Corresponding memory bank is enabled

SRAM/NOR-Flash chip-select timing register for bank x (FMC_BTRx)

Address offset: $0x04 + 8 * (x - 1)$, ($x = 1$ to 4)

Reset value: 0xFFFF FFFF

This register contains the control information of each memory bank, used for SRAMs, PSRAM and NOR Flash memories. If the EXTMOD bit is set in the FMC_BCRx register, then this register is partitioned for write and read access, that is, 2 registers are available: one to configure read accesses (this register) and one to configure write accesses (FMC_BWTRx registers).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	Res	ACCMOD[1:0]		DATLAT[3:0]				CLKDIV[3:0]				BUSTURN[3:0]			
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATAST[7:0]								ADDHLD[3:0]				ADDSET[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30 Reserved, must be kept at reset value.

Bits 29:28 **ACCMOD[1:0]: Access mode**

Specifies the asynchronous access modes as shown in the timing diagrams. These bits are taken into account only when the EXTMOD bit in the FMC_BCRx register is 1.

- 00: Access mode A
- 01: Access mode B
- 10: Access mode C
- 11: Access mode D

Bits 27:24 **DATLAT[3:0]: (see note below bit descriptions): Data latency for synchronous memory**

For synchronous access with read/write Burst mode enabled (BURSTEN / CBURSTRW bits set), defines the number of memory clock cycles (+2) to issue to the memory before reading/writing the first data:

This timing parameter is not expressed in HCLK periods, but in FMC_CLK periods.

For asynchronous access, this value is don't care.

0000: Data latency of 2 CLK clock cycles for first burst access

1111: Data latency of 17 CLK clock cycles for first burst access (default value after reset)

Bits 23:20 **CLKDIV[3:0]: Clock divide ratio (for FMC_CLK signal)**

Defines the period of FMC_CLK clock output signal, expressed in number of HCLK cycles:

0000: Reserved

0001: FMC_CLK period = 2 × HCLK periods

0010: FMC_CLK period = 3 × HCLK periods

1111: FMC_CLK period = 16 × HCLK periods (default value after reset)

In asynchronous NOR Flash, SRAM or PSRAM accesses, this value is don't care.

Note: Refer to [Section 12.5.5: Synchronous transactions](#) for FMC_CLK divider ratio formula)

Bits 19:16 BUSTURN[3:0]: Bus turnaround phase duration

These bits are written by software to add a delay at the end of a write-to-read (and read-to-write) transaction. This delay allows to match the minimum time between consecutive transactions (tEHEL from NEx high to NEx low) and the maximum time needed by the memory to free the data bus after a read access (tEHQZ). The programmed bus turnaround delay is inserted between an asynchronous read (muxed or mode D) or write transaction and any other asynchronous /synchronous read or write to or from a static bank. The bank can be the same or different in case of read, in case of write the bank can be different except for muxed or mode D.

In some cases, whatever the programmed BUSTURN values, the bus turnaround delay is fixed

as follows:

- The bus turnaround delay is not inserted between two consecutive asynchronous write transfers to the same static memory bank except for muxed and D modes.
- There is a bus turnaround delay of 1 HCLK clock cycle between:
 - Two consecutive asynchronous read transfers to the same static memory bank except for muxed and D modes.
 - An asynchronous read to an asynchronous or synchronous write to any static bank or dynamic bank except for muxed and D modes.
 - An asynchronous (modes 1, 2, A, B or C) read and a read from another static bank.
- There is a bus turnaround delay of 2 HCLK clock cycle between:
 - Two consecutive synchronous writes (burst or single) to the same bank.
 - A synchronous write (burst or single) access and an asynchronous write or read transfer to or from static memory bank (the bank can be the same or different for the case of read).
 - Two consecutive synchronous reads (burst or single) followed by any synchronous/asynchronous read or write from/to another static memory bank.
- There is a bus turnaround delay of 3 HCLK clock cycle between:
 - Two consecutive synchronous writes (burst or single) to different static bank.
 - A synchronous write (burst or single) access and a synchronous read from the same or a different bank.

0000: BUSTURN phase duration = 0 HCLK clock cycle added

...

1111: BUSTURN phase duration = 15 x HCLK clock cycles added (default value after reset)

Bits 15:8 DATAST[7:0]: Data-phase duration

These bits are written by software to define the duration of the data phase (refer to [Figure 35](#) to [Figure 47](#)), used in asynchronous accesses:

0000 0000: Reserved

0000 0001: DATAST phase duration = 1 × HCLK clock cycles

0000 0010: DATAST phase duration = 2 × HCLK clock cycles

...

1111 1111: DATAST phase duration = 255 × HCLK clock cycles (default value after reset)

For each memory type and access mode data-phase duration, refer to the respective figure ([Figure 35](#) to [Figure 47](#)).

Example: Mode1, write access, DATAST=1: Data-phase duration= DATAST+1 = 2 HCLK clock cycles.

Note: In synchronous accesses, this value is don't care.

Bits 7:4 ADDHLD[3:0]: Address-hold phase duration

These bits are written by software to define the duration of the *address hold* phase (refer to [Figure 35](#) to [Figure 47](#)), used in mode D or multiplexed accesses:

0000: Reserved

0001: ADDHLD phase duration = $1 \times$ HCLK clock cycle

0010: ADDHLD phase duration = $2 \times$ HCLK clock cycle

...

1111: ADDHLD phase duration = $15 \times$ HCLK clock cycles (default value after reset)

For each access mode address-hold phase duration, refer to the respective figure ([Figure 35](#) to [Figure 47](#)).

Note: In synchronous accesses, this value is not used, the address hold phase is always 1 memory clock period duration.

Bits 3:0 ADDSET[3:0]: Address setup phase duration

These bits are written by software to define the duration of the *address setup* phase (refer to [Figure 35](#) to [Figure 47](#)), used in SRAMs, ROMs, asynchronous NOR Flash and PSRAM:

0000: ADDSET phase duration = $0 \times$ HCLK clock cycle

...

1111: ADDSET phase duration = $15 \times$ HCLK clock cycles (default value after reset)

For each access mode address setup phase duration, refer to the respective figure ([Figure 35](#) to [Figure 47](#)).

Note: In synchronous accesses, this value is don't care.

In Muxed mode or Mode D, the minimum value for ADDSET is 1.

In mode 1 and PSRAM memory, the minimum value for ADDSET is 1.

Note:

PSRAMs (CRAMs) have a variable latency due to internal refresh. Therefore these memories issue the NWAIT signal during the whole latency phase to prolong the latency as needed.

With PSRAMs (CRAMs) the filled DATLAT must be set to 0, so that the FMC exits its latency phase soon and starts sampling NWAIT from memory, then starts to read or write when the memory is ready.

This method can be used also with the latest generation of synchronous Flash memories that issue the NWAIT signal, unlike older Flash memories (check the datasheet of the specific Flash memory being used).

SRAM/NOR-Flash write timing registers 1..4 (FMC_BWTR1..4)

Address offset: $0x104 + 8 * (x - 1)$, $x = 1\dots4$

Reset value: 0xFFFF FFFF

This register contains the control information of each memory bank. It is used for SRAMs, PSRAMs and NOR Flash memories. When the EXTMOD bit is set in the FMC_BCRx register, then this register is active for write access.

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

Bits 31:30 Reserved, must be kept at reset value.

Bits 29:28 **ACCMOD[1:0]**: Access mode.

Specifies the asynchronous access modes as shown in the next timing diagrams. These bits are taken into account only when the EXTMOD bit in the FMC_BCRx register is 1.

- 00: Access mode A
- 01: Access mode B
- 10: Access mode C
- 11: Access mode D

Bits 27:20 Reserved, must be kept at reset value.

Bits 19:16 **BUSTURN[3:0]**: Bus turnaround phase duration

The programmed bus turnaround delay is inserted between an asynchronous write transfer and any other asynchronous /synchronous read or write transfer to or from a static bank. The bank can be the same or different in case of read, in case of write the bank can be different expect for muxed or mode D.

In some cases, whatever the programmed BUSTURN values, the bus turnaround delay is fixed as follows:

- The bus turnaround delay is not inserted between two consecutive asynchronous write transfers to the same static memory bank except for muxed and D modes.
- There is a bus turnaround delay of 2 HCLK clock cycle between:
 - Two consecutive synchronous writes (burst or single) to the same bank.
 - A synchronous write (burst or single) transfer and an asynchronous write or read transfer to or from static memory bank.
- There is a bus turnaround delay of 3 HCLK clock cycle between:
 - Two consecutive synchronous writes (burst or single) to different static bank.
 - A synchronous write (burst or single) transfer and a synchronous read from the same or a different bank.

0000: BUSTURN phase duration = 0 HCLK clock cycle added

...

1111: BUSTURN phase duration = 15 HCLK clock cycles added (default value after reset)

Bits 15:8 **DATAST[7:0]**: Data-phase duration.

These bits are written by software to define the duration of the data phase (refer to [Figure 35](#) to [Figure 47](#)), used in asynchronous SRAM, PSRAM and NOR Flash memory accesses:

0000 0000: Reserved

0000 0001: DATAST phase duration = 1 × HCLK clock cycles

0000 0010: DATAST phase duration = 2 × HCLK clock cycles

...

1111 1111: DATAST phase duration = 255 × HCLK clock cycles (default value after reset)

Bits 7:4 **ADDHLD[3:0]**: Address-hold phase duration.

These bits are written by software to define the duration of the *address hold* phase (refer to [Figure 44](#) to [Figure 47](#)), used in asynchronous multiplexed accesses:

0000: Reserved

0001: ADDHLD phase duration = 1 × HCLK clock cycle

0010: ADDHLD phase duration = 2 × HCLK clock cycle

...

1111: ADDHLD phase duration = 15 × HCLK clock cycles (default value after reset)

Note: In synchronous NOR Flash accesses, this value is not used, the address hold phase is always 1 Flash clock period duration.

Bits 3:0 **ADDSET[3:0]**: Address setup phase duration.

These bits are written by software to define the duration of the *address setup* phase in HCLK cycles (refer to [Figure 35](#) to [Figure 47](#)), used in asynchronous accesses:

0000: ADDSET phase duration = 0 × HCLK clock cycle

...

1111: ADDSET phase duration = 15 × HCLK clock cycles (default value after reset)

Note: In synchronous accesses, this value is not used, the address setup phase is always 1 Flash clock period duration. In muxed mode, the minimum ADDSET value is 1.

12.6 NAND Flash controller

The FMC generates the appropriate signal timings to drive the following types of device:

- 8- and 16-bit NAND Flash memories

The NAND bank is configured through dedicated registers ([Section 12.6.7](#)). The programmable memory parameters include access timings (shown in [Table 75](#)) and ECC configuration.

Table 75. Programmable NAND Flash access parameters

Parameter	Function	Access mode	Unit	Min.	Max.
Memory setup time	Number of clock cycles (HCLK) required to set up the address before the command assertion	Read/Write	AHB clock cycle (HCLK)	1	255
Memory wait	Minimum duration (in HCLK clock cycles) of the command assertion	Read/Write	AHB clock cycle (HCLK)	2	255
Memory hold	Number of clock cycles (HCLK) during which the address must be held (as well as the data if a write access is performed) after the command de-assertion	Read/Write	AHB clock cycle (HCLK)	1	254
Memory databus high-Z	Number of clock cycles (HCLK) during which the data bus is kept in high-Z state after a write access has started	Write	AHB clock cycle (HCLK)	1	255

12.6.1 External memory interface signals

The following tables list the signals that are typically used to interface NAND Flash memory.

Note: The prefix “N” identifies the signals which are active low.

8-bit NAND Flash memory

Table 76. 8-bit NAND Flash

FMC signal name	I/O	Function
A[17]	O	NAND Flash address latch enable (ALE) signal
A[16]	O	NAND Flash command latch enable (CLE) signal

Table 76. 8-bit NAND Flash (continued)

FMC signal name	I/O	Function
D[7:0]	I/O	8-bit multiplexed, bidirectional address/data bus
NCE	O	Chip select
NOE(= NRE)	O	Output enable (memory signal name: read enable, NRE)
NWE	O	Write enable
NWAIT/INT	I	NAND Flash ready/busy input signal to the FMC

Theoretically, there is no capacity limitation as the FMC can manage as many address cycles as needed.

16-bit NAND Flash memory

Table 77. 16-bit NAND Flash

FMC signal name	I/O	Function
A[17]	O	NAND Flash address latch enable (ALE) signal
A[16]	O	NAND Flash command latch enable (CLE) signal
D[15:0]	I/O	16-bit multiplexed, bidirectional address/data bus
NCE	O	Chip select
NOE(= NRE)	O	Output enable (memory signal name: read enable, NRE)
NWE	O	Write enable
NWAIT/INT	I	NAND Flash ready/busy input signal to the FMC

Theoretically, there is no capacity limitation as the FMC can manage as many address cycles as needed.

12.6.2 NAND Flash supported memories and transactions

Table 78 shows the supported devices, access modes and transactions. Transactions not allowed (or not supported) by the NAND Flash controller are shown in gray.

Table 78. Supported memories and transactions

Device	Mode	R/W	AHB data size	Memory data size	Allowed/not allowed	Comments
NAND 8-bit	Asynchronous	R	8	8	Y	-
	Asynchronous	W	8	8	Y	-
	Asynchronous	R	16	8	Y	Split into 2 FMC accesses
	Asynchronous	W	16	8	Y	Split into 2 FMC accesses
	Asynchronous	R	32	8	Y	Split into 4 FMC accesses
	Asynchronous	W	32	8	Y	Split into 4 FMC accesses
NAND 16-bit	Asynchronous	R	8	16	Y	-
	Asynchronous	W	8	16	N	-
	Asynchronous	R	16	16	Y	-
	Asynchronous	W	16	16	Y	-
	Asynchronous	R	32	16	Y	Split into 2 FMC accesses
	Asynchronous	W	32	16	Y	Split into 2 FMC accesses

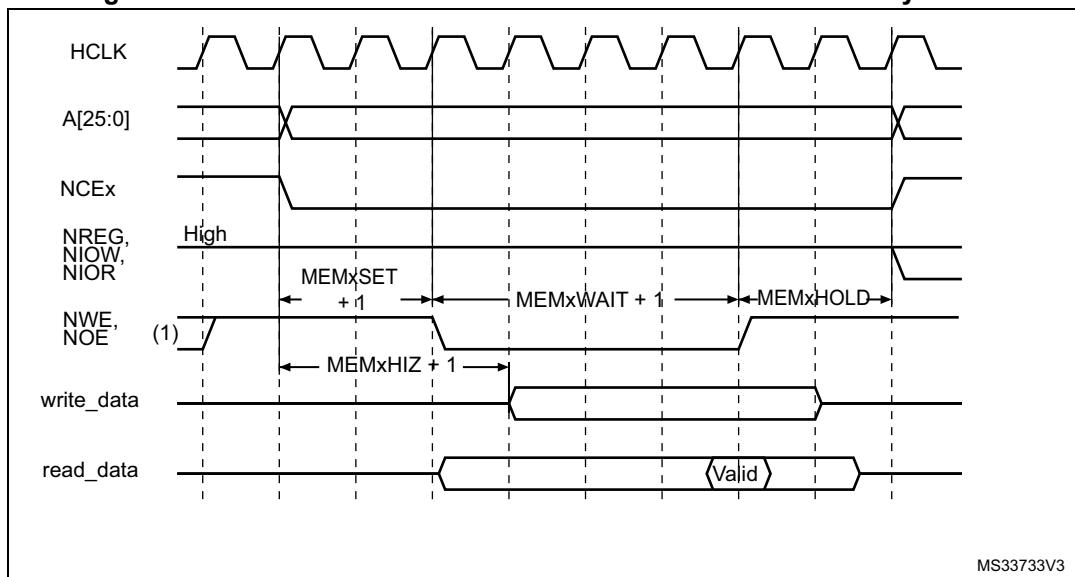
12.6.3 Timing diagrams for NAND Flash memory

The NAND Flash memory bank is managed through a set of registers:

- Control register: FMC_PCR
- Interrupt status register: FMC_SR
- ECC register: FMC_ECCR
- Timing register for Common memory space: FMC_PMEM
- Timing register for Attribute memory space: FMC_PATT

Each timing configuration register contains three parameters used to define number of HCLK cycles for the three phases of any NAND Flash access, plus one parameter that defines the timing for starting driving the data bus when a write access is performed.

Figure 53 shows the timing parameter definitions for common memory accesses, knowing that Attribute memory space access timings are similar.

Figure 53. NAND Flash controller waveforms for common memory access

1. NOE remains high (inactive) during write accesses. NWE remains high (inactive) during read accesses.
2. For write access, the hold phase delay is (MEMHOLD) HCLK cycles and for read access is (MEMHOLD + 2) HCLK cycles.

12.6.4 NAND Flash operations

The command latch enable (CLE) and address latch enable (ALE) signals of the NAND Flash memory device are driven by address signals from the FMC controller. This means that to send a command or an address to the NAND Flash memory, the CPU has to perform a write to a specific address in its memory space.

A typical page read operation from the NAND Flash device requires the following steps:

1. Program and enable the corresponding memory bank by configuring the FMC_PCR and FMC_PMEM (and for some devices, FMC_PATT, see [Section 12.6.5: NAND Flash prewait functionality](#)) registers according to the characteristics of the NAND Flash memory (PWID bits for the data bus width of the NAND Flash, PTYP = 1, PWAITEN = 0 or 1 as needed, see [Section 12.4.2: NAND Flash memory address mapping](#) for timing configuration).
2. The CPU performs a byte write to the common memory space, with data byte equal to one Flash command byte (for example 0x00 for Samsung NAND Flash devices). The LE input of the NAND Flash memory is active during the write strobe (low pulse on NWE), thus the written byte is interpreted as a command by the NAND Flash memory. Once the command is latched by the memory device, it does not need to be written again for the following page read operations.
3. The CPU can send the start address (STARTAD) for a read operation by writing four bytes (or three for smaller capacity devices), STARTAD[7:0], STARTAD[16:9], STARTAD[24:17] and finally STARTAD[25] (for 64 Mb x 8 bit NAND Flash memories) in the common memory or attribute space. The ALE input of the NAND Flash device is active during the write strobe (low pulse on NWE), thus the written bytes are interpreted as the start address for read operations. Using the attribute memory space makes it possible to use a different timing configuration of the FMC, which can be used to implement the prewait functionality needed by some NAND Flash memories (see

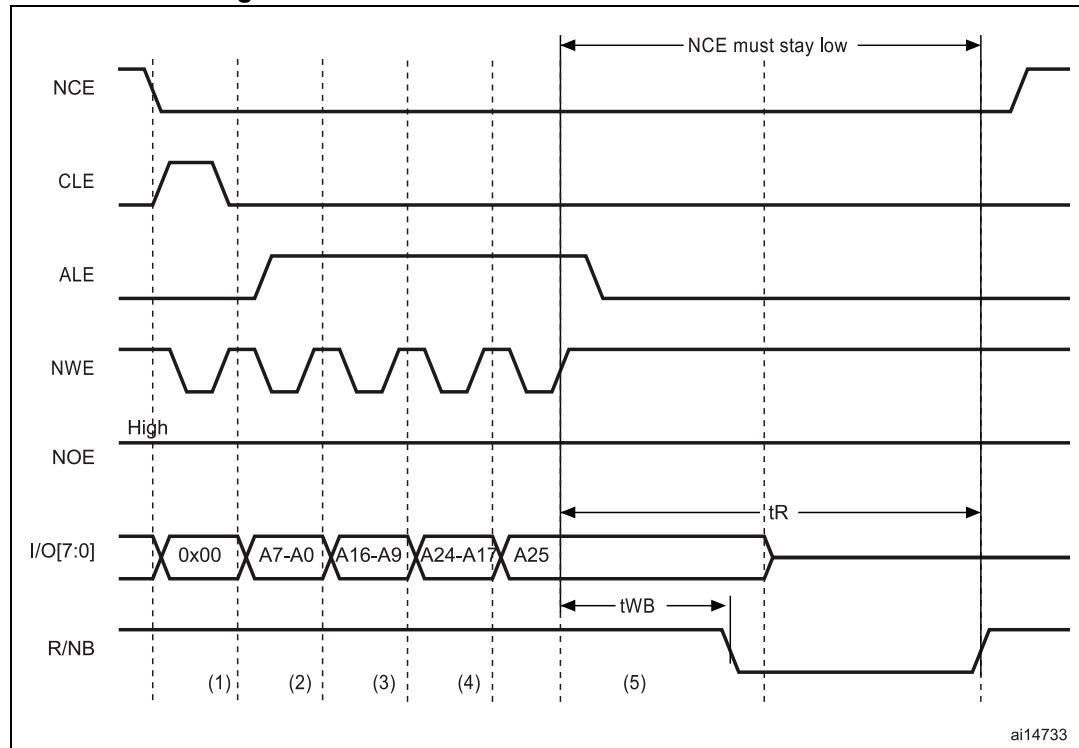
details in [Section 12.6.5: NAND Flash prewait functionality](#)).

4. The controller waits for the NAND Flash memory to be ready (R/NB signal high), before starting a new access to the same or another memory bank. While waiting, the controller holds the NCE signal active (low).
5. The CPU can then perform byte read operations from the common memory space to read the NAND Flash page (data field + Spare field) byte by byte.
6. The next NAND Flash page can be read without any CPU command or address write operation. This can be done in three different ways:
 - by simply performing the operation described in step 5
 - a new random address can be accessed by restarting the operation at step 3
 - a new command can be sent to the NAND Flash device by restarting at step 2

12.6.5 NAND Flash prewait functionality

Some NAND Flash devices require that, after writing the last part of the address, the controller waits for the R/NB signal to go low. (see [Figure 54](#)).

Figure 54. Access to non ‘CE don’t care’ NAND-Flash



1. CPU wrote byte 0x00 at address 0x7001 0000.
2. CPU wrote byte A7~A0 at address 0x7002 0000.
3. CPU wrote byte A16~A9 at address 0x7002 0000.
4. CPU wrote byte A24~A17 at address 0x7002 0000.
5. CPU wrote byte A25 at address 0x7802 0000: FMC performs a write access using FMC_PATT timing definition, where ATTHOLD \geq 7 (providing that $(7+1) \times \text{HCLK} = 112 \text{ ns} > t_{WB} \text{ max}$). This guarantees that NCE remains low until R/NB goes low and high again (only requested for NAND Flash memories where NCE is not don’t care).

When this functionality is required, it can be ensured by programming the MEMHOLD value to meet the t_{WB} timing. However any CPU read access to the NAND Flash memory has a hold delay of (MEMHOLD + 2) HCLK cycles and CPU write access has a hold delay of (MEMHOLD) HCLK cycles inserted between the rising edge of the NWE signal and the next access.

To cope with this timing constraint, the attribute memory space can be used by programming its timing register with an ATTHOLD value that meets the t_{WB} timing, and by keeping the MEMHOLD value at its minimum value. The CPU must then use the common memory space for all NAND Flash read and write accesses, except when writing the last address byte to the NAND Flash device, where the CPU must write to the attribute memory space.

12.6.6 Computation of the error correction code (ECC) in NAND Flash memory

The FMC NAND Card controller includes two error correction code computation hardware blocks, one per memory bank. They reduce the host CPU workload when processing the ECC by software.

These two ECC blocks are identical and associated with Bank 2 and Bank 3. As a consequence, no hardware ECC computation is available for memories connected to Bank 4.

The ECC algorithm implemented in the FMC can perform 1-bit error correction and 2-bit error detection per 256, 512, 1 024, 2 048, 4 096 or 8 192 bytes read or written from/to the NAND Flash memory. It is based on the Hamming coding algorithm and consists in calculating the row and column parity.

The ECC modules monitor the NAND Flash data bus and read/write signals (NCE and NWE) each time the NAND Flash memory bank is active.

The ECC operates as follows:

- When accessing NAND Flash memory bank 2 or bank 3, the data present on the D[15:0] bus is latched and used for ECC computation.
- When accessing any other address in NAND Flash memory, the ECC logic is idle, and does not perform any operation. As a result, write operations to define commands or addresses to the NAND Flash memory are not taken into account for ECC computation.

Once the desired number of bytes has been read/written from/to the NAND Flash memory by the host CPU, the FMC_ECCR registers must be read to retrieve the computed value. Once read, they should be cleared by resetting the ECCEN bit to '0'. To compute a new data block, the ECCEN bit must be set to one in the FMC_PCR registers.

To perform an ECC computation:

1. Enable the ECCEN bit in the FMC_PCR register.
2. Write data to the NAND Flash memory page. While the NAND page is written, the ECC block computes the ECC value.
3. Read the ECC value available in the FMC_ECCR register and store it in a variable.
4. Clear the ECCEN bit and then enable it in the FMC_PCR register before reading back the written data from the NAND page. While the NAND page is read, the ECC block computes the ECC value.
5. Read the new ECC value available in the FMC_ECCR register.
6. If the two ECC values are the same, no correction is required, otherwise there is an ECC error and the software correction routine returns information on whether the error can be corrected or not.

12.6.7 NAND Flash controller registers

NAND Flash control registers (FMC_PCR)

Address offset: 0x80

Reset value: 0x0000 0018

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ECCPS[2:0]			TAR3	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
TAR[2:0]				TCLR[3:0]				Res.	Res.	ECCEN	PWID[1:0]		PTYP	PBKEN	PWAITEN	Res.
RW	RW	RW	RW	RW	RW	RW			RW	RW	RW	RW	RW	RW	RW	

Bits 31:20 Reserved, must be kept at reset value.

Bits 19:17 **ECCPS[2:0]: ECC page size.**

Defines the page size for the extended ECC:

000: 256 bytes

001: 512 bytes

010: 1024 bytes

011: 2048 bytes

100: 4096 bytes

101: 8192 bytes

Bits 16:13 **TAR[3:0]: ALE to RE delay.**

Sets time from ALE low to RE low in number of AHB clock cycles (HCLK).

Time is: $t_{ar} = (TAR + SET + 2) \times THCLK$ where THCLK is the HCLK clock period

0000: 1 HCLK cycle (default)

1111: 16 HCLK cycles

Note: SET is MEMSET or ATTSET according to the addressed space.

Bits 12:9 **TCLR[3:0]**: CLE to RE delay.

Sets time from CLE low to RE low in number of AHB clock cycles (HCLK).

Time is $t_{clr} = (TCLR + SET + 2) \times THCLK$ where THCLK is the HCLK clock period

0000: 1 HCLK cycle (default)

1111: 16 HCLK cycles

Note: SET is MEMSET or ATTSET according to the addressed space.

Bits 8:7 Reserved, must be kept at reset value.

Bit 6 **ECCEN**: ECC computation logic enable bit

0: ECC logic is disabled and reset (default after reset),

1: ECC logic is enabled.

Bits 5:4 **PWID[1:0]**: Data bus width.

Defines the external memory device width.

00: 8 bits

01: 16 bits (default after reset).

10: reserved.

11: reserved.

Bit 3 **PTYP**: Memory type.

Defines the type of device attached to the corresponding memory bank:

0: Reserved, must be kept at reset value

1: NAND Flash (default after reset)

Bit 2 **PBKEN**: NAND Flash memory bank enable bit.

Enables the memory bank. Accessing a disabled memory bank causes an ERROR on AHB bus

0: Corresponding memory bank is disabled (default after reset)

1: Corresponding memory bank is enabled

Bit 1 **PWAITEN**: Wait feature enable bit.

Enables the Wait feature for the NAND Flash memory bank:

0: disabled

1: enabled

Bit 0 Reserved, must be kept at reset value.

FIFO status and interrupt register (FMC_SR)

Address offset: 0x84

Reset value: 0x0000 0040

This register contains information about the FIFO status and interrupt. The FMC features a FIFO that is used when writing to memories to transfer up to 16 words of data from the AHB.

This is used to quickly write to the FIFO and free the AHB for transactions to peripherals other than the FMC, while the FMC is draining its FIFO into the memory. One of these register bits indicates the status of the FIFO, for ECC purposes.

The ECC is calculated while the data are written to the memory. To read the correct ECC, the software must consequently wait until the FIFO is empty.

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

Bits 31:7 Reserved, must be kept at reset value.

Bit 6 **FEMPT**: FIFO empty.

Read-only bit that provides the status of the FIFO

0: FIFO not empty

1: FIFO empty

Bit 5 **IFEN**: Interrupt falling edge detection enable bit

0: Interrupt falling edge detection request disabled

1: Interrupt falling edge detection request enabled

Bit 4 **ILEN**: Interrupt high-level detection enable bit

0: Interrupt high-level detection request disabled

1: Interrupt high-level detection request enabled

Bit 3 **IREN**: Interrupt rising edge detection enable bit

0: Interrupt rising edge detection request disabled

1: Interrupt rising edge detection request enabled

Bit 2 **IFS**: Interrupt falling edge status

The flag is set by hardware and reset by software.

0: No interrupt falling edge occurred

1: Interrupt falling edge occurred

Note: If this bit is written by software to 1 it will be set.

Bit 1 **ILS**: Interrupt high-level status

The flag is set by hardware and reset by software.

0: No Interrupt high-level occurred

1: Interrupt high-level occurred

Bit 0 **IRS**: Interrupt rising edge status

The flag is set by hardware and reset by software.

0: No interrupt rising edge occurred

1: Interrupt rising edge occurred

Note: If this bit is written by software to 1 it will be set.

Common memory space timing register 2..4 (FMC_PMEM)

Address offset: Address: 0x88

Reset value: 0xFCFC FCFC

The FMC_PMEM read/write register contains the timing information for NAND Flash memory bank. This information is used to access either the common memory space of the NAND Flash for command, address write access and data read/write access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MEMHIZ[7:0]								MEMHOLD[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
MEMWAIT[7:0]								MEMSET[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **MEMHIZ[7:0]:** Common memory x data bus Hi-Z time

Defines the number of HCLK clock cycles during which the data bus is kept Hi-Z after the start of a NAND Flash write access to common memory space on socket. This is only valid for write transactions:

- 0000 0000: 1 HCLK cycle
- 1111 1110: 255 HCLK cycles
- 1111 1111: reserved.

Bits 23:16 **MEMHOLD[7:0]:** Common memory hold time

Defines the number of HCLK clock cycles for write access and HCLK (+2) clock cycles for read access during which the address is held (and data for write accesses) after the command is deasserted (NWE, NOE), for NAND Flash read or write access to common memory space on socket x:

- 0000 0000: reserved.
- 0000 0001: 1 HCLK cycle for write access / 3 HCLK cycles for read access
- 1111 1110: 254 HCLK cycles for write access / 256 HCLK cycles for read access
- 1111 1111: reserved.

Bits 15:8 **MEMWAIT[7:0]:** Common memory wait time

Defines the minimum number of HCLK (+1) clock cycles to assert the command (NWE, NOE), for NAND Flash read or write access to common memory space on socket. The duration of command assertion is extended if the wait signal (NWAIT) is active (low) at the end of the programmed value of HCLK:

- 0000 0000: reserved
- 0000 0001: 2HCLK cycles (+ wait cycle introduced by deasserting NWAIT)
- 1111 1110: 255 HCLK cycles (+ wait cycle introduced by deasserting NWAIT)
- 1111 1111: reserved.

Bits 7:0 **MEMSET[7:0]:** Common memory x setup time

Defines the number of HCLK (+1) clock cycles to set up the address before the command assertion (NWE, NOE), for NAND Flash read or write access to common memory space on socket x:

- 0000 0000: 1 HCLK cycle
- 1111 1110: 255 HCLK cycles
- 1111 1111: reserved

Attribute memory space timing registers (FMC_PATT)

Address offset: 0x8C

Reset value: 0xFCFC FCFC

The FMC_PATT read/write register contains the timing information for NAND Flash memory bank. It is used for 8-bit accesses to the attribute memory space of the NAND Flash for the last address write access if the timing must differ from that of previous accesses (for Ready/Busy management, refer to [Section 12.6.5: NAND Flash prewait functionality](#)).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ATTHZ[7:0]								ATTHold[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
ATTWAIT[7:0]								ATTSET[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **ATTHZ[7:0]:** Attribute memory data bus Hi-Z time

Defines the number of HCLK clock cycles during which the data bus is kept in Hi-Z after the start of a NAND Flash write access to attribute memory space on socket. Only valid for write transaction:

- 0000 0000: 0 HCLK cycle
- 1111 1110: 255 HCLK cycles
- 1111 1111: reserved.

Bits 23:16 **ATTHold[7:0]:** Attribute memory hold time

Defines the number of HCLK clock cycles for write access and HCLK (+2) clock cycles for read access during which the address is held (and data for write access) after the command deassertion (NWE, NOE), for NAND Flash read or write access to attribute memory space on socket:

- 0000 0000: reserved
- 0000 0001: 1 HCLK cycle for write access / 3 HCLK cycles for read access
- 1111 1110: 254 HCLK cycles for write access / 256 HCLK cycles for read access
- 1111 1111: reserved.

Bits 15:8 **ATTWAIT[7:0]:** Attribute memory wait time

Defines the minimum number of HCLK (+1) clock cycles to assert the command (NWE, NOE), for NAND Flash read or write access to attribute memory space on socket x. The duration for command assertion is extended if the wait signal (NWAIT) is active (low) at the end of the programmed value of HCLK:

- 0000 0000: reserved
- 0000 0001: 2 HCLK cycles (+ wait cycle introduced by deassertion of NWAIT)
- 1111 1110: 255 HCLK cycles (+ wait cycle introduced by deasserting NWAIT)
- 1111 1111: reserved.

Bits 7:0 **ATTSET[7:0]:** Attribute memory setup time

Defines the number of HCLK (+1) clock cycles to set up address before the command assertion (NWE, NOE), for NAND Flash read or write access to attribute memory space on socket:

- 0000 0000: 1 HCLK cycle
- 1111 1110: 255 HCLK cycles
- 1111 1111: reserved.

ECC result registers (FMC_ECCR)

Address offset: 0x94

Reset value: 0x0000 0000

This register contain the current error correction code value computed by the ECC computation modules of the FMC NAND controller. When the CPU reads the data from a NAND Flash memory page at the correct address (refer to [Section 12.6.6: Computation of the error correction code \(ECC\) in NAND Flash memory](#)), the data read/written from/to the NAND Flash memory are processed automatically by the ECC computation module. When X bytes have been read (according to the ECCPS field in the FMC_PCR registers), the CPU must read the computed ECC value from the FMC_ECC registers. It then verifies if these computed parity data are the same as the parity value recorded in the spare area, to determine whether a page is valid, and, to correct it otherwise. The FMC_ECCR register should be cleared after being read by setting the ECCEN bit to '0'. To compute a new data block, the ECCEN bit must be set to '1'.

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

Bits 31:0 **ECC[31:0]: ECC result**

This field contains the value computed by the ECC computation logic. [Table 79](#) describes the contents of these bit fields.

Table 79. ECC result relevant bits

ECCPS[2:0]	Page size in bytes	ECC bits
000	256	ECC[21:0]
001	512	ECC[23:0]
010	1024	ECC[25:0]
011	2048	ECC[27:0]
100	4096	ECC[29:0]
101	8192	ECC[31:0]

12.7 SDRAM controller

12.7.1 SDRAM controller main features

The main features of the SDRAM controller are the following:

- Two SDRAM banks with independent configuration
- 8-bit, 16-bit, 32-bit data bus width
- 13-bits Address Row, 11-bits Address Column, 4 internal banks: 4x16Mx32bit (256 MB), 4x16Mx16bit (128 MB), 4x16Mx8bit (64 MB)
- Word, half-word, byte access
- SDRAM clock can be HCLK/2 or HCLK/3
- Automatic row and bank boundary management
- Multibank ping-pong access
- Programmable timing parameters
- Automatic Refresh operation with programmable Refresh rate
- Self-refresh mode
- Power-down mode
- SDRAM power-up initialization by software
- CAS latency of 1,2,3
- Cacheable Read FIFO with depth of 6 lines x32-bit (6 x14-bit address tag)

12.7.2 SDRAM External memory interface signals

At startup, the SDRAM I/O pins used to interface the FMC SDRAM controller with the external SDRAM devices must be configured by the user application. The SDRAM controller I/O pins which are not used by the application, can be used for other purposes.

Table 80. SDRAM signals

SDRAM signal	I/O type	Description	Alternate function
SDCLK	O	SDRAM clock	-
SDCKE[1:0]	O	SDCKE0: SDRAM Bank 1 Clock Enable SDCKE1: SDRAM Bank 2 Clock Enable	-
SDNE[1:0]	O	SDNE0: SDRAM Bank 1 Chip Enable SDNE1: SDRAM Bank 2 Chip Enable	-
A[12:0]	O	Address	FMC_A[12:0]
D[31:0]	I/O	Bidirectional data bus	FMC_D[31:0]
BA[1:0]	O	Bank Address	FMC_A[15:14]
NRAS	O	Row Address Strobe	-
NCAS	O	Column Address Strobe	-
SDNWE	O	Write Enable	-
NBL[3:0]	O	Output Byte Mask for write accesses (memory signal name: DQM[3:0])	FMC_NBL[3:0]

12.7.3 SDRAM controller functional description

All SDRAM controller outputs (signals, address and data) change on the falling edge of the memory clock (FMC_SDCLK).

SDRAM initialization

The initialization sequence is managed by software. If the two banks are used, the initialization sequence must be generated simultaneously to Bank 1 and Bank 2 by setting the Target Bank bits CTB1 and CTB2 in the FMC_SDCMR register:

1. Program the memory device features into the FMC_SDCRx register. The SDRAM clock frequency, RBURST and RPIPE must be programmed in the FMC_SDCR1 register.
2. Program the memory device timing into the FMC_SDTRx register. The TRP and TRC timings must be programmed in the FMC_SDTR1 register.
3. Set MODE bits to '001' and configure the Target Bank bits (CTB1 and/or CTB2) in the FMC_SDCMR register to start delivering the clock to the memory (SDCKE is driven high).
4. Wait during the prescribed delay period. Typical delay is around 100 μ s (refer to the SDRAM datasheet for the required delay after power-up).
5. Set MODE bits to '010' and configure the Target Bank bits (CTB1 and/or CTB2) in the FMC_SDCMR register to issue a "Precharge All" command.
6. Set MODE bits to '011', and configure the Target Bank bits (CTB1 and/or CTB2) as well as the number of consecutive Auto-refresh commands (NRFS) in the FMC_SDCMR register. Refer to the SDRAM datasheet for the number of Auto-refresh commands that should be issued. Typical number is 8.
7. Configure the MRD field according to the SDRAM device, set the MODE bits to '100', and configure the Target Bank bits (CTB1 and/or CTB2) in the FMC_SDCMR register to issue a "Load Mode Register" command in order to program the SDRAM device. In particular:
 - a) the CAS latency must be selected following configured value in FMC_SDCR1/2 registers
 - b) the Burst Length (BL) of 1 must be selected by configuring the M[2:0] bits to 000 in the mode register. Refer to SDRAM device datasheet.

If the Mode Register is not the same for both SDRAM banks, this step has to be repeated twice, once for each bank, and the Target Bank bits set accordingly.

8. Program the refresh rate in the FMC_SDRTR register
The refresh rate corresponds to the delay between refresh cycles. Its value must be adapted to SDRAM devices.
9. For mobile SDRAM devices, to program the extended mode register it should be done once the SDRAM device is initialized: First, a dummy read access should be performed while BA1=1 and BA=0 (refer to SDRAM address mapping section for BA[1:0] address mapping) in order to select the extended mode register instead of the load mode register and then program the needed value.

At this stage the SDRAM device is ready to accept commands. If a system reset occurs during an ongoing SDRAM access, the data bus might still be driven by the SDRAM device. Therefor the SDRAM device must be first reinitialized after reset before issuing any new access by the NOR Flash/PSRAM/SRAM or NAND Flash controller.

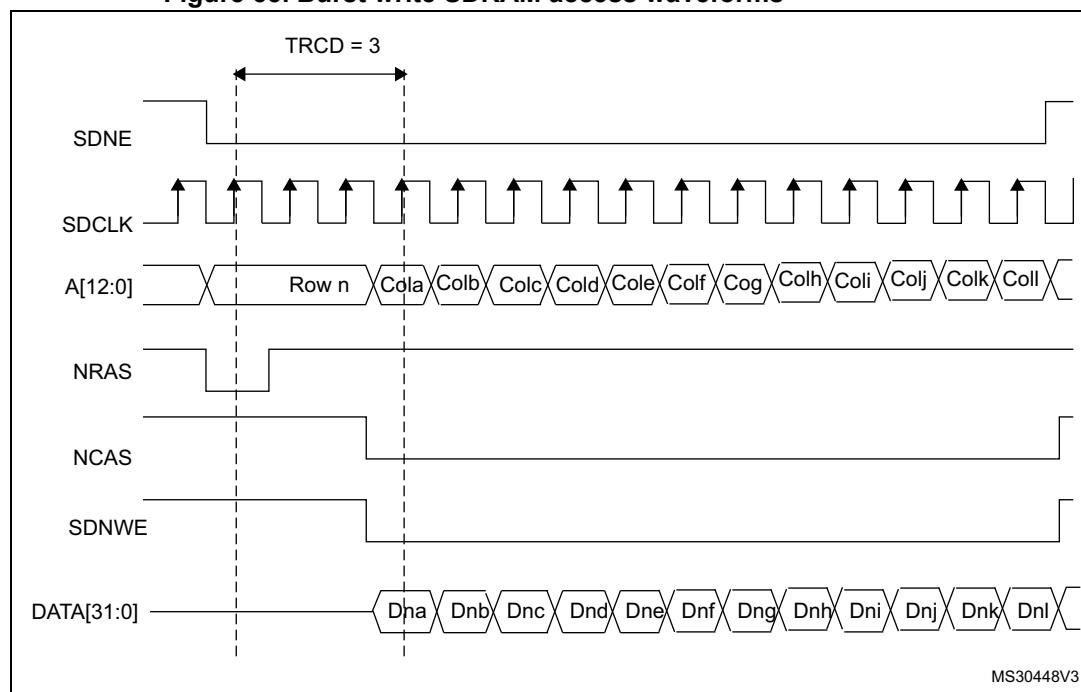
Note: If two SDRAM devices are connected to the FMC, all the accesses performed at the same time to both devices by the Command Mode register (Load Mode Register command) are issued using the timing parameters configured for SDRAM Bank 1 (TMRD and TRAS timings) in the FMC_SDTR1 register.

SDRAM controller write cycle

The SDRAM controller accepts single and burst write requests and translates them into single memory accesses. In both cases, the SDRAM controller keeps track of the active row for each bank to be able to perform consecutive write accesses to different banks (Multibank ping-pong access).

Before performing any write access, the SDRAM bank write protection must be disabled by clearing the WP bit in the FMC_SDCRx register.

Figure 55. Burst write SDRAM access waveforms



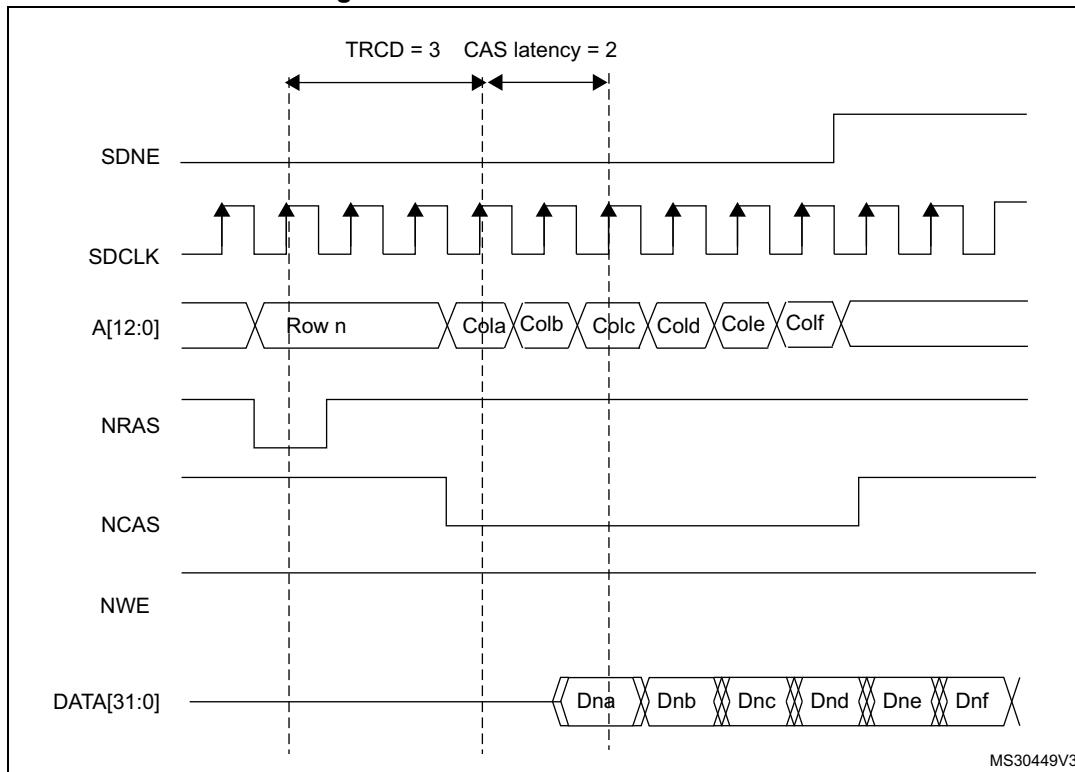
The SDRAM controller always checks the next access.

- If the next access is in the same row or in another active row, the write operation is carried out,
- if the next access targets another row (not active), the SDRAM controller generates a precharge command, activates the new row and initiates a write command.

SDRAM controller read cycle

The SDRAM controller accepts single and burst read requests and translates them into single memory accesses. In both cases, the SDRAM controller keeps track of the active row in each bank to be able to perform consecutive read accesses in different banks (Multibank ping-pong access).

Figure 56. Burst read SDRAM access



The FMC SDRAM controller features a Cacheable read FIFO (6 lines x 32 bits). It is used to store data read in advance during the CAS latency period and the RPIPE delay following the below formula. The RBURST bit must be set in the FMC_SDCR1 register to anticipate the next read access.

$$\text{Number of anticipated data} = \text{CAS latency} + 1 + (\text{RPIPE delay})/2$$

Examples:

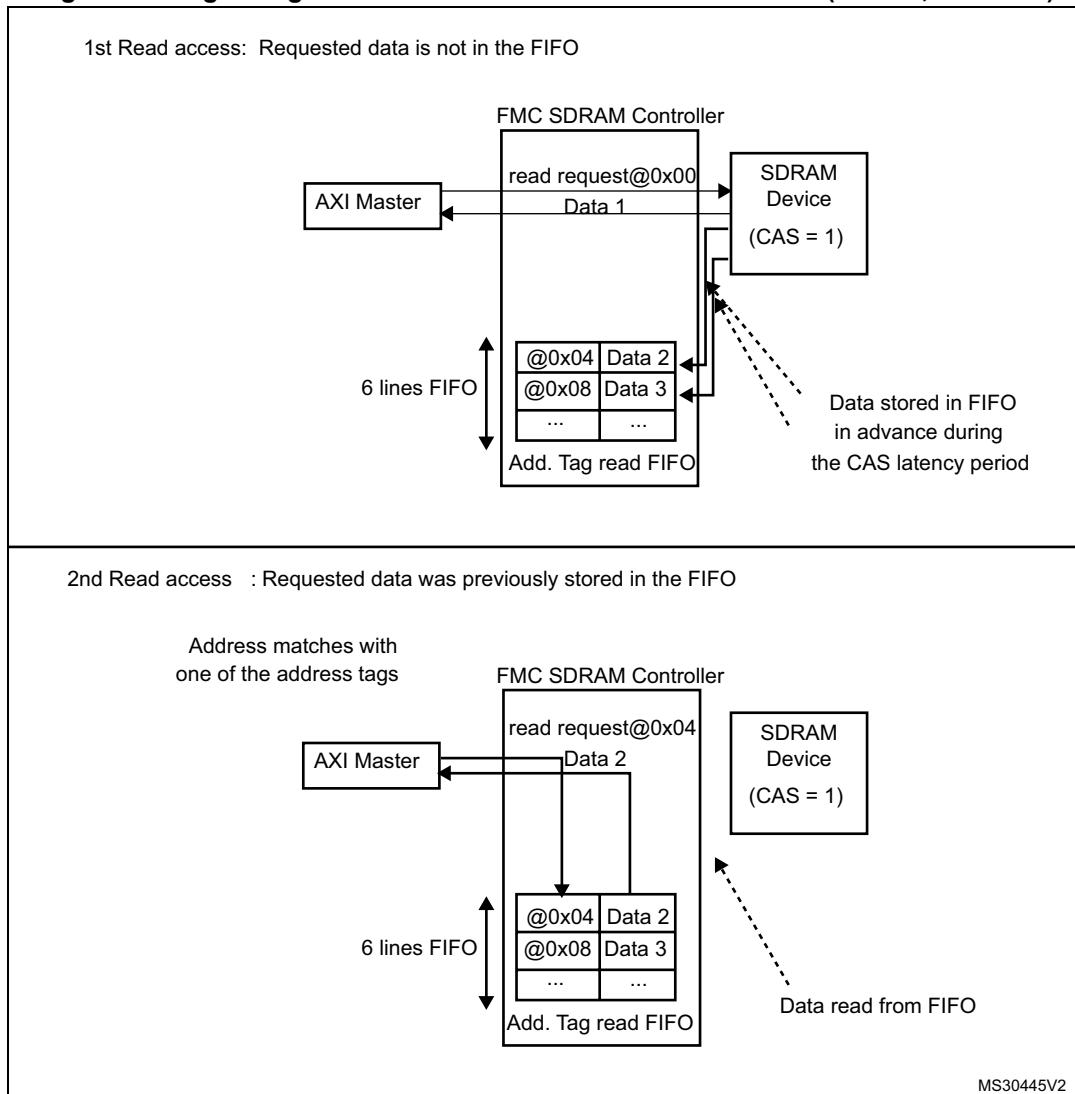
- CAS latency = 3, RPIPE delay = 0: Four data (not committed) are stored in the FIFO.
- CAS latency = 3, RPIPE delay = 2: Five data (not committed) are stored in the FIFO.

The read FIFO features a 14-bit address tag to each line to identify its content: 11 bits for the column address, 2 bits to select the internal bank and the active row, and 1 bit to select the SDRAM device

When the end of the row is reached in advance during an AHB burst read, the data read in advance (not committed) are not stored in the read FIFO. For single read access, data are correctly stored in the FIFO.

Each time a read request occurs, the SDRAM controller checks:

- If the address matches one of the address tags, data are directly read from the FIFO and the corresponding address tag/ line content is cleared and the remaining data in the FIFO are compacted to avoid empty lines.
- Otherwise, a new read command is issued to the memory and the FIFO is updated with new data. If the FIFO is full, the older data are lost.

Figure 57. Logic diagram of Read access with RBURST bit set (CAS=1, RPIPE=0)

During a write access or a Precharge command, the read FIFO is flushed and ready to be filled with new data.

After the first read request, if the current access was not performed to a row boundary, the SDRAM controller anticipates the next read access during the CAS latency period and the RPIPE delay (if configured). This is done by incrementing the memory address. The following condition must be met:

- RBURST control bit should be set to '1' in the FMC_SDCR1 register.

The address management depends on the next AHB request:

- Next AHB request is sequential (AHB Burst)
In this case, the SDRAM controller increments the address.
- Next AHB request is not sequential
 - If the new read request targets the same row or another active row, the new address is passed to the memory and the master is stalled for the CAS latency period, waiting for the new data from memory.
 - If the new read request does not target an active row, the SDRAM controller generates a Precharge command, activates the new row, and initiates a read command.

If the RURST is reset, the read FIFO is not used.

Row and bank boundary management

When a read or write access crosses a row boundary, if the next read or write access is sequential and the current access was performed to a row boundary, the SDRAM controller executes the following operations:

1. Precharge of the active row,
2. Activation of the new row
3. Start of a read/write command.

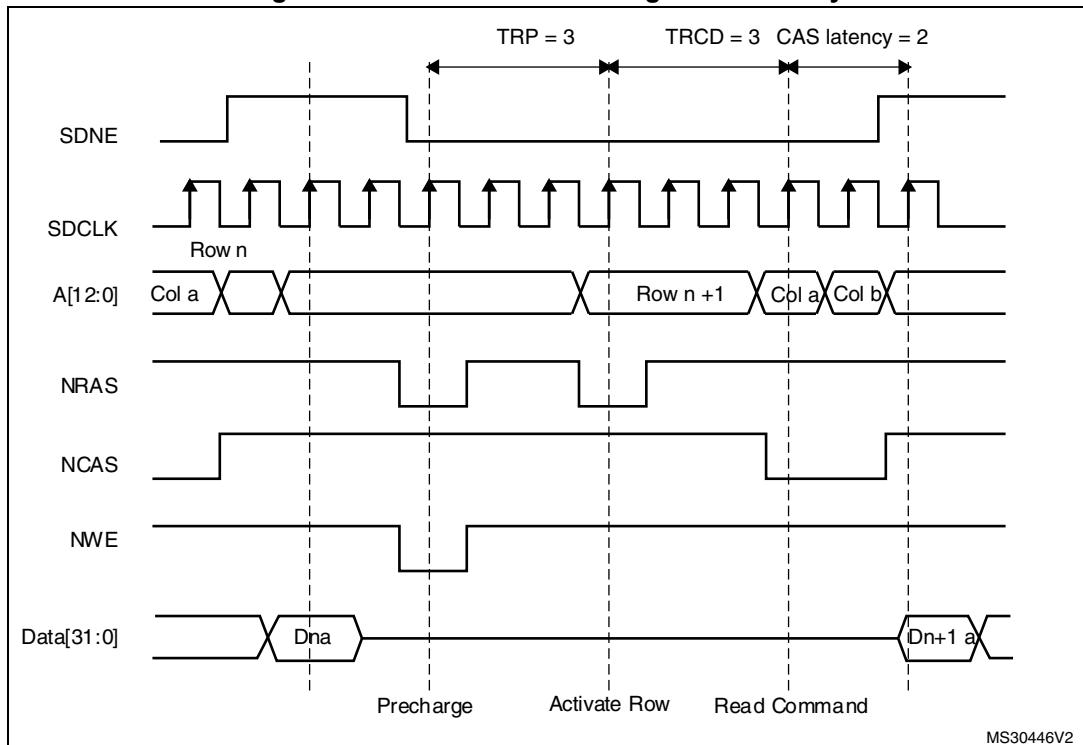
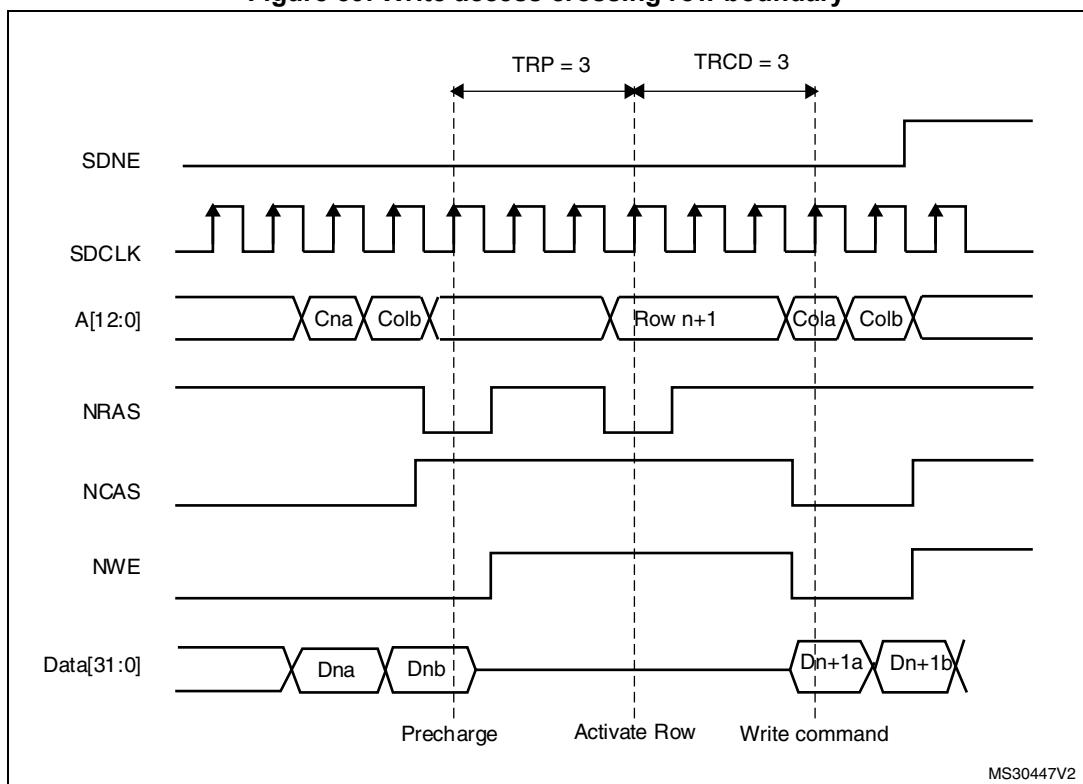
At a row boundary, the automatic activation of the next row is supported for all columns and data bus width configurations.

If necessary, the SDRAM controller inserts additional clock cycles between the following commands:

- Between Precharge and Active commands to match TRP parameter (only if the next access is in a different row in the same bank),
- Between Active and Read commands to match the TRCD parameter.

These parameters are defined into the FMC_SDTRx register.

Refer to [Figure 55](#) and [Figure 56](#) for read and burst write access crossing a row boundary.

Figure 58. Read access crossing row boundary**Figure 59. Write access crossing row boundary**

If the next access is sequential and the current access crosses a bank boundary, the SDRAM controller activates the first row in the next bank and initiates a new read/write command. Two cases are possible:

- If the current bank is not the last one, the active row in the new bank must be precharged. At a bank boundary, the automatic activation of the next row is supported for all rows/columns and data bus width configuration.
- If the current bank is the last one, the automatic activation of the next row is supported only when addressing 13-bit rows, 11-bit columns, 4 internal banks and 32-bit data bus SDRAM devices. Otherwise, the SDRAM address range is violated and an AHB error is generated.
- In case of 13-bit row address, 11-bit column address, 4 internal banks and bus width 32-bit SDRAM memories, at boundary bank, the SDRAM controller continues to read/write from the second SDRAM device (assuming it has been initialized):
 - a) The SDRAM controller activates the first row (after precharging the active row, if there is already an active row in the first internal bank, and initiates a new read/write command).
 - b) If the first row is already activated, the SDRAM controller just initiates a read/write command.

SDRAM controller refresh cycle

The Auto-refresh command is used to refresh the SDRAM device content. The SDRAM controller periodically issues auto-refresh commands. An internal counter is loaded with the COUNT value in the register FMC_SDRTR. This value defines the number of memory clock cycles between the refresh cycles (refresh rate). When this counter reaches zero, an internal pulse is generated.

If a memory access is ongoing, the auto-refresh request is delayed. However, if the memory access and the auto-refresh requests are generated simultaneously, the auto-refresh request takes precedence.

If the memory access occurs during an auto-refresh operation, the request is buffered and processed when the auto-refresh is complete.

If a new auto-refresh request occurs while the previous one was not served, the RE (Refresh Error) bit is set in the Status register. An Interrupt is generated if it has been enabled (REIE = '1').

If SDRAM lines are not in idle state (not all row are closed), the SDRAM controller generates a PALL (Precharge ALL) command before the auto-refresh.

If the Auto-refresh command is generated by the FMC_SDCMR Command Mode register (Mode bits = '011'), a PALL command (Mode bits = '010') must be issued first.

12.7.4 Low-power modes

Two low-power modes are available:

- Self-refresh mode
 - The auto-refresh cycles are performed by the SDRAM device itself to retain data without external clocking.
- Power-down mode
 - The auto-refresh cycles are performed by the SDRAM controller.

Self-refresh mode

This mode is selected by setting the MODE bits to '101' and by configuring the Target Bank bits (CTB1 and/or CTB2) in the FMC_SDCMR register.

The SDRAM clock stops running after a TRAS delay and the internal refresh timer stops counting only if one of the following conditions is met:

- A Self-refresh command is issued to both devices
- One of the devices is not activated (SDRAM bank is not initialized).

Before entering Self-Refresh mode, the SDRAM controller automatically issues a PALL command.

If the Write data FIFO is not empty, all data are sent to the memory before activating the Self-refresh mode and the BUSY status flag remains set.

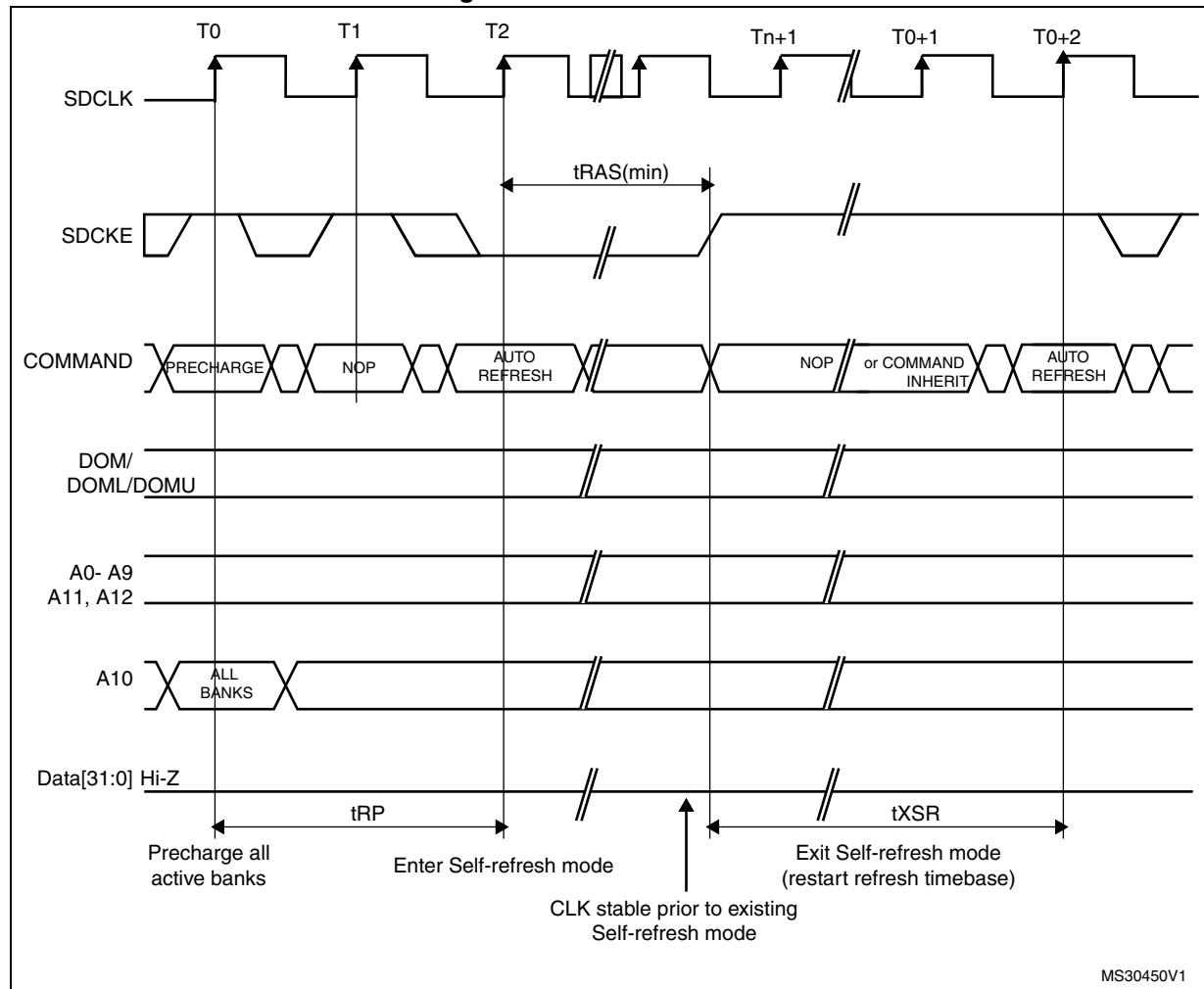
In Self-refresh mode, all SDRAM device inputs become don't care except for SDCKE which remains low.

The SDRAM device must remain in Self-refresh mode for a minimum period of time of TRAS and can remain in Self-refresh mode for an indefinite period beyond that. To guarantee this minimum period, the BUSY status flag remains high after the Self-refresh activation during a TRAS delay.

As soon as an SDRAM device is selected, the SDRAM controller generates a sequence of commands to exit from Self-refresh mode. After the memory access, the selected device remains in Normal mode.

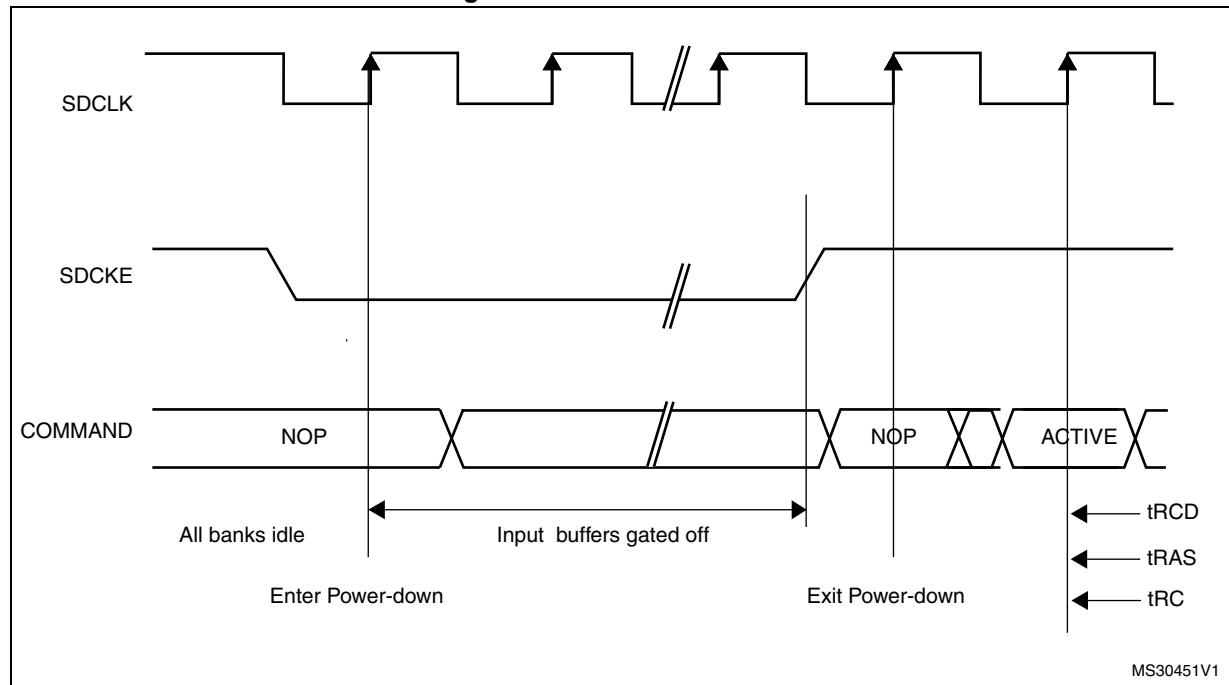
To exit from Self-refresh, the MODE bits must be set to '000' (Normal mode) and the Target Bank bits (CTB1 and/or CTB2) must be configured in the FMC_SDCMR register.

Figure 60. Self-refresh mode



Power-down mode

This mode is selected by setting the MODE bits to '110' and by configuring the Target Bank bits (CTB1 and/or CTB2) in the FMC_SDCMR register.

Figure 61. Power-down mode

If the Write data FIFO is not empty, all data are sent to the memory before activating the Power-down mode.

As soon as an SDRAM device is selected, the SDRAM controller exits from the Power-down mode. After the memory access, the selected SDRAM device remains in Normal mode.

During Power-down mode, all SDRAM device input and output buffers are deactivated except for the SDCKE which remains low.

The SDRAM device cannot remain in Power-down mode longer than the refresh period and cannot perform the Auto-refresh cycles by itself. Therefore, the SDRAM controller carries out the refresh operation by executing the operations below:

1. Exit from Power-down mode and drive the SDCKE high
2. Generate the PALL command only if a row was active during Power-down mode
3. Generate the auto-refresh command
4. Drive SDCKE low again to return to Power-down mode.

To exit from Power-down mode, the MODE bits must be set to '000' (Normal mode) and the Target Bank bits (CTB1 and/or CTB2) must be configured in the FMC_SDCMR register.

12.7.5 SDRAM controller registers

SDRAM Control registers 1,2 (FMC_SDCR1,2)

Address offset: 0x140+ 4* (x - 1), x = 1,2

Reset value: 0x0000 02D0

This register contains the control parameters for each SDRAM memory bank

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	RPIPE[1:0]	RBURST	SDCLK	WP	CAS	NB	MWID	NR						NC	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 14:13 **RPIPE[1:0]: Read pipe**

These bits define the delay, in HCLK clock cycles, for reading data after CAS latency.

- 00: No HCLK clock cycle delay
- 01: One HCLK clock cycle delay
- 10: Two HCLK clock cycle delay
- 11: reserved.

Note: The corresponding bits in the FMC_SDCR2 register is read only.

Bit 12 **RBURST: Burst read**

This bit enables Burst read mode. The SDRAM controller anticipates the next read commands during the CAS latency and stores data in the Read FIFO.

- 0: single read requests are not managed as bursts
- 1: single read requests are always managed as bursts

Note: The corresponding bit in the FMC_SDCR2 register is don't care.

Bits 11:10 **SDCLK[1:0]: SDRAM clock configuration**

These bits define the SDRAM clock period for both SDRAM banks and allow disabling the clock before changing the frequency. In this case the SDRAM must be re-initialized.

- 00: SDCLK clock disabled
- 01: reserved
- 10: SDCLK period = 2 x HCLK periods
- 11: SDCLK period = 3 x HCLK periods

Note: The corresponding bits in the FMC_SDCR2 register are don't care.

Bit 9 **WP: Write protection**

This bit enables write mode access to the SDRAM bank.

- 0: Write accesses allowed
- 1: Write accesses ignored

Bits 8:7 **CAS[1:0]: CAS Latency**

This bits sets the SDRAM CAS latency in number of memory clock cycles

- 00: reserved.
- 01: 1 cycle
- 10: 2 cycles
- 11: 3 cycles

Bit 6 **NB: Number of internal banks**

This bit sets the number of internal banks.

- 0: Two internal Banks
- 1: Four internal Banks

Bits 5:4 **MWID[1:0]**: Memory data bus width.

These bits define the memory device width.

- 00: 8 bits
- 01: 16 bits
- 10: 32 bits
- 11: reserved.

Bits 3:2 **NR[1:0]**: Number of row address bits

These bits define the number of bits of a row address.

- 00: 11 bit
- 01: 12 bits
- 10: 13 bits
- 11: reserved.

Bits 1:0 **NC[1:0]**: Number of column address bits

These bits define the number of bits of a column address.

- 00: 8 bits
- 01: 9 bits
- 10: 10 bits
- 11: 11 bits.

Note: Before modifying the RBURST or RPIPE settings or disabling the SDCLK clock, the user must first send a PALL command to make sure ongoing operations are complete.

SDRAM Timing registers 1,2 (FMC_SDTR1,2)

Address offset: $0x148 + 4 * (x - 1)$, $x = 1,2$

Reset value: 0xFFFF FFFF

This register contains the timing parameters of each SDRAM bank

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	TRCD				TRP				TWR			
				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
TRC				TRAS				TXSR				TMRD			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:28 Reserved, must be kept at reset value.

Bits 27:24 **TRCD[3:0]**: Row to column delay

These bits define the delay between the Activate command and a Read/Write command in number of memory clock cycles.

- 0000: 1 cycle.
- 0001: 2 cycles
-

- 1111: 16 cycles

Bits 23:20 **TRP[3:0]:** Row precharge delay

These bits define the delay between a Precharge command and another command in number of memory clock cycles. The TRP timing is only configured in the FMC_SDTR1 register. If two SDRAM devices are used, the TRP must be programmed with the timing of the slowest device.

0000: 1 cycle

0001: 2 cycles

....

1111: 16 cycles

Note: The corresponding bits in the FMC_SDTR2 register are don't care.

Bits 19:16 **TWR[3:0]:** Recovery delay

These bits define the delay between a Write and a Precharge command in number of memory clock cycles.

0000: 1 cycle

0001: 2 cycles

....

1111: 16 cycles

Note: TWR must be programmed to match the write recovery time (t_{WR}) defined in the SDRAM datasheet, and to guarantee that:

$TWR \geq TRAS - TRCD$ and $TWR \geq TRC - TRCD - TRP$

Example: TRAS= 4 cycles, TRCD= 2 cycles. So, TWR >= 2 cycles. TWR must be programmed to 0x1.

If two SDRAM devices are used, the FMC_SDTR1 and FMC_SDTR2 must be programmed with the same TWR timing corresponding to the slowest SDRAM device.

If only one SDRAM device is used, the TWR timing must be kept at reset value (0xF) for the not used bank.

Bits 15:12 **TRC[3:0]:** Row cycle delay

These bits define the delay between the Refresh command and the Activate command, as well as the delay between two consecutive Refresh commands. It is expressed in number of memory clock cycles. The TRC timing is only configured in the FMC_SDTR1 register. If two SDRAM devices are used, the TRC must be programmed with the timings of the slowest device.

0000: 1 cycle

0001: 2 cycles

....

1111: 16 cycles

Note: TRC must match the TRC and TRFC (Auto Refresh period) timings defined in the SDRAM device datasheet.

Note: The corresponding bits in the FMC_SDTR2 register are don't care.

Bits 11:8 **TRAS[3:0]:** Self refresh time

These bits define the minimum Self-refresh period in number of memory clock cycles.

0000: 1 cycle

0001: 2 cycles

....

1111: 16 cycles

Bits 7:4 TXSR[3:0]: Exit Self-refresh delay

These bits define the delay from releasing the Self-refresh command to issuing the Activate command in number of memory clock cycles.

0000: 1 cycle
0001: 2 cycles
....

1111: 16 cycles

Note: If two SDRAM devices are used, the FMC_SDTR1 and FMC_SDTR2 must be programmed with the same TXSR timing corresponding to the slowest SDRAM device.

Bits 3:0 TMRD[3:0]: Load Mode Register to Active

These bits define the delay between a Load Mode Register command and an Active or Refresh command in number of memory clock cycles.

0000: 1 cycle
0001: 2 cycles
....

1111: 16 cycles

Note: If two SDRAM devices are connected, all the accesses performed simultaneously to both devices by the Command Mode register (Load Mode Register command) are issued using the timing parameters configured for Bank 1 (TMRD and TRAS timings) in the FMC_SDTR1 register.

The TRP and TRC timings are only configured in the FMC_SDTR1 register. If two SDRAM devices are used, the TRP and TRC timings must be programmed with the timings of the slowest device.

SDRAM Command Mode register (FMC_SDCMR)

Address offset: 0x150

Reset value: 0x0000 0000

This register contains the command issued when the SDRAM device is accessed. This register is used to initialize the SDRAM device, and to activate the Self-refresh and the Power-down modes. As soon as the MODE field is written, the command will be issued only to one or to both SDRAM banks according to CTB1 and CTB2 command bits. This register is the same for both SDRAM banks.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16								
Res.	MRD																						
										rw	rw	rw	rw	rw	rw								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0								
MRD								NRFS				CTB1	CTB2	MODE									
rw	rw	rw	rw	rw	rw	rw																	

Bits 31:22 Reserved, must be kept at reset value.

Bits 21:9 MRD[12:0]: Mode Register definition

This 13-bit field defines the SDRAM Mode Register content. The Mode Register is programmed using the Load Mode Register command.

Bits 8:5 NRFS[3:0]: Number of Auto-refresh

These bits define the number of consecutive Auto-refresh commands issued when MODE = '011'.

0000: 1 Auto-refresh cycle

0001: 2 Auto-refresh cycles

....

1110: 15 Auto-refresh cycles

1111: 16 Auto-refresh cycles

Bit 4 CTB1: Command Target Bank 1

This bit indicates whether the command will be issued to SDRAM Bank 1 or not.

0: Command not issued to SDRAM Bank 1

1: Command issued to SDRAM Bank 1

Bit 3 CTB2: Command Target Bank 2

This bit indicates whether the command will be issued to SDRAM Bank 2 or not.

0: Command not issued to SDRAM Bank 2

1: Command issued to SDRAM Bank 2

Bits 2:0 MODE[2:0]: Command mode

These bits define the command issued to the SDRAM device.

000: Normal Mode

001: Clock Configuration Enable

010: PALL ("All Bank Precharge") command

011: Auto-refresh command

100: Load Mode Register

101: Self-refresh command

110: Power-down command

111: Reserved

Note: When a command is issued, at least one Command Target Bank bit (CTB1 or CTB2) must be set otherwise the command will be ignored.

Note: If two SDRAM banks are used, the Auto-refresh and PALL command must be issued simultaneously to the two devices with CTB1 and CTB2 bits set otherwise the command will be ignored.

Note: If only one SDRAM bank is used and a command is issued with it's associated CTB bit set, the other CTB bit of the the unused bank must be kept to 0.

SDRAM Refresh Timer register (FMC_SDRTR)

Address offset:0x154

Reset value: 0x0000 0000

This register sets the refresh rate in number of SDCLK clock cycles between the refresh cycles by configuring the Refresh Timer Count value.

$$\text{Refresh rate} = (\text{COUNT} + 1) \times \text{SDRAM clock frequency}$$

$$\text{COUNT} = (\text{SDRAM refresh period} / \text{Number of rows}) - 20$$

Example

$$\text{Refresh rate} = 64 \text{ ms} / (8196 \text{ rows}) = 7.81 \mu\text{s}$$

where 64 ms is the SDRAM refresh period.

$$7.81\mu\text{s} \times 60\text{MHz} = 468.6$$

The refresh rate must be increased by 20 SDRAM clock cycles (as in the above example) to obtain a safe margin if an internal refresh request occurs when a read request has been accepted. It corresponds to a COUNT value of '0000111000000' (448).

This 13-bit field is loaded into a timer which is decremented using the SDRAM clock. This timer generates a refresh pulse when zero is reached. The COUNT value must be set at least to 41 SDRAM clock cycles.

As soon as the FMC_SDRTR register is programmed, the timer starts counting. If the value programmed in the register is '0', no refresh is carried out. This register must not be reprogrammed after the initialization procedure to avoid modifying the refresh rate.

Each time a refresh pulse is generated, this 13-bit COUNT field is reloaded into the counter.

If a memory access is in progress, the Auto-refresh request is delayed. However, if the memory access and Auto-refresh requests are generated simultaneously, the Auto-refresh takes precedence. If the memory access occurs during a refresh operation, the request is buffered to be processed when the refresh is complete.

This register is common to SDRAM bank 1 and bank 2.

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

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

Bit 14 **REIE:** RES Interrupt Enable

0: Interrupt is disabled

1: An Interrupt is generated if RE = 1

Bits 13:1 **COUNT[12:0]:** Refresh Timer Count

This 13-bit field defines the refresh rate of the SDRAM device. It is expressed in number of memory clock cycles. It must be set at least to 41 SDRAM clock cycles (0x29).

Refresh rate = (COUNT + 1) x SDRAM frequency clock

COUNT = (SDRAM refresh period / Number of rows) - 20

Bit 0 **CRE:** Clear Refresh error flag

This bit is used to clear the Refresh Error Flag (RE) in the Status Register.

0: no effect

1: Refresh Error flag is cleared

Note: The programmed COUNT value must not be equal to the sum of the following timings:
TWR+TRP+TRC+TRCD+4 memory clock cycles.

SDRAM Status register (FMC_SDSR)

Address offset: 0x158

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	BUSY	MODES2	MODES1	RE											
										r	r	r	r	r	r

Bits 31:5 Reserved, must be kept at reset value.

Bit 5 BUSY: Busy status

This bit defines the status of the SDRAM controller after a Command Mode request

0: SDRAM Controller is ready to accept a new request

1; SDRAM Controller is not ready to accept a new request

Bits 4:3 MODES2[1:0]: Status Mode for Bank 2

This bit defines the Status Mode of SDRAM Bank 2.

00: Normal Mode

01: Self-refresh mode

10: Power-down mode

Bits 2:1 MODES1[1:0]: Status Mode for Bank 1

This bit defines the Status Mode of SDRAM Bank 1.

00: Normal Mode

01: Self-refresh mode

10: Power-down mode

Bit 0 RE: Refresh error flag

0: No refresh error has been detected

1: A refresh error has been detected

An interrupt is generated if REIE = 1 and RE = 1

12.8 FMC register map

Table 81. FMC register map

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	FMC_BCR1	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CBURSTRW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
	Reset value																																			
0x08	FMC_BCR2	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CBURSTRW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
	Reset value																																			
0x10	FMC_BCR3	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CBURSTRW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
	Reset value																																			
0x18	FMC_BCR4	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CBURSTRW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
	Reset value																																			
0x04	FMC_BTR1	Res.	Res.	Res.	Res.	0 ACCMOD[1:0]	DATLAT[3:0]	CLKDIV[3:0]	BUSTURN[3:0]																											
	Reset value					0 ACCMOD[1:0]	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1																							
0x0C	FMC_BTR2	Res.	Res.	Res.	Res.	0 ACCMOD[1:0]	DATLAT[3:0]	CLKDIV[3:0]	BUSTURN[3:0]																											
	Reset value					0 ACCMOD[1:0]	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1																							
0x14	FMC_BTR3	Res.	Res.	Res.	Res.	0 ACCMOD[1:0]	DATLAT[3:0]	CLKDIV[3:0]	BUSTURN[3:0]																											
	Reset value					0 ACCMOD[1:0]	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1																							
0x1C	FMC_BTR4	Res.	Res.	Res.	Res.	0 ACCMOD[1:0]	DATLAT[3:0]	CLKDIV[3:0]	BUSTURN[3:0]																											
	Reset value					0 ACCMOD[1:0]	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1																							
0x104	FMC_BWTR1	Res.	Res.	Res.	Res.	0 ACCMOD[1:0]			BUSTURN[3:0]																											
	Reset value					0 ACCMOD[1:0]				1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1																						
0x10C	FMC_BWTR2	Res.	Res.	Res.	Res.	0 ACCMOD[1:0]			BUSTURN[3:0]																											
	Reset value					0 ACCMOD[1:0]				1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1																						

Table 81. FMC register map (continued)

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

13 Quad-SPI interface (QUADSPI)

13.1 Introduction

The QUADSPI is a specialized communication interface targeting single, dual or quad SPI Flash memories. It can operate in any of the three following modes:

- indirect mode: all the operations are performed using the QUADSPI registers
- status polling mode: the external Flash memory status register is periodically read and an interrupt can be generated in case of flag setting
- memory-mapped mode: the external Flash memory is mapped to the microcontroller address space and is seen by the system as if it was an internal memory

Both throughput and capacity can be increased two-fold using dual-flash mode, where two Quad-SPI Flash memories are accessed simultaneously.

13.2 QUADSPI main features

- Three functional modes: indirect, status-polling, and memory-mapped
- Dual-flash mode, where 8 bits can be sent/received simultaneously by accessing two Flash memories in parallel.
- SDR and DDR support
- Fully programmable opcode for both indirect and memory mapped mode
- Fully programmable frame format for both indirect and memory mapped mode
- Integrated FIFO for reception and transmission
- 8, 16, and 32-bit data accesses are allowed
- DMA channel for indirect mode operations
- Interrupt generation on FIFO threshold, timeout, operation complete, and access error

13.3 QUADSPI functional description

13.3.1 QUADSPI block diagram

Figure 62. QUADSPI block diagram when dual-flash mode is disabled

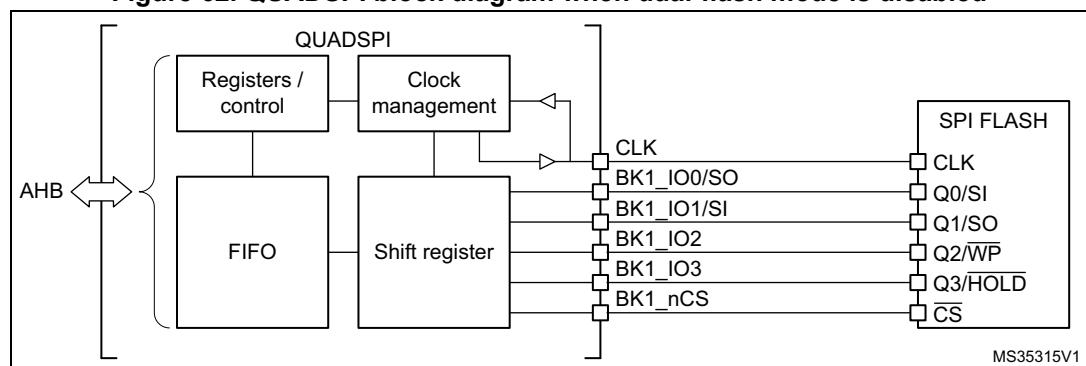
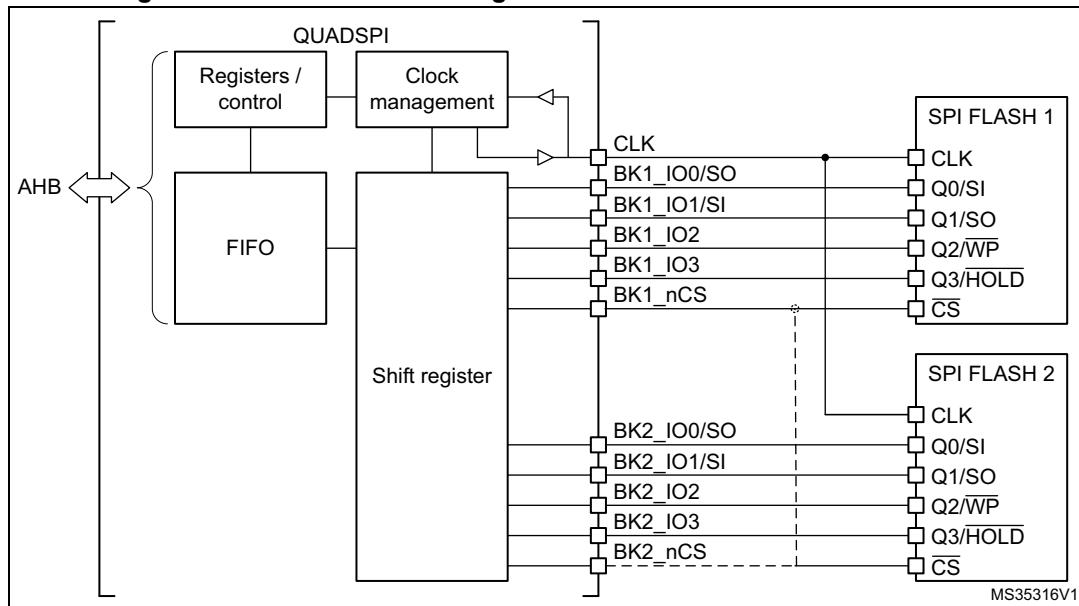


Figure 63. QUADSPI block diagram when dual-flash mode is enabled

13.3.2 QUADSPI pins

Table 82 lists the QUADSPI pins, six for interfacing with a single Flash memory, or 10 to 11 for interfacing with two Flash memories (FLASH 1 and FLASH 2) in dual-flash mode.

Table 82. QUADSPI pins

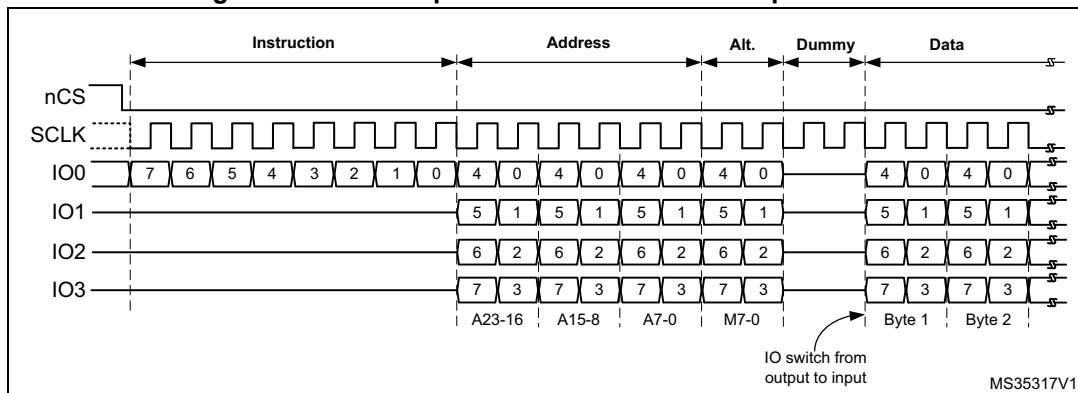
Signal name	Signal type	Description
CLK	Digital output	Clock to FLASH 1 and FLASH 2
BK1_IO0/SO	Digital input/output	Bidirectional IO in dual/quad modes or serial output in single mode, for FLASH 1
BK1_IO1/SI	Digital input/output	Bidirectional IO in dual/quad modes or serial input in single mode, for FLASH 1
BK1_IO2	Digital input/output	Bidirectional IO in quad mode, for FLASH 1
BK1_IO3	Digital input/output	Bidirectional IO in quad mode, for FLASH 1
BK2_IO0/SO	Digital input/output	Bidirectional IO in dual/quad modes or serial output in single mode, for FLASH 2
BK2_IO1/SI	Digital input/output	Bidirectional IO in dual/quad modes or serial input in single mode, for FLASH 2
BK2_IO2	Digital input/output	Bidirectional IO in quad mode, for FLASH 2
BK2_IO3	Digital input/output	Bidirectional IO in quad mode, for FLASH 2
BK1_nCS	Digital output	Chip select (active low) for FLASH 1. Can also be used for FLASH 2 if QUADSPI is always used in dual-flash mode.
BK2_nCS	Digital output	Chip select (active low) for FLASH 2. Can also be used for FLASH 1 if QUADSPI is always used in dual-flash mode.

13.3.3 QUADSPI command sequence

The QUADSPI communicates with the Flash memory using commands. Each command can include 5 phases: instruction, address, alternate byte, dummy, data. Any of these phases can be configured to be skipped, but at least one of the instruction, address, alternate byte, or data phase must be present.

nCS falls before the start of each command and rises again after each command finishes.

Figure 64. An example of a read command in quad mode



Instruction phase

During this phase, an 8-bit instruction, configured in INSTRUCTION field of QUADSPI_CCR[7:0] register, is sent to the Flash memory, specifying the type of operation to be performed.

Though most Flash memories can receive instructions only one bit at a time from the IO0/SO signal (single SPI mode), the instruction phase can optionally send 2 bits at a time (over IO0/IO1 in dual SPI mode) or 4 bits at a time (over IO0/IO1/IO2/IO3 in quad SPI mode). This can be configured using the IMODE[1:0] field of QUADSPI_CCR[9:8] register.

When IMODE = 00, the instruction phase is skipped, and the command sequence starts with the address phase, if present.

Address phase

In the address phase, 1-4 bytes are sent to the Flash memory to indicate the address of the operation. The number of address bytes to be sent is configured in the ADSIZE[1:0] field of QUADSPI_CCR[13:12] register. In indirect and automatic-polling modes, the address bytes to be sent are specified in the ADDRESS[31:0] field of QUADSPI_AR register, while in memory-mapped mode the address is given directly via the AHB (from the Cortex® or from a DMA).

The address phase can send 1 bit at a time (over SO in single SPI mode), 2 bits at a time (over IO0/IO1 in dual SPI mode), or 4 bits at a time (over IO0/IO1/IO2/IO3 in quad SPI mode). This can be configured using the ADMODE[1:0] field of QUADSPI_CCR[11:10] register.

When ADMODE = 00, the address phase is skipped, and the command sequence proceeds directly to the next phase, if any.

Alternate-bytes phase

In the alternate-bytes phase, 1-4 bytes are sent to the Flash memory, generally to control the mode of operation. The number of alternate bytes to be sent is configured in the ABSIZE[1:0] field of QUADSPI_CCR[17:16] register. The bytes to be sent are specified in the QUADSPI_ABR register.

The alternate-bytes phase can send 1 bit at a time (over SO in single SPI mode), 2 bits at a time (over IO0/IO1 in dual SPI mode), or 4 bits at a time (over IO0/IO1/IO2/IO3 in quad SPI mode). This can be configured using the ABMODE[1:0] field of QUADSPI_CCR[15:14] register.

When ABMODE = 00, the alternate-bytes phase is skipped, and the command sequence proceeds directly to the next phase, if any.

There may be times when only a single nibble needs to be sent during the alternate-byte phase rather than a full byte, such as when dual-mode is used and only two cycles are used for the alternate bytes. In this case, firmware can use quad-mode (ABMODE = 11) and send a byte with bits 7 and 3 of ALTERNATE set to '1' (keeping the IO3 line high), and bits 6 and 2 set to '0' (keeping the IO2 line low). In this case the upper two bits of the nibble to be sent are placed in bits 4:3 of ALTERNATE while the lower two bits are placed in bits 1 and 0. For example, if the nibble 2 (0010) is to be sent over IO0/IO1, then ALTERNATE should be set to 0x8A (1000_1010).

Dummy-cycles phase

In the dummy-cycles phase, 1-31 cycles are given without any data being sent or received, in order to allow the Flash memory the time to prepare for the data phase when higher clock frequencies are used. The number of cycles given during this phase is specified in the DCYC[4:0] field of QUADSPI_CCR[22:18] register. In both SDR and DDR modes, the duration is specified as a number of full CLK cycles.

When DCYC is zero, the dummy-cycles phase is skipped, and the command sequence proceeds directly to the data phase, if present.

The operating mode of the dummy-cycles phase is determined by DMODE.

In order to assure enough “turn-around” time for changing the data signals from output mode to input mode, there must be at least one dummy cycle when using dual or quad mode to receive data from the Flash memory.

Data phase

During the data phase, any number of bytes can be sent to, or received from the Flash memory.

In indirect and automatic-polling modes, the number of bytes to be sent/received is specified in the QUADSPI_DLR register.

In indirect write mode the data to be sent to the Flash memory must be written to the QUADSPI_DR register, while in indirect read mode the data received from the Flash memory is obtained by reading from the QUADSPI_DR register.

In memory-mapped mode, the data which is read is sent back directly over the AHB to the Cortex or to a DMA.

The data phase can send/receive 1 bit at a time (over SO/SI in single SPI mode), 2 bits at a time (over IO0/IO1 in dual SPI mode), or 4 bits at a time (over IO0/IO1/IO2/IO3 in quad SPI

mode). This can be configured using the ABMODE[1:0] field of QUADSPI_CCR[15:14] register.

When DMODE = 00, the data phase is skipped, and the command sequence finishes immediately by raising nCS. This configuration must only be used in only indirect write mode.

13.3.4 QUADSPI signal interface protocol modes

Single SPI mode

Legacy SPI mode allows just a single bit to be sent/received serially. In this mode, data is sent to the Flash memory over the SO signal (whose I/O shared with IO0). Data received from the Flash memory arrives via SI (whose I/O shared with IO1).

The different phases can each be configured separately to use this single bit mode by setting the IMODE/ADMODE/ABMODE/DMODE fields (in QUADSPI_CCR) to 01.

In each phase which is configured in single mode:

- IO0 (SO) is in output mode
- IO1 (SI) is in input mode (high impedance)
- IO2 is in output mode and forced to '0' (to deactivate the "write protect" function)
- IO3 is in output mode and forced to '1' (to deactivate the "hold" function)

This is the case even for the dummy phase if DMODE = 01.

Dual SPI mode

In dual SPI mode, two bits are sent/received simultaneously over the IO0/IO1 signals.

The different phases can each be configured separately to use dual SPI mode by setting the IMODE/ADMODE/ABMODE/DMODE fields of QUADSPI_CCR register to 10.

In each phase which is configured in dual mode:

- IO0/IO1 are at high-impedance (input) during the data phase for read operations, and outputs in all other cases
- IO2 is in output mode and forced to '0'
- IO3 is in output mode and forced to '1'

In the dummy phase when DMODE = 01, IO0/IO1 are always high-impedance.

Quad SPI mode

In quad SPI mode, four bits are sent/received simultaneously over the IO0/IO1/IO2/IO3 signals.

The different phases can each be configured separately to use quad SPI mode by setting the IMODE/ADMODE/ABMODE/DMODE fields of QUADSPI_CCR register to 11.

In each phase which is configured in quad mode, IO0/IO1/IO2/IO3 are all are at high-impedance (input) during the data phase for read operations, and outputs in all other cases.

In the dummy phase when DMODE = 11, IO0/IO1/IO2/IO3 are all high-impedance.

IO2 and IO3 are used only in Quad SPI mode. If none of the phases are configured to use Quad SPI mode, then the pins corresponding to IO2 and IO3 can be used for other functions even while QUADSPI is active.

SDR mode

By default, the DDRM bit (QUADSPI_CCR[31]) is 0 and the QUADSPI operates in single data rate (SDR) mode.

In SDR mode, when the QUADSPI is driving the IO0/SO, IO1, IO2, IO3 signals, these signals transition only with the falling edge of CLK.

When receiving data in SDR mode, the QUADSPI assumes that the Flash memories also send the data using CLK's falling edge. By default (when SSHIFT = 0), the signals are sampled using the following (rising) edge of CLK.

DDR mode

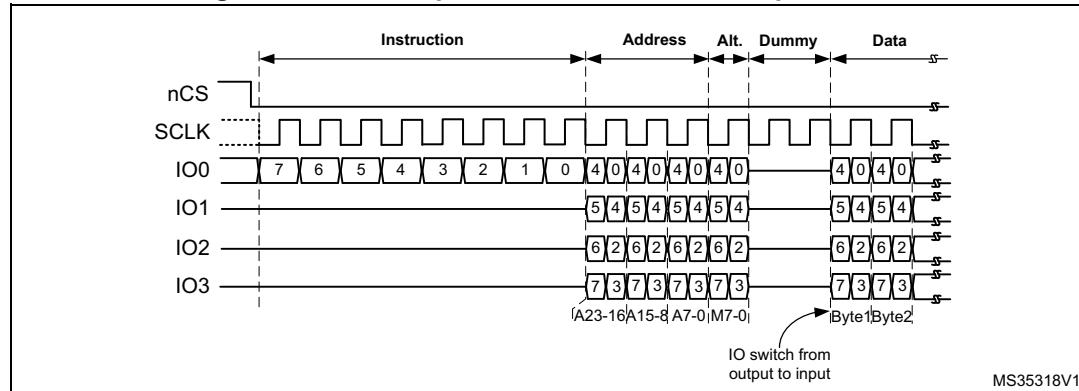
When the DDRM bit (QUADSPI_CCR[31]) is set to 1, the QUADSPI operates in double data rate (DDR) mode.

In DDR mode, when the QUADSPI is driving the IO0/SO, IO1, IO2, IO3 signals in the address/alternate-byte/data phases, a bit is sent on each of the falling and rising edges of CLK.

The instruction phase is not affected by DDRM. The instruction is always sent using CLK's falling edge.

When receiving data in DDR mode, the QUADSPI assumes that the Flash memories also send the data using both rising and falling CLK edges. When DDRM = 1, firmware must clear SSHIFT bit (bit 4 of QUADSPI_CR). Thus, the signals are sampled one half of a CLK cycle later (on the following, opposite edge).

Figure 65. An example of a DDR command in quad mode



Dual-flash mode

When the DFM bit (bit 6 of QUADSPI_CR) is 1, the QUADSPI is in dual-flash mode, where two external quad SPI Flash memories (FLASH 1 and FLASH 2) are used in order to send/receive 8 bits (or 16 bits in DDR mode) every cycle, effectively doubling the throughput as well as the capacity.

Each of the Flash memories use the same CLK and optionally the same nCS signals, but each have separate IO0, IO1, IO2, and IO3 signals.

Dual-flash mode can be used in conjunction with single-bit, dual-bit, and quad-bit modes, as well as with either SDR or DDR mode.

The Flash memory size, as specified in FSIZE[4:0] (QUADSPI_DCR[20:16]), should reflect the total Flash memory capacity, which is double the size of one individual component.

If address X is even, then the byte which the QUADSPI gives for address X is the byte at the address X/2 of FLASH 1, and the byte which the QUADSPI gives for address X+1 is the byte at the address X/2 of FLASH 2. In other words, bytes at even addresses are all stored in FLASH 1 and bytes at odd addresses are all stored in FLASH 2.

When reading the Flash memories status registers in dual-flash mode, twice as many bytes should be read compared to doing the same read in single-flash mode. This means that if each Flash memory gives 8 valid bits after the instruction for fetching the status register, then the QUADSPI must be configured with a data length of 2 bytes (16 bits), and the QUADSPI will receive one byte from each Flash memory. If each Flash memory gives a status of 16 bits, then the QUADSPI must be configured to read 4 bytes to get all the status bits of both Flash memories in dual-flash mode. The least-significant byte of the result (in the data register) is the least-significant byte of FLASH 1 status register, while the next byte is the least-significant byte of FLASH 2 status register. Then, the third byte of the data register is FLASH 1 second byte, while the forth byte is FLASH 2 second byte (in the case that the Flash memories have 16-bit status registers).

An even number of bytes must always be accessed in dual-flash mode. For this reason, bit 0 of the data length field (QUADSPI_DLR[0]) is stuck at 1 when DRM = 1.

In dual-flash mode, the behavior of FLASH 1 interface signals are basically the same as in normal mode. FLASH 2 interface signals have exactly the same waveforms as FLASH 1 during the instruction, address, alternate-byte, and dummy-cycles phases. In other words, each Flash memory always receives the same instruction and the same address. Then, during the data phase, the BK1_IOx and BK2_IOx buses are both transferring data in parallel, but the data that are sent to (or received from) FLASH 1 are distinct from those of FLASH 2.

13.3.5 QUADSPI indirect mode

When in indirect mode, commands are started by writing to QUADSPI registers and data is transferred by writing or reading the data register, in the same way as for other communication peripherals.

When FMODE = 00 (QUADSPI_CCR[27:26]), the QUADSPI is in indirect write mode, where bytes are sent to the Flash memory during the data phase. Data are provided by writing to the data register (QUADSPI_DR).

When FMODE = 01, the QUADSPI is in indirect read mode, where bytes are received from the Flash memory during the data phase. Data are recovered by reading QUADSPI_DR.

The number of bytes to be read/written is specified in the data length register (QUADSPI_DLR). If QUADSPI_DLR = 0xFFFF_FFFF (all 1's), then the data length is considered undefined and the QUADSPI simply continues to transfer data until the end of Flash memory (as defined by FSIZE) is reached. If no bytes are to be transferred, DMODE (QUADSPI_CCR[25:24]) should be set to 00.

If QUADSPI_DLR = 0xFFFF_FFFF and FSIZE = 0x1F (max value indicating a 4GB Flash memory), then in this special case the transfers continue indefinitely, stopping only after an abort request or after the QUADSPI is disabled. After the last memory address is read (at address 0xFFFF_FFFF), reading continues with address = 0x0000_0000.

When the programmed number of bytes to be transmitted or received is reached, TCF is set and an interrupt is generated if TCIE = 1. In the case of undefined number of data, the TCF

is set when the limit of the external SPI memory is reached according to the Flash memory size defined in the QUADSPI_CR.

Triggering the start of a command

Essentially, a command starts as soon as firmware gives the last information that is necessary for this command. Depending on the QUADSPI configuration, there are three different ways to trigger the start of a command in indirect mode. The commands starts immediately after:

1. a write is performed to INSTRUCTION[7:0] (QUADSPI_CCR), if no address is necessary (when ADMODE = 00) and if no data needs to be provided by the firmware (when FMODE = 01 or DMODE = 00)
2. a write is performed to ADDRESS[31:0] (QUADSPI_AR), if an address is necessary (when ADMODE != 00) and if no data needs to be provided by the firmware (when FMODE = 01 or DMODE = 00)
3. a write is performed to DATA[31:0] (QUADSPI_DR), if an address is necessary (when ADMODE != 00) and if data needs to be provided by the firmware (when FMODE = 00 and DMODE != 00)

Writes to the alternate byte register (QUADSPI_ABR) never trigger the communication start. If alternate bytes are required, they must be programmed before.

As soon as a command is started, the BUSY bit (bit 5 of QUADSPI_SR) is automatically set.

FIFO and data management

In indirect mode, data go through a 32-byte FIFO which is internal to the QUADSPI. FLEVEL[5:0] (QUADSPI_SR[13:8]) indicates how many bytes are currently being held in the FIFO.

In indirect write mode (FMODE = 00), firmware adds data to the FIFO when it writes QUADSPI_DR. Word writes add 4 bytes to the FIFO, halfword writes add 2 bytes, and byte writes add only 1 byte. If firmware adds too many bytes to the FIFO (more than is indicated by DL[31:0]), the extra bytes are flushed from the FIFO at the end of the write operation (when TCF is set).

Byte/halfword accesses to QUADSPI_DR must be done only to the least significant byte/halfword of the 32-bit register.

FTHRES[3:0] is used to define a FIFO threshold. When the threshold is reached, the FTF (FIFO threshold flag) is set. In indirect read mode, FTF is set when the number of valid bytes to be read from the FIFO is above the threshold. FTF is also set if there are data in the FIFO after the last byte is read from the Flash memory, regardless of the FTHRES setting. In indirect write mode, FTF is set when the number of empty bytes in the FIFO is above the threshold.

If FTIE = 1, there is an interrupt when FTF is set. If DMAEN = 1, a DMA transfer is initiated when FTF is set. FTF is cleared by HW as soon as the threshold condition is no longer true (after enough data has been transferred by the CPU or DMA).

In indirect read mode, when the FIFO becomes full, the QUADSPI temporarily stops reading bytes from the Flash memory to avoid an overrun. Note that the reading of the Flash memory does not restart until 4 bytes become vacant in the FIFO (when FLEVEL ≤ 28). Thus, when FTHRES ≥ 29, the application must take care to read enough bytes to assure that the QUADSPI starts retrieving data from the Flash memory again. Otherwise, the FTF flag stays at '0' as long as 28 < FLEVEL < FTHRES.

13.3.6 QUADSPI status flag polling mode

In automatic-polling mode, the QUADSPI periodically starts a command to read a defined number of status bytes (up to 4). The received bytes can be masked to isolate some status bits and an interrupt can be generated when the selected bits have a defined value.

The accesses to the Flash memory begin in the same way as in indirect read mode: if no address is required (AMODE = 00), accesses begin as soon as the QUADSPI_CCR is written. Otherwise, if an address is required, the first access begins when QUADSPI_AR is written. BUSY goes high at this point and stays high even between the periodic accesses.

The contents of MASK[31:0] (QUADSPI_PSMAR) are used to mask the data from the Flash memory in automatic-polling mode. If the MASK[n] = 0, then bit n of the result is masked and not considered. If MASK[n] = 1, and the content of bit[n] is the same as MATCH[n] (QUADSPI_PSMAR), then there is a match for bit n.

If the polling match mode bit (PMM, bit 23 of QUADSPI_CR) is 0, then “AND” match mode is activated. This means status match flag (SMF) is set only when there is a match on all of the unmasked bits.

If PMM = 1, then “OR” match mode is activated. This means SMF is set if there is a match on any of the unmasked bits.

An interrupt is called when SMF is set if SMIE = 1.

If the automatic-polling-mode-stop (APMS) bit is set, operation stops and BUSY goes to 0 as soon as a match is detected. Otherwise, BUSY stays at ‘1’ and the periodic accesses continue until there is an abort or the QUADSPI is disabled (EN = 0).

The data register (QUADSPI_DR) contains the latest received status bytes (the FIFO is deactivated). The content of the data register is not affected by the masking used in the matching logic. The FTF status bit is set as soon as a new reading of the status is complete, and FTF is cleared as soon as the data is read.

13.3.7 QUADSPI memory-mapped mode

When configured in memory-mapped mode, the external SPI device is seen as an internal memory.

It is forbidden to access QUADSPI Flash bank area before having properly configured and enabled the QUADSPI peripheral.

No more than 256MB can be addressed even if the Flash memory capacity is larger.

If an access is made to an address outside of the range defined by FSIZE but still within the 256MB range, then a bus error is given. The effect of this error depends on the bus master that attempted the access:

- If it is the Cortex® CPU, bus fault exception is generated when enabled (or a hard fault exception when bus fault is disabled)
- If it is a DMA, a DMA transfer error is generated and the corresponding DMA channel is automatically disabled.

Byte, halfword, and word access types are all supported.

Support for execute in place (XIP) operation is implemented, where the QUADSPI anticipates the next microcontroller access and load in advance the byte at the following address. If the subsequent access is indeed made at a continuous address, the access will be completed faster since the value is already prefetched.

By default, the QUADSPI never stops its prefetch operation, keeping the previous read operation active with nCS maintained low, even if no access to the Flash memory occurs for a long time. Since Flash memories tend to consume more when nCS is held low, the application might want to activate the timeout counter (TCEN = 1, bit 3 of QUADSPI_CR) so that nCS is released after a period of TIMEOUT[15:0] (QUADSPI_LPTR) cycles have elapsed without any access since when the FIFO becomes full with prefetch data.

BUSY goes high as soon as the first memory-mapped access occurs. Because of the prefetch operations, BUSY does not fall until there is a timeout, there is an abort, or the peripheral is disabled.

13.3.8 QUADSPI Flash memory configuration

The device configuration register (QUADSPI_DCR) can be used to specify the characteristics of the external SPI Flash memory.

The FSIZE[4:0] field defines the size of external memory using the following formula:

$$\text{Number of bytes in Flash memory} = 2^{\text{FSIZE}+1}$$

FSIZE+1 is effectively the number of address bits required to address the Flash memory. The Flash memory capacity can be up to 4GB (addressed using 32 bits) in indirect mode, but the addressable space in memory-mapped mode is limited to 256MB.

If DFM = 1, FSIZE indicates the total capacity of the two Flash memories together.

When the QUADSPI executes two commands, one immediately after the other, it raises the chip select signal (nCS) high between the two commands for only one CLK cycle by default. If the Flash memory requires more time between commands, the chip select high time (CSHT) field can be used to specify the minimum number of CLK cycles (up to 8) that nCS must remain high.

The clock mode (CKMODE) bit indicates the CLK signal logic level in between commands (when nCS = 1).

13.3.9 QUADSPI delayed data sampling

By default, the QUADSPI samples the data driven by the Flash memory one half of a CLK cycle after the Flash memory drives the signal.

In case of external signal delays, it may be beneficial to sample the data later. Using the SSHIFT bit (bit 4 of QUADSPI_CR), the sampling of the data can be shifted by half of a CLK cycle.

Clock shifting is not supported in DDR mode: the SSHIFT bit must be clear when DDRM bit is set.

13.3.10 QUADSPI configuration

The QUADSPI configuration is done in two phases:

- QUADSPI IP configuration
- QUADSPI Flash memory configuration

Once configured and enabled, the QUADSPI can be used in one of its three operating modes: indirect mode, status-polling mode, or memory-mapped mode.

QUADSPI IP configuration

The QUADSPI IP is configured using the QUADSPI_CR. The user shall configure the clock prescaler division factor and the sample shifting settings for the incoming data.

DDR mode can be set through the DDRM bit. Once enabled, the address and the alternate bytes are sent on both clock edges and the data are sent/received on both clock edges. Regardless of the DDRM bit setting, instructions are always sent in SDR mode.

The DMA requests are enabled setting the DMAEN bit. In case of interrupt usage, their respective enable bit can be also set during this phase.

FIFO level for either DMA request generation or interrupt generation is programmed in the FTHRES bits.

If timeout counter is needed, the TCEN bit can be set and the timeout value programmed in the QUADSPI_LPTR register.

Dual-flash mode can be activated by setting DFM to 1.

QUADSPI Flash memory configuration

The parameters related to the targeted external Flash memory are configured through the QUADSPI_DCR register. The user shall program the Flash memory size in the FSIZE bits, the Chip Select minimum high time in the CSHT bits, and the functional mode (Mode 0 or Mode 3) in the MODE bit.

13.3.11 QUADSPI usage

The operating mode is selected using FMODE[1:0] (QUADSPI_CCR[27:26]).

Indirect mode procedure

When FMODE is programmed to 00, indirect write mode is selected and data can be sent to the Flash memory. With FMODE = 01, indirect read mode is selected where data can be read from the Flash memory.

When the QUADSPI is used in indirect mode, the frames are constructed in the following way:

1. Specify a number of data bytes to read or write in the QUADSPI_DLR.
2. Specify the frame format, mode and instruction code in the QUADSPI_CCR.
3. Specify optional alternate byte to be sent right after the address phase in the QUADSPI_ABR.
4. Specify the operating mode in the QUADSPI_CR. If FMODE = 00 (indirect write mode) and DMAEN = 1, then QUADSPI_AR should be specified before QUADSPI_CR, because otherwise QUADSPI_DR might be written by the DMA before QUADSPI_AR is updated (if the DMA controller has already been enabled)
5. Specify the targeted address in the QUADSPI_AR.
6. Read/Write the data from/to the FIFO through the QUADSPI_DR.

When writing the control register (QUADSPI_CR) the user specifies the following settings:

- The enable bit (EN) set to '1'
- The DMA enable bit (DMAEN) for transferring data to/from RAM
- Timeout counter enable bit (TCEN)
- Sample shift setting (SSSHIFT)
- FIFO threshold level (FTRHES) to indicate when the FTF flag should be set
- Interrupt enables
- Automatic polling mode parameters: match mode and stop mode (valid when FMODE = 11)
- Clock prescaler

When writing the communication configuration register (QUADSPI_CCR) the user specifies the following parameters:

- The instruction byte through the INSTRUCTION bits
- The way the instruction has to be sent through the IMODE bits (1/2/4 lines)
- The way the address has to be sent through the ADMODE bits (None/1/2/4 lines)
- The address size (8/16/24/32-bit) through the ADSIZE bits
- The way the alternate bytes have to be sent through the ABMODE (None/1/2/4 lines)
- The alternate bytes number (1/2/3/4) through the ABSIZE bits
- The presence or not of dummy bytes through the DBMODE bit
- The number of dummy bytes through the DCYC bits
- The way the data have to be sent/received (None/1/2/4 lines) through the DMODE bits

If neither the address register (QUADSPI_AR) nor the data register (QUADSPI_DR) need to be updated for a particular command, then the command sequence starts as soon as QUADSPI_CCR is written. This is the case when both ADMODE and DMODE are 00, or if just ADMODE = 00 when in indirect read mode (FMODE = 01).

When an address is required (ADMODE is not 00) and the data register does not need to be written (when FMODE = 01 or DMODE = 00), the command sequence starts as soon as the address is updated with a write to QUADSPI_AR.

In case of data transmission (FMODE = 00 and DMODE! = 00), the communication start is triggered by a write in the FIFO through QUADSPI_DR.

Status flag polling mode

The status flag polling mode is enabled setting the FMODE field (QUADSPI_CCR[27:26]) to 10. In this mode, the programmed frame will be sent and the data retrieved periodically.

The maximum amount of data read in each frame is 4 bytes. If more data is requested in QUADSPI_DLR, it will be ignored and only 4 bytes will be read.

The periodicity is specified in the QUADSPI_PISR register.

Once the status data has been retrieved, it can internally be processed in order to:

- set the status match flag and generate an interrupt if enabled
- stop automatically the periodic retrieving of the status bytes

The received value can be masked with the value stored in the QUADSPI_PSMKR and ORed or ANDed with the value stored in the QUADSPI_PSMAR.

In case of match, the status match flag is set and an interrupt is generated if enabled, and the QUADSPI can be automatically stopped if the AMPS bit is set.

In any case, the latest retrieved value is available in the QUADSPI_DR.

Memory-mapped mode

In memory-mapped mode, the external Flash memory is seen as internal memory but with some latency during accesses. Only read operations are allowed to the external Flash memory in this mode.

Memory-mapped mode is entered by setting the FMODE to 11 in the QUADSPI_CCR register.

The programmed instruction and frame is sent when a master is accessing the memory mapped space.

The FIFO is used as a prefetch buffer to anticipate linear reads. Any access to QUADSPI_DR in this mode returns zero.

The data length register (QUADSPI_DLR) has no meaning in memory-mapped mode.

13.3.12 Sending the instruction only once

Some Flash memories (e.g. Winbond) might provide a mode where an instruction must be sent only with the first command sequence, while subsequent commands start directly with the address. One can take advantage of such a feature using the SIOO bit (QUADSPI_CCR[28]).

SIOO is valid for all functional modes (indirect, automatic polling, and memory-mapped). If the SIOO bit is set, the instruction is sent only for the first command following a write to QUADSPI_CCR. Subsequent command sequences skip the instruction phase, until there is a write to QUADSPI_CCR.

SIOO has no effect when IMODE = 00 (no instruction).

13.3.13 QUADSPI error management

An error can be generated in the following case:

- In indirect mode or status flag polling mode when a wrong address has been programmed in the QUADSPI_AR (according to the Flash memory size defined by FSIZE[4:0] in the QUADSPI_DCR): this will set the TEF and an interrupt is generated if enabled.
- Also in indirect mode, if the address plus the data length exceeds the Flash memory size, TEF will be set as soon as the access is triggered.
- In memory-mapped mode, when an out of range access is done by a master or when the QUADSPI is disabled: this will generate a bus error as a response to the faulty bus master request.
- When a master is accessing the memory mapped space while the memory mapped mode is disabled: this will generate a bus error as a response to the faulty bus master request.

13.3.14 QUADSPI busy bit and abort functionality

Once the QUADSPI starts an operation with the Flash memory, the BUSY bit is automatically set in the QUADSPI_SR.

In indirect mode, the BUSY bit is reset once the QUADSPI has completed the requested command sequence and the FIFO is empty.

In automatic-polling mode, BUSY goes low only after the last periodic access is complete, due to a match when APMS = 1, or due to an abort.

After the first access in memory-mapped mode, BUSY goes low only on a timeout event or on an abort.

Any operation can be aborted by setting the ABORT bit in the QUADSPI_CR. Once the abort is completed, the BUSY bit and the ABORT bit are automatically reset, and the FIFO is flushed.

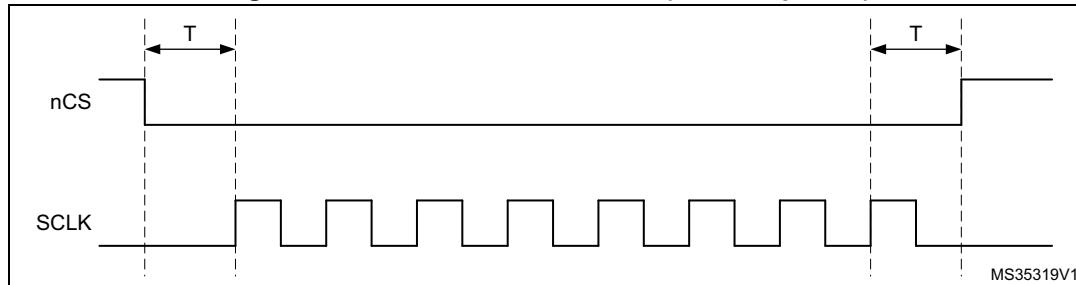
Note: Some Flash memories might misbehave if a write operation to a status registers is aborted.

13.3.15 nCS behavior

By default, nCS is high, deselecting the external Flash memory. nCS falls before an operation begins and rises as soon as it finishes.

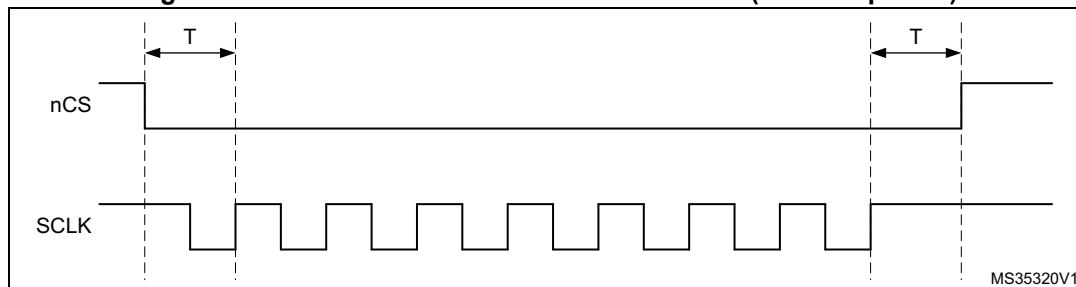
When CKMODE = 0 (“mode0”, where CLK stays low when no operation is in progress) nCS falls one CLK cycle before an operation first rising CLK edge, and nCS rises one CLK cycle after the operation final rising CLK edge, as shown in [Figure 66](#).

Figure 66. nCS when CKMODE = 0 (T = CLK period)



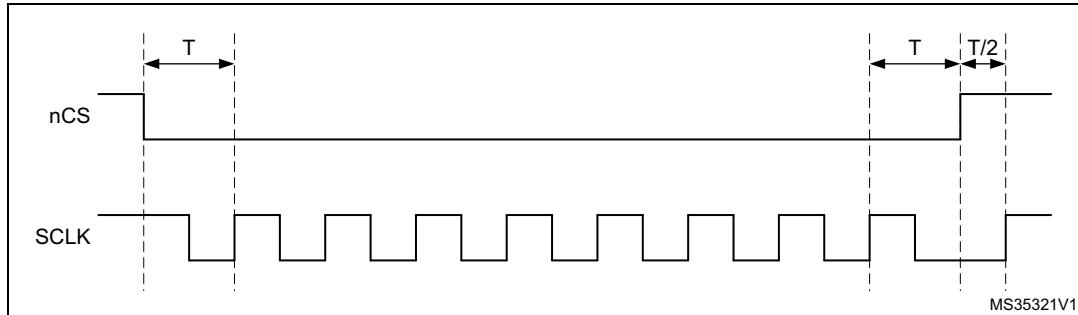
When CKMODE=1 (“mode3”, where CLK goes high when no operation is in progress) and DDRM=0 (SDR mode), nCS still falls one CLK cycle before an operation first rising CLK edge, and nCS rises one CLK cycle after the operation final rising CLK edge, as shown in [Figure 67](#).

Figure 67. nCS when CKMODE = 1 in SDR mode (T = CLK period)



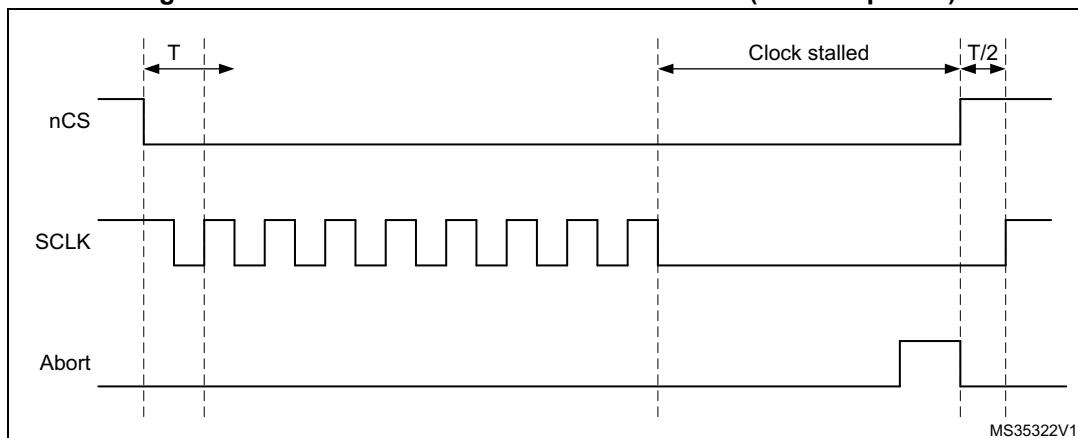
When CKMODE = 1 (“mode3”) and DDRM = 1 (DDR mode), nCS falls one CLK cycle before an operation first rising CLK edge, and nCS rises one CLK cycle after the operation final active rising CLK edge, as shown in [Figure 68](#). Because DDR operations must finish with a falling edge, CLK is low when nCS rises, and CLK rises back up one half of a CLK cycle afterwards.

Figure 68. nCS when CKMODE = 1 in DDR mode (T = CLK period)



When the FIFO stays full in a read operation or if the FIFO stays empty in a write operation, the operation stalls and CLK stays low until firmware services the FIFO. If an abort occurs when an operation is stalled, nCS rises just after the abort is requested and then CLK rises one half of a CLK cycle later, as shown in [Figure 69](#).

Figure 69. nCS when CKMODE = 1 with an abort (T = CLK period)



When not in dual-flash mode (DFM = 0), only FLASH 1 is accessed and thus the BK2_nCS stays high. In dual-flash mode, BK2_nCS behaves exactly the same as BK1_nCS. Thus, if there is a FLASH 2 and if the application always stays in dual-flash mode, then FLASH 2 may use BK1_nCS and the pin outputting BK2_nCS can be used for other functions.

13.4 QUADSPI interrupts

An interrupt can be produced on the following events:

- Timeout
- Status match
- FIFO threshold
- Transfer complete
- Transfer error

Separate interrupt enable bits are available for flexibility.

Table 83. QUADSPI interrupt requests

Interrupt event	Event flag	Enable control bit
Timeout	TOF	TOIE
Status match	SMF	SMIE
FIFO threshold	FTF	FTIE
Transfer complete	TCF	TCIE
Transfer error	TEF	TEIE

13.5 QUADSPI registers

13.5.1 QUADSPI control register (QUADSPI_CR)

Address offset: 0x0000

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PRESCALER[7:0]								PMM	APMS	Res.	TOIE	SMIE	FTIE	TCIE	TEIE
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	FTHRES[4:0]				FSEL	DFM	Res.	SSSHIFT	TCEN	DMAEN	ABORT	EN	
			rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw

Bits 31:24 **PRESCALER[7:0]**: Clock prescaler

This field defines the scaler factor for generating CLK based on the AHB clock (value+1).

0: $F_{CLK} = F_{AHB}$, AHB clock used directly as QUADSPI CLK (prescaler bypassed)

1: $F_{CLK} = F_{AHB}/2$

2: $F_{CLK} = F_{AHB}/3$

...

255: $F_{CLK} = F_{AHB}/256$

For odd clock division factors, CLK's duty cycle is not 50%. The clock signal remains low one cycle longer than it stays high.

This field can be modified only when BUSY = 0.

Bit 23 **PMM**: Polling match mode

This bit indicates which method should be used for determining a "match" during automatic polling mode.

0: AND match mode. SMF is set if all the unmasked bits received from the Flash memory match the corresponding bits in the match register.

1: OR match mode. SMF is set if any one of the unmasked bits received from the Flash memory matches its corresponding bit in the match register.

This bit can be modified only when BUSY = 0.

Bit 22 **APMS**: Automatic poll mode stop

This bit determines if automatic polling is stopped after a match.

0: Automatic polling mode is stopped only by abort or by disabling the QUADSPI.

1: Automatic polling mode stops as soon as there is a match.

This bit can be modified only when BUSY = 0.

Bit 21 Reserved, must be kept at reset value.

Bit 20 **TOIE**: TimeOut interrupt enable

This bit enables the TimeOut interrupt.

0: Interrupt disable

1: Interrupt enabled

Bit 19 **SMIE**: Status match interrupt enable

This bit enables the status match interrupt.

0: Interrupt disable

1: Interrupt enabled

Bit 18 **FTIE**: FIFO threshold interrupt enable

This bit enables the FIFO threshold interrupt.

0: Interrupt disabled

1: Interrupt enabled

Bit 17 **TCIE**: Transfer complete interrupt enable

This bit enables the transfer complete interrupt.

0: Interrupt disabled

1: Interrupt enabled

Bit 16 **TEIE**: Transfer error interrupt enable

This bit enables the transfer error interrupt.

0: Interrupt disable

1: Interrupt enabled

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

Bits 12:8 **FTHRES[4:0]** FIFO threshold level

Defines, in indirect mode, the threshold number of bytes in the FIFO that will cause the FIFO threshold flag (FTF, QUADSPI_SR[2]) to be set.

In indirect write mode (FMODE = 00):

0: FTF is set if there are 1 or more free bytes available to be written to in the FIFO

1: FTF is set if there are 2 or more free bytes available to be written to in the FIFO

...

31: FTF is set if there are 32 free bytes available to be written to in the FIFO

In indirect read mode (FMODE = 01):

0: FTF is set if there are 1 or more valid bytes that can be read from the FIFO

1: FTF is set if there are 2 or more valid bytes that can be read from the FIFO

...

31: FTF is set if there are 32 valid bytes that can be read from the FIFO

If DMAEN = 1, then the DMA controller for the corresponding channel must be disabled before changing the FTHRES value.

Bit 7 **FSEL**: Flash memory selection

This bit selects the Flash memory to be addressed in single flash mode (when DFM = 0).

0: FLASH 1 selected

1: FLASH 2 selected

This bit can be modified only when BUSY = 0.

This bit is ignored when DFM = 1.

Bit 6 **DFM**: Dual-flash mode

This bit activates dual-flash mode, where two external Flash memories are used simultaneously to double throughput and capacity.

0: Dual-flash mode disabled

1: Dual-flash mode enabled

This bit can be modified only when BUSY = 0.

Bit 5 Reserved, must be kept at reset value.

Bit 4 SSHIFT: Sample shift

By default, the QUADSPI samples data 1/2 of a CLK cycle after the data is driven by the Flash memory. This bit allows the data to be sampled later in order to account for external signal delays.

0: No shift

1: 1/2 cycle shift

Firmware must assure that **SSSHIFT = 0** when in DDR mode (when **DDRM = 1**).

This field can be modified only when **BUSY = 0**.

Bit 3 TCEN: Timeout counter enable

This bit is valid only when memory-mapped mode (**FMODE = 11**) is selected. Activating this bit causes the chip select (nCS) to be released (and thus reduces consumption) if there has not been an access after a certain amount of time, where this time is defined by **TIMEOUT[15:0]** (**QUADSPI_LPTR**).

Enable the timeout counter.

By default, the QUADSPI never stops its prefetch operation, keeping the previous read operation active with nCS maintained low, even if no access to the Flash memory occurs for a long time. Since Flash memories tend to consume more when nCS is held low, the application might want to activate the timeout counter (**TCEN = 1**, bit 3 of **QUADSPI_CR**) so that nCS is released after a period of **TIMEOUT[15:0]** (**QUADSPI_LPTR**) cycles have elapsed without an access since when the FIFO becomes full with prefetch data.

0: Timeout counter is disabled, and thus the chip select (nCS) remains active indefinitely after an access in memory-mapped mode.

1: Timeout counter is enabled, and thus the chip select is released in memory-mapped mode after **TIMEOUT[15:0]** cycles of Flash memory inactivity.

This bit can be modified only when **BUSY = 0**.

Bit 2 DMAEN: DMA enable

In indirect mode, DMA can be used to input or output data via the **QUADSPI_DR** register. DMA transfers are initiated when the FIFO threshold flag, **FTF**, is set.

0: DMA is disabled for indirect mode

1: DMA is enabled for indirect mode

Bit 1 ABORT: Abort request

This bit aborts the on-going command sequence. It is automatically reset once the abort is complete.

This bit stops the current transfer.

In polling mode or memory-mapped mode, this bit also reset the **APM** bit or the **DM** bit.

0: No abort requested

1: Abort requested

Bit 0 EN: Enable

Enable the QUADSPI.

0: QUADSPI is disabled

1: QUADSPI is enabled

13.5.2 QUADSPI device configuration register (QUADSPI_DCR)

Address offset: 0x0004

Reset value: 0x0000 0000

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

Bits 31:21 Reserved, must be kept at reset value.

Bits 20:16 **FSIZE[4:0]**: Flash memory size

This field defines the size of external memory using the following formula:

$$\text{Number of bytes in Flash memory} = 2^{\text{FSIZE}+1}$$

FSIZE+1 is effectively the number of address bits required to address the Flash memory. The Flash memory capacity can be up to 4GB (addressed using 32 bits) in indirect mode, but the addressable space in memory-mapped mode is limited to 256MB.

If DFM = 1, FSIZE indicates the total capacity of the two Flash memories together.
This field can be modified only when BUSY = 0.

Bits 15:11 Reserved, must be kept at reset value.

Bits 10:8 **CSHT[2:0]**: Chip select high time

CSHT+1 defines the minimum number of CLK cycles which the chip select (nCS) must remain high between commands issued to the Flash memory.

0: nCS stays high for at least 1 cycle between Flash memory commands

1: nCS stays high for at least 2 cycles between Flash memory commands

...

7: nCS stays high for at least 8 cycles between Flash memory commands

This field can be modified only when BUSY = 0.

Bits 7:1 Reserved, must be kept at reset value.

Bit 0 **CKMODE**: Mode 0 / mode 3

This bit indicates the level that CLK takes between commands (when nCS = 1).

0: CLK must stay low while nCS is high (chip select released). This is referred to as mode 0.

1: CLK must stay high while nCS is high (chip select released). This is referred to as mode 3.

This field can be modified only when BUSY = 0.

13.5.3 QUADSPI status register (QUADSPI_SR)

Address offset: 0x0008

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	Res.	FLEVEL[5:0]							Res.	Res.	BUSY	TOF	SMF	FTF	TCF	TEF
		r	r	r	r	r	r			r	r	r	r	r	r	

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

Bits 13:8 **FLEVEL[5:0]**: FIFO level

This field gives the number of valid bytes which are being held in the FIFO. FLEVEL = 0 when the FIFO is empty, and 32 when it is full. In memory-mapped mode and in automatic status polling mode, FLEVEL is zero.

Bits 7:6 Reserved, must be kept at reset value.

Bit 5 **BUSY**: Busy

This bit is set when an operation is on going. This bit clears automatically when the operation with the Flash memory is finished and the FIFO is empty.

Bit 4 **TOF**: Timeout flag

This bit is set when timeout occurs. It is cleared by writing 1 to CTOF.

Bit 3 **SMF**: Status match flag

This bit is set in automatic polling mode when the unmasked received data matches the corresponding bits in the match register (QUADSPI_PSMAR). It is cleared by writing 1 to CSMF.

Bit 2 **FTF**: FIFO threshold flag

In indirect mode, this bit is set when the FIFO threshold has been reached, or if there is any data left in the FIFO after reads from the Flash memory are complete. It is cleared automatically as soon as threshold condition is no longer true.

In automatic polling mode this bit is set every time the status register is read, and the bit is cleared when the data register is read.

Bit 1 **TCF**: Transfer complete flag

This bit is set in indirect mode when the programmed number of data has been transferred or in any mode when the transfer has been aborted. It is cleared by writing 1 to CTCF.

Bit 0 **TEF**: Transfer error flag

This bit is set in indirect mode when an invalid address is being accessed in indirect mode. It is cleared by writing 1 to CTEF.

13.5.4 QUADSPI flag clear register (QUADSPI_FCR)

Address offset: 0x000C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CTOF	CSMF	Res.	CTCF	CTEF										
										w	w		w	w	

Bits 31:5 Reserved, must be kept at reset value.

Bit 4 **CTOF**: Clear timeout flag

Writing 1 clears the TOF flag in the QUADSPI_SR register

Bit 3 **CSMF**: Clear status match flag

Writing 1 clears the SMF flag in the QUADSPI_SR register

Bit 2 Reserved, must be kept at reset value.

Bit 1 **CTCF**: Clear transfer complete flag

Writing 1 clears the TCF flag in the QUADSPI_SR register

Bit 0 **CTEF**: Clear transfer error flag

Writing 1 clears the TEF flag in the QUADSPI_SR register

13.5.5 QUADSPI data length register (QUADSPI_DLR)

Address offset: 0x0010

Reset value: 0x0000 0000

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

Bits 31:0 **DL[31:0]**: Data length

Number of data to be retrieved (value+1) in indirect and status-polling modes. A value no greater than 3 (indicating 4 bytes) should be used for status-polling mode.

All 1s in indirect mode means undefined length, where QUADSPI will continue until the end of memory, as defined by FSIZE.

0x0000_0000: 1 byte is to be transferred

0x0000_0001: 2 bytes are to be transferred

0x0000_0002: 3 bytes are to be transferred

0x0000_0003: 4 bytes are to be transferred

...

0xFFFF_FFFD: 4,294,967,294 (4G-2) bytes are to be transferred

0xFFFF_FFFE: 4,294,967,295 (4G-1) bytes are to be transferred

0xFFFF_FFFF: undefined length -- all bytes until the end of Flash memory (as defined by FSIZE) are to be transferred. Continue reading indefinitely if FSIZE = 0x1F.

DL[0] is stuck at '1' in dual-flash mode (DFM = 1) even when '0' is written to this bit, thus assuring that each access transfers an even number of bytes.

This field has no effect when in memory-mapped mode (FMODE = 10).

This field can be written only when BUSY = 0.

13.5.6 QUADSPI communication configuration register (QUADSPI_CCR)

Address offset: 0x0014

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DDRM	DHHC	Res.	SIOO	FMODE[1:0]	DMODE[1:0]	Res.	DCYC[4:0]				ABSIZE[1:0]				
rw	rw		rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABMODE[1:0]		ADSIZE[1:0]		ADMODE[1:0]		IMODE[1:0]		INSTRUCTION[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **DDRM**: Double data rate mode

This bit sets the DDR mode for the address, alternate byte and data phase:

0: DDR Mode disabled

1: DDR Mode enabled

This field can be written only when BUSY = 0.

Bit 30 **DHHC**: DDR hold

Delay the data output by 1/4 of the QUADSPI output clock cycle in DDR mode:

0: Delay the data output using analog delay

1: Delay the data output by 1/4 of a QUADSPI output clock cycle.

This feature is only active in DDR mode.

This field can be written only when BUSY = 0.

Bit 29 Reserved, must be kept at reset value.

Bit 28 **SIOO**: Send instruction only once mode

See [Section 13.3.12: Sending the instruction only once on page 359](#). This bit has no effect when IMODE = 00.

0: Send instruction on every transaction

1: Send instruction only for the first command

This field can be written only when BUSY = 0.

Bits 27:26 **FMODE[1:0]**: Functional mode

This field defines the QUADSPI functional mode of operation.

00: Indirect write mode

01: Indirect read mode

10: Automatic polling mode

11: Memory-mapped mode

If DMAEN = 1 already, then the DMA controller for the corresponding channel must be disabled before changing the FMODE value.

This field can be written only when BUSY = 0.

Bits 25:24 **D MODE[1:0]**: Data mode

This field defines the data phase's mode of operation:

00: No data

01: Data on a single line

10: Data on two lines

11: Data on four lines

This field also determines the dummy phase mode of operation.

This field can be written only when BUSY = 0.

Bit 23 Reserved, must be kept at reset value.

Bits 22:18 **DCYC[4:0]**: Number of dummy cycles

This field defines the duration of the dummy phase. In both SDR and DDR modes, it specifies a number of CLK cycles (0-31).

This field can be written only when BUSY = 0.

Bits 17:16 **ABSIZE[1:0]**: Alternate bytes size

This bit defines alternate bytes size:

00: 8-bit alternate byte

01: 16-bit alternate bytes

10: 24-bit alternate bytes

11: 32-bit alternate bytes

This field can be written only when BUSY = 0.

Bits 15:14 **ABMODE[1:0]**: Alternate bytes mode

This field defines the alternate-bytes phase mode of operation:

00: No alternate bytes

01: Alternate bytes on a single line

10: Alternate bytes on two lines

11: Alternate bytes on four lines

This field can be written only when BUSY = 0.

Bits 13:12 ADSIZE[1:0]: Address size

This bit defines address size:

- 00: 8-bit address
- 01: 16-bit address
- 10: 24-bit address
- 11: 32-bit address

This field can be written only when BUSY = 0.

Bits 11:10 ADMODE[1:0]: Address mode

This field defines the address phase mode of operation:

- 00: No address
- 01: Address on a single line
- 10: Address on two lines
- 11: Address on four lines

This field can be written only when BUSY = 0.

Bits 9:8 IMODE[1:0]: Instruction mode

This field defines the instruction phase mode of operation:

- 00: No instruction
- 01: Instruction on a single line
- 10: Instruction on two lines
- 11: Instruction on four lines

This field can be written only when BUSY = 0.

Bits 7:0 INSTRUCTION[7:0]: Instruction

Instruction to be send to the external SPI device.

This field can be written only when BUSY = 0.

13.5.7 QUADSPI address register (QUADSPI_AR)

Address offset: 0x0018

Reset value: 0x0000 0000

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

Bits 31:0 ADDRESS[31:0]: Address

Address to be send to the external Flash memory

Writes to this field are ignored when BUSY = 0 or when FMODE = 11 (memory-mapped mode).

In dual flash mode, ADDRESS[0] is automatically stuck to '0' as the address should always be even

13.5.8 QUADSPI alternate bytes registers (QUADSPI_ABR)

Address offset: 0x001C

Reset value: 0x0000 0000

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

Bits 31:0 **ALTERNATE[31:0]**: Alternate Bytes

Optional data to be send to the external SPI device right after the address.

This field can be written only when BUSY = 0.

13.5.9 QUADSPI data register (QUADSPI_DR)

Address offset: 0x0020

Reset value: 0x0000 0000

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

Bits 31:0 **DATA[31:0]**: Data

Data to be sent/received to/from the external SPI device.

In indirect write mode, data written to this register is stored on the FIFO before it is sent to the Flash memory during the data phase. If the FIFO is too full, a write operation is stalled until the FIFO has enough space to accept the amount of data being written.

In indirect read mode, reading this register gives (via the FIFO) the data which was received from the Flash memory. If the FIFO does not have as many bytes as requested by the read operation and if BUSY=1, the read operation is stalled until enough data is present or until the transfer is complete, whichever happens first.

In automatic polling mode, this register contains the last data read from the Flash memory (without masking).

Word, halfword, and byte accesses to this register are supported. In indirect write mode, a byte write adds 1 byte to the FIFO, a halfword write 2, and a word write 4. Similarly, in indirect read mode, a byte read removes 1 byte from the FIFO, a halfword read 2, and a word read 4. Accesses in indirect mode must be aligned to the bottom of this register: a byte read must read DATA[7:0] and a halfword read must read DATA[15:0].

13.5.10 QUADSPI polling status mask register (QUADSPI_PSMKR)

Address offset: 0x0024

Reset value: 0x0000 0000

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

Bits 31:0 **MASK[31:0]**: Status mask

Mask to be applied to the status bytes received in polling mode.

For bit n:

0: Bit n of the data received in automatic polling mode is masked and its value is not considered in the matching logic

1: Bit n of the data received in automatic polling mode is unmasked and its value is considered in the matching logic

This field can be written only when BUSY = 0.

13.5.11 QUADSPI polling status match register (QUADSPI_PSMAR)

Address offset: 0x0028

Reset value: 0x0000 0000

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

Bits 31:0 **MATCH[31:0]**: Status match

Value to be compared with the masked status register to get a match.

This field can be written only when BUSY = 0.

13.5.12 QUADSPI polling interval register (QUADSPI_PIR)

Address offset: 0x002C

Reset value: 0x0000 0000

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

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

Bits 15:0 **INTERVAL[15:0]**: Polling interval

Number of CLK cycles between to read during automatic polling phases.

This field can be written only when BUSY = 0.

13.5.13 QUADSPI low-power timeout register (QUADSPI_LPTR)

Address offset: 0x0030

Reset value: 0x0000 0000

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

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

Bits 15:0 **TIMEOUT[15:0]**: Timeout period

After each access in memory-mapped mode, the QUADSPI prefetches the subsequent bytes and holds these bytes in the FIFO. This field indicates how many CLK cycles the QUADSPI waits after the FIFO becomes full until it raises nCS, putting the Flash memory in a lower-consumption state.

This field can be written only when BUSY = 0.

13.5.14 QUADSPI register map

Table 84. QUADSPI register map and reset values

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6		
0x0000	QUADSPI_CR																												
		PRESCALER[7:0]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
0x0004	QUADSPI_DCR	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		Reset value	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
0x0008	QUADSPI_SR																												
		Reset value	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
0x000C	QUADSPI_FCR																												
		Reset value	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
0x0010	QUADSPI_DLR																DL[31:0]												
		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	
0x0014	QUADSPI_CCR		DDRM	DHHC	SIOO	FMODE[1:0]	DMODE[1:0]	DCYC[4:0]	ABSIZE[1:0]	ADSIZE[1:0]	ADMODE[1:0]	IMODE[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	
0x0018	QUADSPI_AR																ADDRESS[31:0]												
		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	
0x001C	QUADSPI_ABR																ALTERNATE[31:0]												
		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	
0x0020	QUADSPI_DR																DATA[31:0]												
		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	
0x0024	QUADSPI_PSMKR																MASK[31:0]												
		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	
0x0028	QUADSPI_PSMAR																MATCH[31:0]												
		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	
0x002C	QUADSPI_PIR																INTERVAL[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	
0x0030	QUADSPI_LPTR																TIMEOUT[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	

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

14 Analog-to-digital converter (ADC)

14.1 ADC introduction

The 12-bit ADC is a successive approximation analog-to-digital converter. It has up to 19 multiplexed channels allowing it to measure signals from 16 external sources, two internal sources, and the V_{BAT} channel. The A/D conversion of the channels can be performed in single, continuous, scan or discontinuous mode. The result of the ADC is stored into a left- or right-aligned 16-bit data register.

The analog watchdog feature allows the application to detect if the input voltage goes beyond the user-defined, higher or lower thresholds.

14.2 ADC main features

- 12-bit, 10-bit, 8-bit or 6-bit configurable resolution
- Interrupt generation at the end of conversion, end of injected conversion, and in case of analog watchdog or overrun events
- Single and continuous conversion modes
- Scan mode for automatic conversion of channel 0 to channel 'n'
- Data alignment with in-built data coherency
- Channel-wise programmable sampling time
- External trigger option with configurable polarity for both regular and injected conversions
- Discontinuous mode
- Dual/Triple mode (on devices with 2 ADCs or more)
- Configurable DMA data storage in Dual/Triple ADC mode
- Configurable delay between conversions in Dual/Triple interleaved mode
- ADC supply requirements: 2.4 V to 3.6 V at full speed and down to 1.8 V at slower speed
- ADC input range: $V_{REF-} \leq V_{IN} \leq V_{REF+}$
- DMA request generation during regular channel conversion

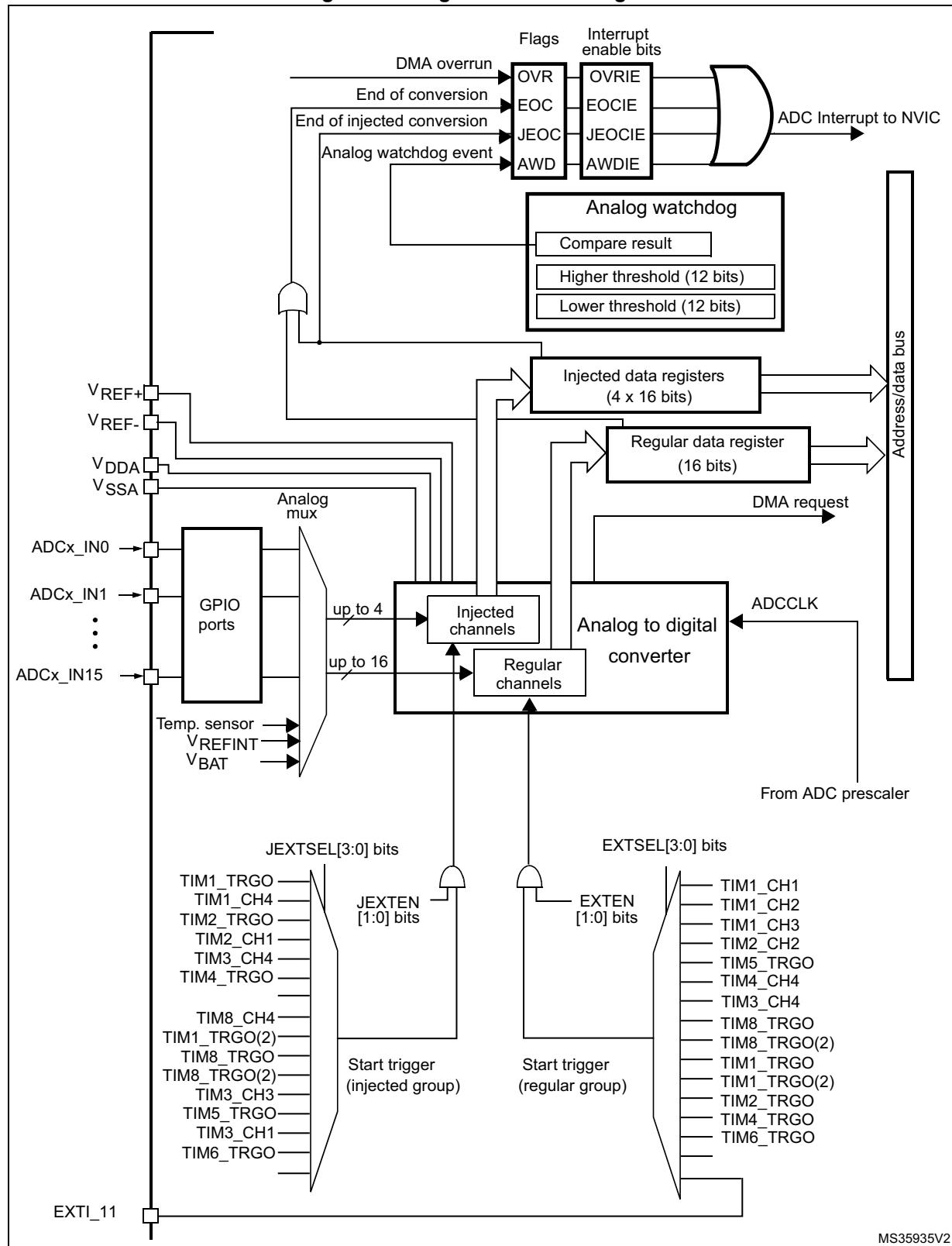
Figure 70 shows the block diagram of the ADC.

Note: V_{REF-} , if available (depending on package), must be tied to V_{SSA} .

14.3 ADC functional description

Figure 70 shows a single ADC block diagram and *Table 85* gives the ADC pin description.

Figure 70. Single ADC block diagram



MS35935V2

Table 85. ADC pins

Name	Signal type	Remarks
V_{REF+}	Input, analog reference positive	The higher/positive reference voltage for the ADC, $1.8 \text{ V} \leq V_{REF+} \leq V_{DDA}$
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) for full speed $1.8 \text{ V} \leq V_{DDA} \leq V_{DD}$ (3.6 V) for reduced speed
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

14.3.1 ADC on-off control

The ADC is powered on by setting the ADON bit in the ADC_CR2 register. When the ADON bit is set for the first time, it wakes up the ADC from the Power-down mode.

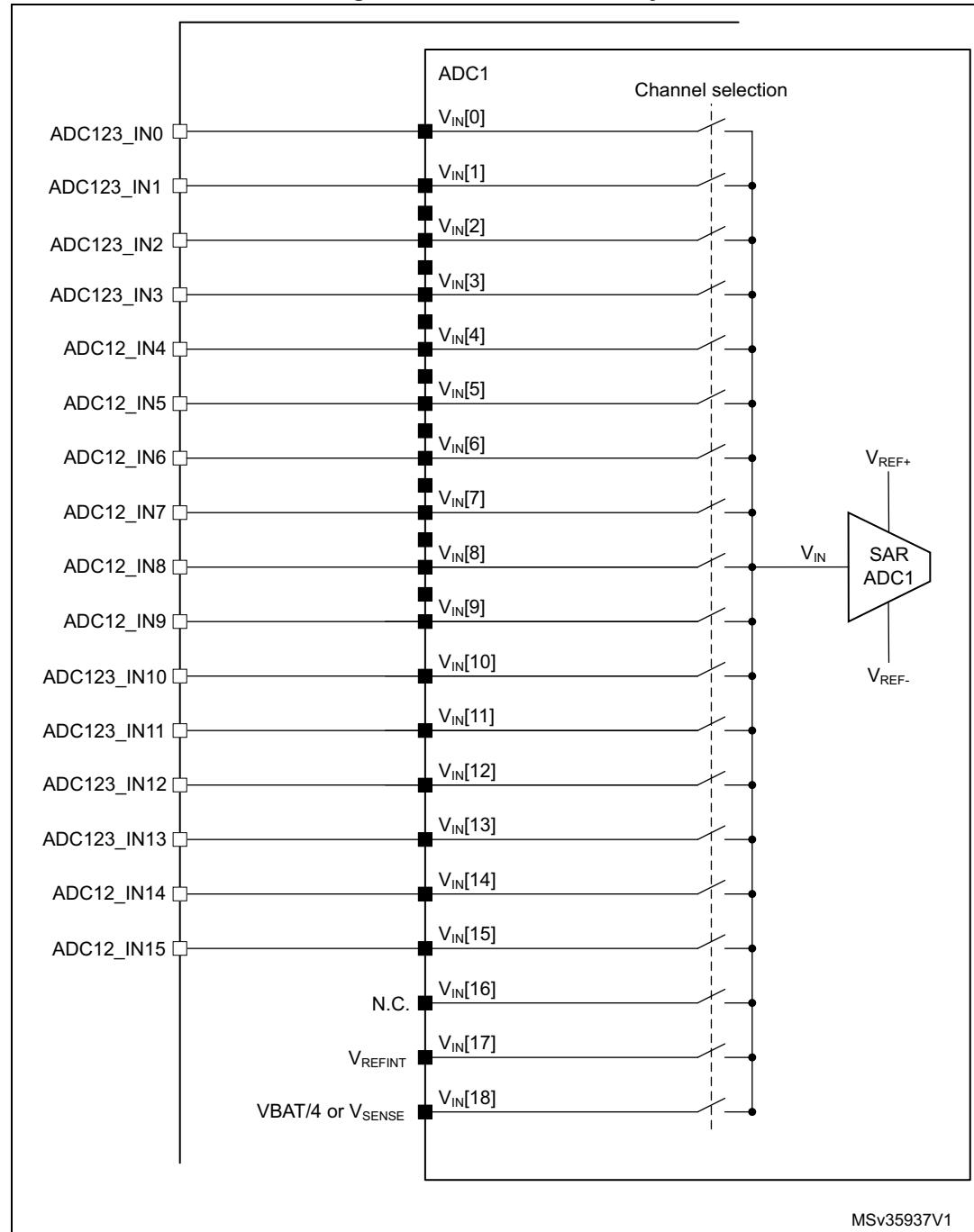
The conversion starts when either the SWSTART or the JSWSTART bit is set.

The user can stop conversion and put the ADC in power down mode by clearing the ADON bit. In this mode the ADC consumes almost no power (only a few μA).

14.3.2 ADC1/2 and ADC3 connectivity

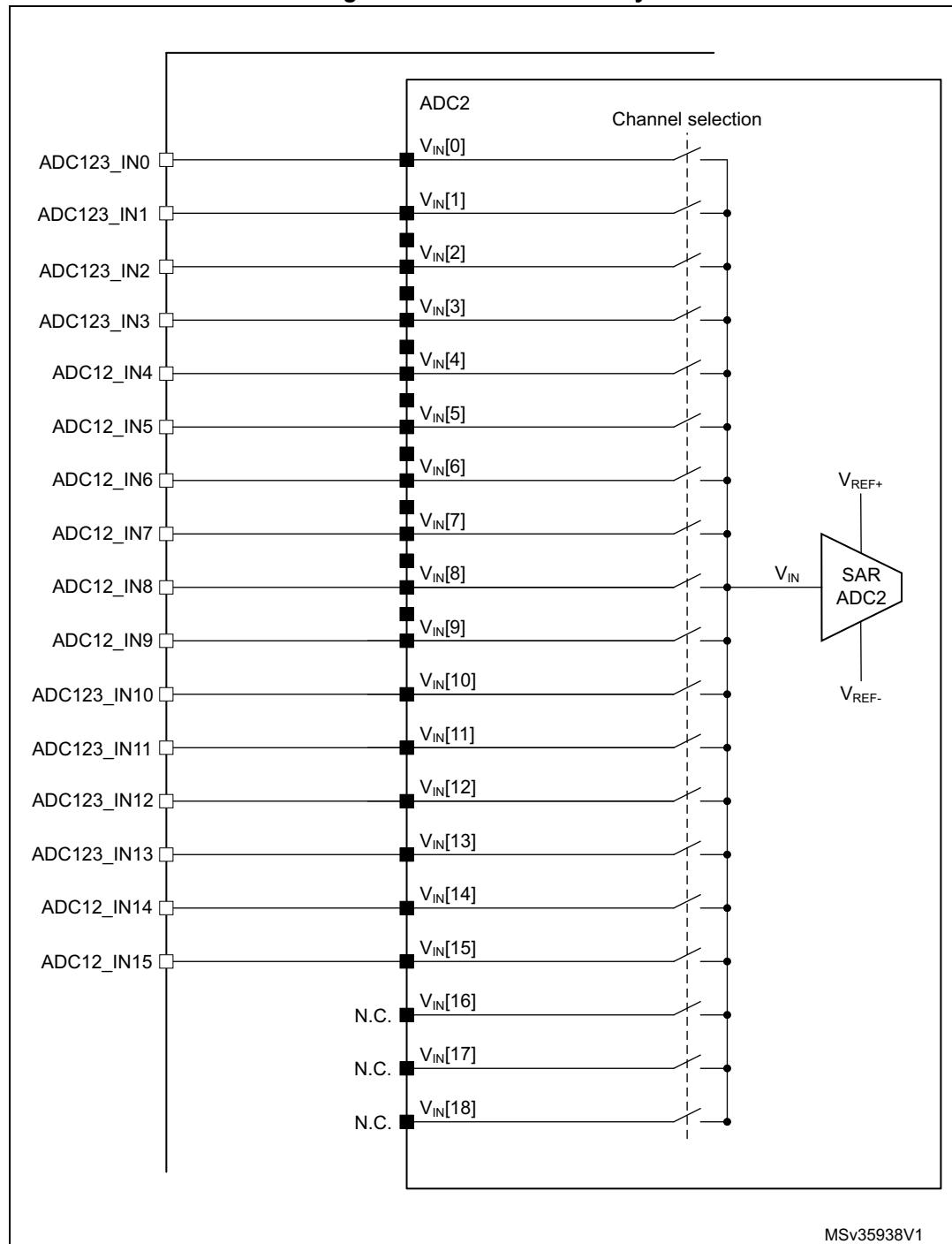
ADC1, ADC2 and ADC3 are tightly coupled and share some external channels as described in [Figure 71](#), [Figure 72](#) and [Figure 73](#).

Figure 71. ADC1 connectivity



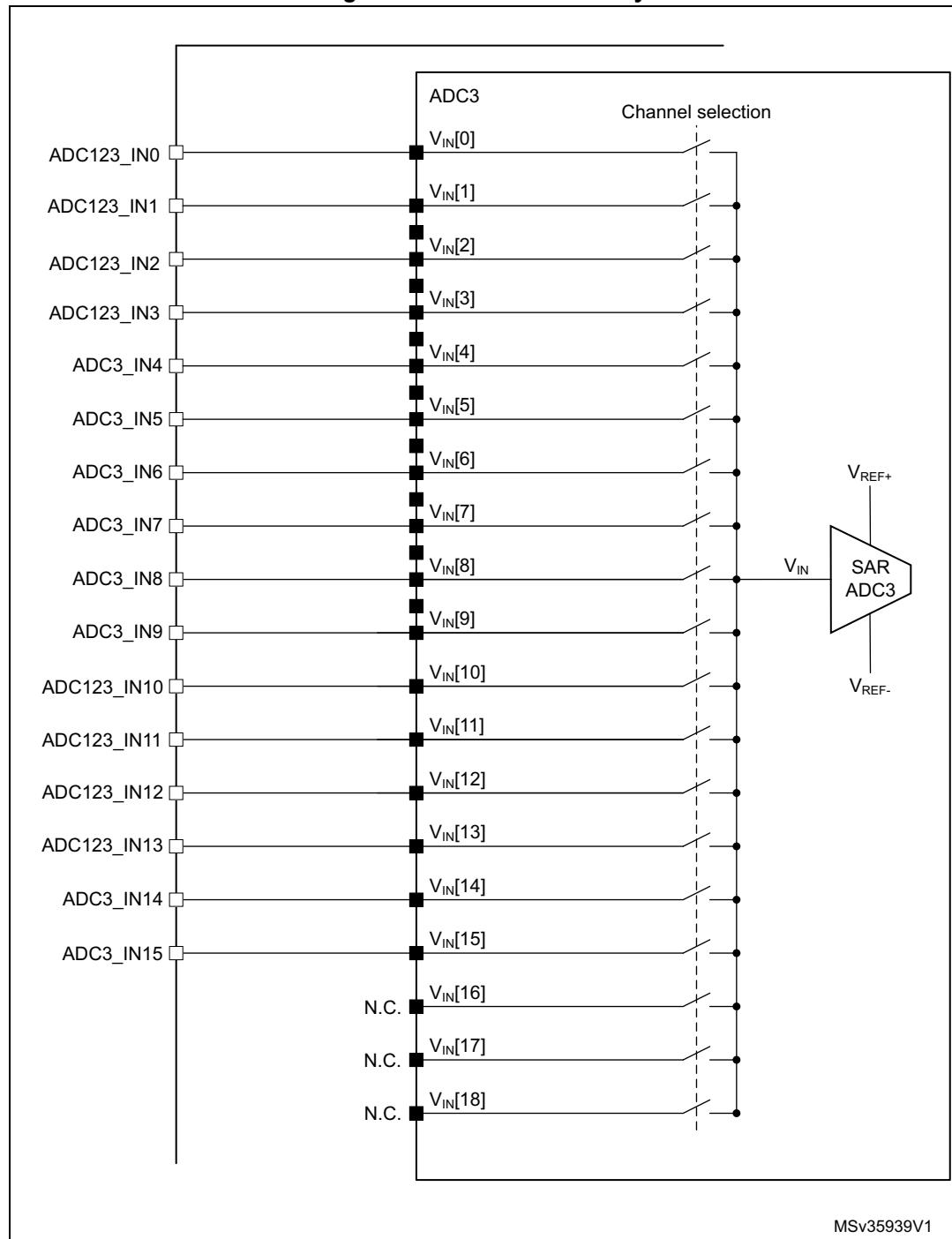
MSv35937V1

Figure 72. ADC2 connectivity



MSv35938V1

Figure 73. ADC3 connectivity



14.3.3 ADC clock

The ADC features two clock schemes:

- Clock for the analog circuitry: ADCCLK, common to all ADCs
This clock is generated from the APB2 clock divided by a programmable prescaler that allows the ADC to work at $f_{PCLK2}/2$, /4, /6 or /8. Refer to the datasheets for the maximum value of ADCCLK.
- Clock for the digital interface (used for registers read/write access)
This clock is equal to the APB2 clock. The digital interface clock can be enabled/disabled individually for each ADC through the RCC APB2 peripheral clock enable register (RCC_APB2ENR).

14.3.4 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 that can be done on any channel and in any order. For instance, it is possible to implement the conversion sequence in the following order: ADC_IN3, ADC_IN8, ADC_IN2, ADC_IN2, ADC_IN0, ADC_IN2, ADC_IN2, ADC_IN15.

- A **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.
- An **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 newly chosen group.

Temperature sensor, V_{REFINT} and V_{BAT} internal channels

- The temperature sensor is internally connected to ADC1_IN18 channel which is shared with VBAT. Only one conversion, temperature sensor or VBAT, must be selected at a time. When the temperature sensor and VBAT conversion are set simultaneously, only the VBAT conversion is performed.

The internal reference voltage VREFINT is connected to ADC1_IN17.

The V_{BAT} channel is connected to ADC1_IN18 channel. It can also be converted as an injected or regular channel.

Note: The temperature sensor, V_{REFINT} and the V_{BAT} channel are available only on the master ADC1 peripheral.

14.3.5 Single conversion mode

In Single conversion mode the ADC does one conversion. This mode is started with the CONT bit at 0 by either:

- setting the SWSTART bit in the ADC_CR2 register (for a regular channel only)
- setting the JSWSTART bit (for an injected channel)
- external trigger (for a regular or injected channel)

Once the conversion of the selected channel is complete:

- If a regular channel was converted:
 - The converted data are stored into the 16-bit ADC_DR register
 - The EOC (end of conversion) flag is set
 - An interrupt is generated if the EOCIE bit is set
- If an injected channel was converted:
 - The converted data are stored into the 16-bit ADC_JDR1 register
 - The JEOC (end of conversion injected) flag is set
 - An interrupt is generated if the JEOCIE bit is set

Then the ADC stops.

14.3.6 Continuous conversion mode

In continuous conversion mode, the ADC starts a new conversion as soon as it finishes one. This mode is started with the CONT bit at 1 either by external trigger or by setting the SWSTRT bit in the ADC_CR2 register (for regular channels only).

After each conversion:

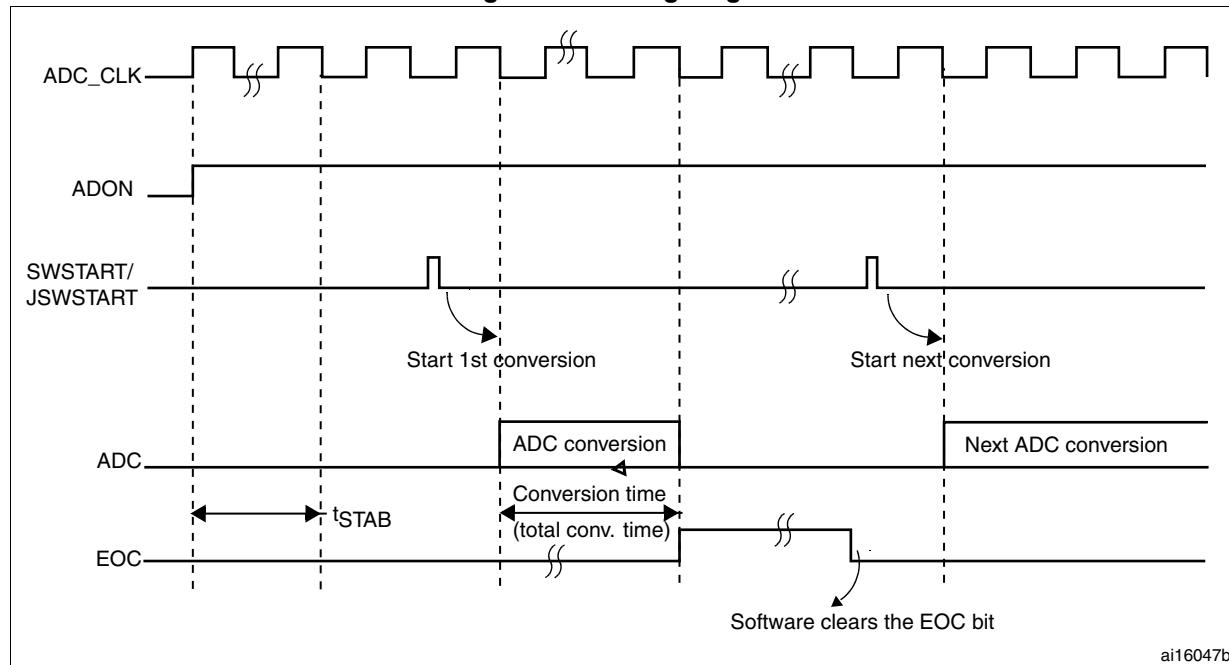
- If a regular group of channels was converted:
 - The last converted data are stored into the 16-bit ADC_DR register
 - The EOC (end of conversion) flag is set
 - An interrupt is generated if the EOCIE bit is set

Note: *Injected channels cannot be converted continuously. The only exception is when an injected channel is configured to be converted automatically after regular channels in continuous mode (using JAUTO bit), refer to [Auto-injection](#) section.*

14.3.7 Timing diagram

As shown in [Figure 74](#), the ADC needs a stabilization time of t_{STAB} before it starts converting accurately. After the start of the ADC conversion and after 15 clock cycles, the EOC flag is set and the 16-bit ADC data register contains the result of the conversion.

Figure 74. Timing diagram



14.3.8 Analog watchdog

The AWD analog watchdog status bit is set if the analog voltage converted by the ADC is below a lower threshold or above a higher threshold. These thresholds are programmed in 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 analog voltage is compared to the lower and higher thresholds before alignment.

[Table 86](#) shows how the ADC_CR1 register should be configured to enable the analog watchdog on one or more channels.

Figure 75. Analog watchdog's guarded area

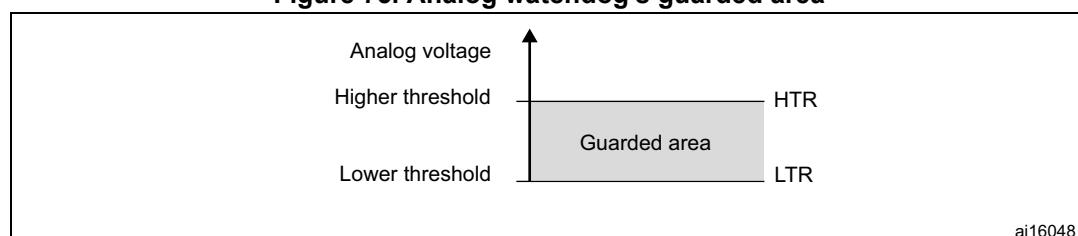


Table 86. Analog watchdog channel selection

Channels guarded by the 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

Table 86. Analog watchdog channel selection (continued)

Channels guarded by the analog watchdog	ADC_CR1 register control bits (x = don't care)		
	AWDSGL bit	AWDEN bit	JAWDEN bit
All regular channels	0	1	0
All regular and injected channels	0	1	1
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 the AWDCH[4:0] bits

14.3.9 Scan mode

This mode is used to scan a group of analog channels.

The Scan mode is selected by setting the SCAN bit in the ADC_CR1 register. Once this bit has been set, the ADC scans all the channels selected in the ADC_SQRx registers (for regular channels) or in the ADC_JSQR register (for injected channels). A single conversion is performed for each channel of the group. After each end of conversion, the next channel in the group is converted automatically. If the CONT bit is set, regular channel conversion does not stop at the last selected channel in the group but continues again from the first selected channel.

If the DMA bit is set, the direct memory access (DMA) controller is used to transfer the data converted from the regular group of channels (stored in the ADC_DR register) to SRAM after each regular channel conversion.

The EOC bit is set in the ADC_SR register:

- At the end of each regular group sequence if the EOCS bit is cleared to 0
- At the end of each regular channel conversion if the EOCS bit is set to 1

The data converted from an injected channel are always stored into the ADC_JDRx registers.

14.3.10 Injected channel management

Triggered injection

To use triggered injection, the JAUTO bit must be cleared in the ADC_CR1 register.

1. Start the conversion of a group of regular channels either by external trigger or by setting the SWSTART bit in the ADC_CR2 register.
2. If an external injected trigger occurs or if the JSWSTART bit is set during the conversion of a regular group of channels, the current conversion is reset and the injected channel sequence switches to Scan-once mode.
3. Then, the regular conversion of the regular group of channels is resumed from the last interrupted regular conversion.

If a regular event occurs during an injected conversion, the injected conversion is not interrupted but the regular sequence is executed at the end of the injected sequence.

Figure 76 shows the corresponding 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 30 ADC clock cycles (that is two conversions with a sampling time of 3 clock periods), the minimum interval between triggers must be 31 ADC clock cycles.

Auto-injection

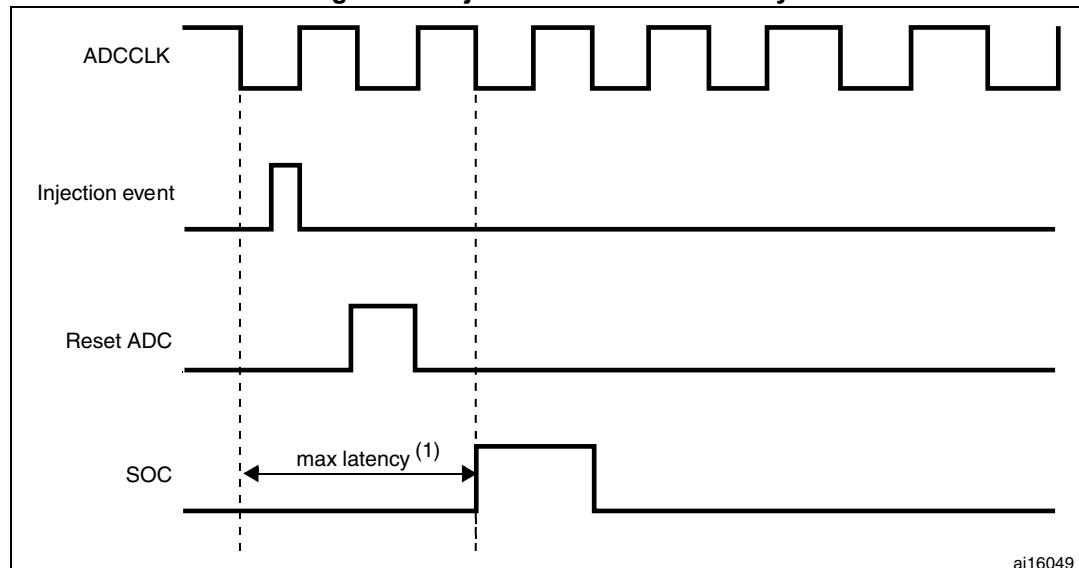
If the JAUTO bit is set, then the channels in the injected group are automatically converted after the regular group of channels. This can be used to convert a sequence of up to 20 conversions programmed in the ADC_SQRx and ADC_JSQR registers.

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.

Note: It is not possible to use both the auto-injected and discontinuous modes simultaneously.

Figure 76. Injected conversion latency



ai16049

1. The maximum latency value can be found in the electrical characteristics of the STM32F72xxx and STM32F73xxx datasheets.

14.3.11 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 ($n \leq 8$) that is 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. An EOC event is generated at each conversion.
- 2nd trigger: sequence converted 3, 6, 7. An EOC event is generated at each conversion
- 3rd trigger: sequence converted 9, 10. An EOC event is generated at each conversion
- 4th trigger: sequence converted 0, 1, 2. An EOC event is generated at each conversion

Note:

When a regular group is converted in discontinuous mode, no rollover occurs.

When all subgroups are converted, the next trigger starts the conversion of the first subgroup. In the example above, the 4th trigger reconverts the channels 0, 1 and 2 in the 1st subgroup.

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 JEOC event generated
- 4th trigger: channel 1

Note:

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.

It is not possible to use both the auto-injected and discontinuous modes simultaneously.

Discontinuous mode must not be set for regular and injected groups at the same time.

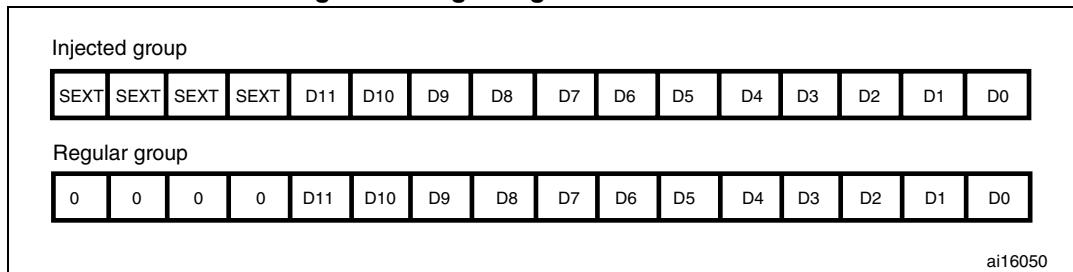
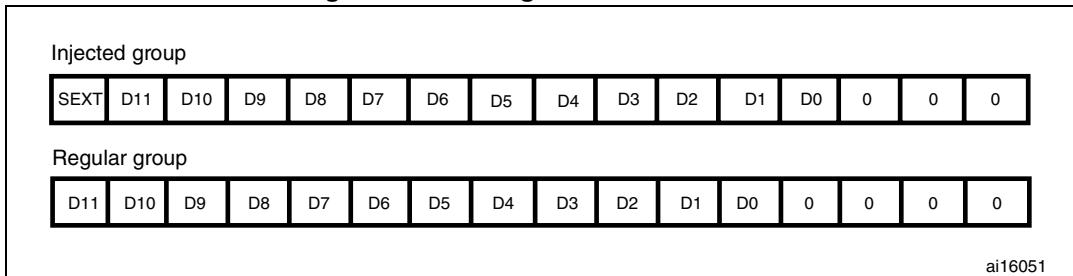
Discontinuous mode must be enabled only for the conversion of one group.

14.4 Data alignment

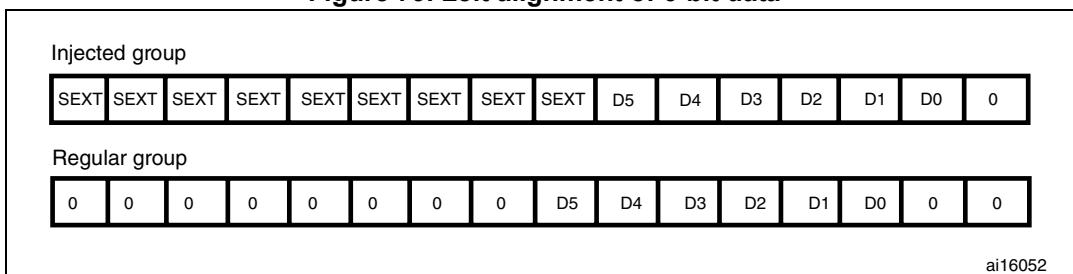
The ALIGN bit in the ADC_CR2 register selects the alignment of the data stored after conversion. Data can be right- or left-aligned as shown in [Figure 77](#) and [Figure 78](#).

The converted data value from the injected group of channels is decreased by the user-defined offset written in the ADC_JOFRx registers so the result can be a negative value. The SEXT bit represents the extended sign value.

For channels in a regular group, no offset is subtracted so only twelve bits are significant.

Figure 77. Right alignment of 12-bit data**Figure 78. Left alignment of 12-bit data**

Special case: when left-aligned, the data are aligned on a half-word basis except when the resolution is set to 6-bit. in that case, the data are aligned on a byte basis as shown in [Figure 79](#).

Figure 79. Left alignment of 6-bit data

14.5 Channel-wise programmable sampling time

The ADC samples the input voltage for a number of ADCCLK cycles that 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 sampling time.

The total conversion time is calculated as follows:

$$T_{\text{conv}} = \text{Sampling time} + 12 \text{ cycles}$$

Example:

With ADCCLK = 30 MHz and sampling time = 3 cycles:

$$T_{\text{conv}} = 3 + 12 = 15 \text{ cycles} = 0.5 \mu\text{s with APB2 at 60 MHz}$$

14.6 Conversion on external trigger and trigger polarity

Conversion can be triggered by an external event (e.g. timer capture, EXTI line). If the EXTEN[1:0] control bits (for a regular conversion) or JEXTEN[1:0] bits (for an injected conversion) are different from “0b00”, then external events are able to trigger a conversion with the selected polarity. [Table 87](#) provides the correspondence between the EXTEN[1:0] and JEXTEN[1:0] values and the trigger polarity.

Table 87. Configuring the trigger polarity

Source	EXTEN[1:0] / JEXTEN[1:0]
Trigger detection disabled	00
Detection on the rising edge	01
Detection on the falling edge	10
Detection on both the rising and falling edges	11

Note: The polarity of the external trigger can be changed on the fly.

The EXTSEL[3:0] and JEXTSEL[3:0] control bits are used to select which out of 16 possible events can trigger conversion for the regular and injected groups.

[Table 88](#) gives the possible external trigger for regular conversion.

Table 88. External trigger for regular channels

Source	Type	EXTSEL[3:0]
TIM1_CH1	Internal signal from on-chip timers	0000
TIM1_CH2		0001
TIM1_CH3		0010
TIM2_CH2		0011
TIM5_TRGO		0100
TIM4_CH4		0101
TIM3_CH4		0110
TIM8_TRGO		0111
TIM8_TRGO(2)		1000
TIM1_TRGO		1001
TIM1_TRGO(2)		1010
TIM2_TRGO		1011
TIM4_TRGO		1100
TIM6_TRGO		1101
EXTI line11	External pin	1111

Table 89 gives the possible external trigger for injected conversion.

Table 89. External trigger for injected channels

Source	Connection type	JEXTSEL[3:0]
TIM1_TRGO	Internal signal from on-chip timers	0000
TIM1_CH4		0001
TIM2_TRGO		0010
TIM2_CH1		0011
TIM3_CH4		0100
TIM4_TRGO		0101
TIM8_CH4	Internal signal from on-chip timers	0111
TIM1_TRGO(2)		1000
TIM8_TRGO		1001
TIM8_TRGO(2)		1010
TIM3_CH3		1011
TIM5_TRGO		1100
TIM3_CH1		1101
TIM6_TRGO		1110

Software source trigger events can be generated by setting SWSTART (for regular conversion) or JSWSTART (for injected conversion) in ADC_CR2.

A regular group conversion can be interrupted by an injected trigger.

Note:

The trigger selection can be changed on the fly. However, when the selection changes, there is a time frame of 1 APB clock cycle during which the trigger detection is disabled. This is to avoid spurious detection during transitions.

14.7 Fast conversion mode

It is possible to perform faster conversion by reducing the ADC resolution. The RES bits are used to select the number of bits available in the data register. The minimum conversion time for each resolution is then as follows:

- 12 bits: $3 + 12 = 15$ ADCCLK cycles
- 10 bits: $3 + 10 = 13$ ADCCLK cycles
- 8 bits: $3 + 8 = 11$ ADCCLK cycles
- 6 bits: $3 + 6 = 9$ ADCCLK cycles

14.8 Data management

14.8.1 Using the DMA

Since converted regular channel values are stored into a unique data register, it is useful to use DMA for conversion of more than one regular channel. This avoids the loss of the data already stored in the ADC_DR register.

When the DMA mode is enabled (DMA bit set to 1 in the ADC_CR2 register), after each conversion of a regular channel, a DMA request is generated. This allows the transfer of the converted data from the ADC_DR register to the destination location selected by the software.

Despite this, if data are lost (overrun), the OVR bit in the ADC_SR register is set and an interrupt is generated (if the OVRIE enable bit is set). DMA transfers are then disabled and DMA requests are no longer accepted. In this case, if a DMA request is made, the regular conversion in progress is aborted and further regular triggers are ignored. It is then necessary to clear the OVR flag and the DMAEN bit in the used DMA stream, and to re-initialize both the DMA and the ADC to have the wanted converted channel data transferred to the right memory location. Only then can the conversion be resumed and the data transfer, enabled again. Injected channel conversions are not impacted by overrun errors.

When OVR = 1 in DMA mode, the DMA requests are blocked after the last valid data have been transferred, which means that all the data transferred to the RAM can be considered as valid.

At the end of the last DMA transfer (number of transfers configured in the DMA controller's DMA_SxNTR register):

- No new DMA request is issued to the DMA controller if the DDS bit is cleared to 0 in the ADC_CR2 register (this avoids generating an overrun error). However the DMA bit is not cleared by hardware. It must be written to 0, then to 1 to start a new transfer.
- Requests can continue to be generated if the DDS bit is set to 1. This allows configuring the DMA in double-buffer circular mode.

To recover the ADC from OVR state when the DMA is used, follow the steps below:

1. Reinitialize the DMA (adjust destination address and NDTR counter)
2. Clear the ADC OVR bit in ADC_SR register
3. Trigger the ADC to start the conversion.

14.8.2 Managing a sequence of conversions without using the DMA

If the conversions are slow enough, the conversion sequence can be handled by the software. In this case the EOCS bit must be set in the ADC_CR2 register for the EOC status bit to be set at the end of each conversion, and not only at the end of the sequence. When EOCS = 1, overrun detection is automatically enabled. Thus, each time a conversion is complete, EOC is set and the ADC_DR register can be read. The overrun management is the same as when the DMA is used.

To recover the ADC from OVR state when the EOCS is set, follow the steps below:

1. Clear the ADC OVR bit in ADC_SR register
2. Trigger the ADC to start the conversion.

14.8.3 Conversions without DMA and without overrun detection

It may be useful to let the ADC convert one or more channels without reading the data each time (if there is an analog watchdog for instance). For that, the DMA must be disabled ($DMA = 0$) and the EOC bit must be set at the end of a sequence only ($EOCS = 0$). In this configuration, overrun detection is disabled.

14.9 Multi ADC mode

In devices with two ADCs or more, the Dual (with two ADCs) and Triple (with three ADCs) ADC modes can be used (see [Figure 80](#)).

In multi ADC mode, the start of conversion is triggered alternately or simultaneously by the ADC1 master to the ADC2 and ADC3 slaves, depending on the mode selected by the $MULTI[4:0]$ bits in the ADC_CCR register.

Note: *In multi ADC mode, when configuring conversion trigger by an external event, the application must set trigger by the master only and disable trigger by slaves to prevent spurious triggers that would start unwanted slave conversions.*

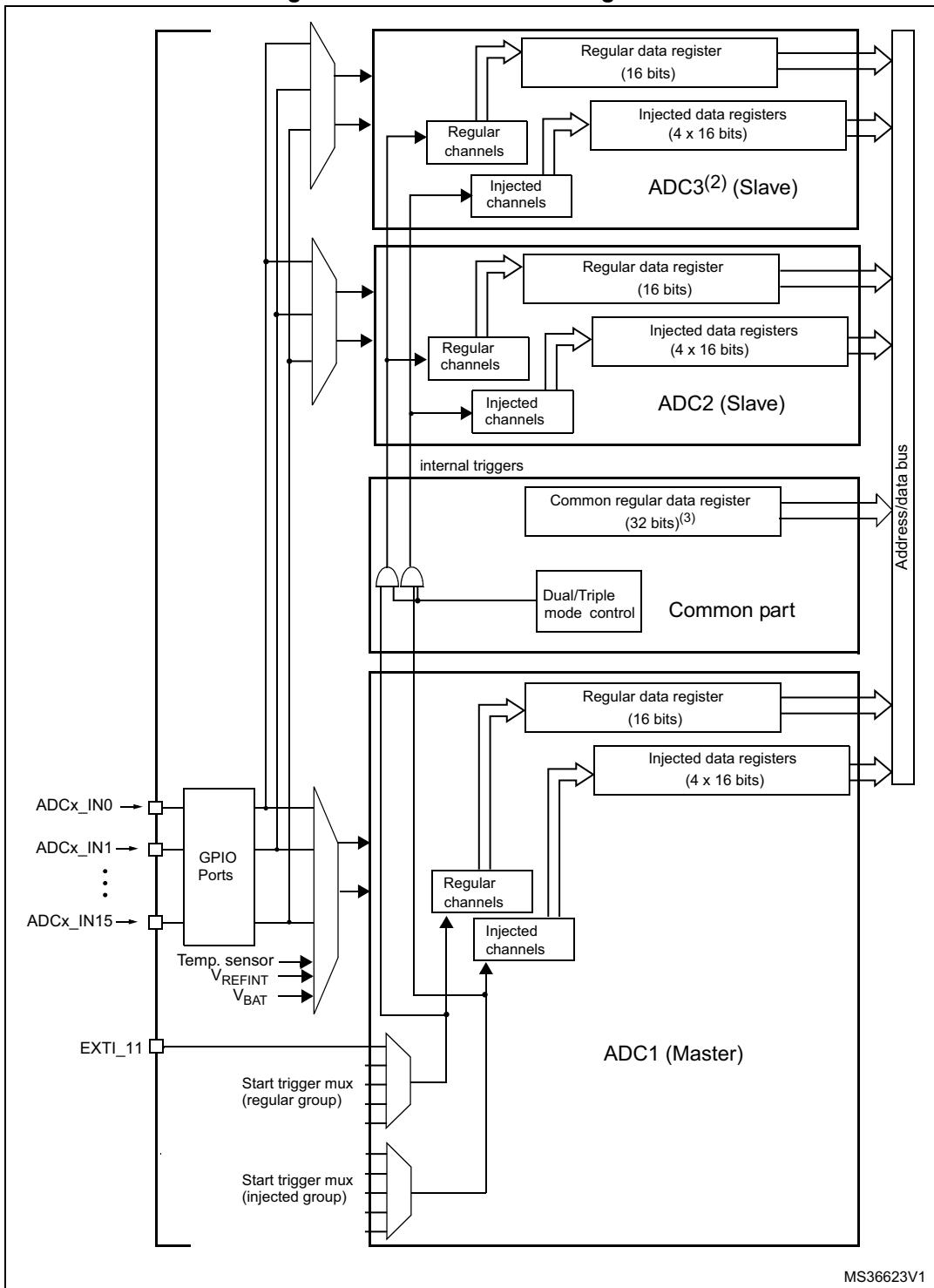
The four possible modes below are implemented:

- Injected simultaneous mode
- Regular simultaneous mode
- Interleaved mode
- Alternate trigger 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

Note: *In multi ADC mode, the converted data can be read on the multi-mode data register (ADC_CDR). The status bits can be read in the multi-mode status register (ADC_CSR).*

Figure 80. Multi ADC block diagram⁽¹⁾

MS36623V1

1. Although external triggers are present on ADC2 and ADC3 they are not shown in this diagram.
2. In the Dual ADC mode, the ADC3 slave part is not present.
3. In Triple ADC mode, the ADC common data register (ADC_CDR) contains the ADC1, ADC2 and ADC3's regular converted data. All 32 register bits are used according to a selected storage order.
In Dual ADC mode, the ADC common data register (ADC_CDR) contains both the ADC1 and ADC2's regular converted data. All 32 register bits are used.

- DMA requests in Multi ADC mode:

In Multi ADC mode the DMA may be configured to transfer converted data in three different modes. In all cases, the DMA streams to use are those connected to the ADC:

- **DMA mode 1:** On each DMA request (one data item is available), a half-word representing an ADC-converted data item is transferred.

In Dual ADC mode, ADC1 data are transferred on the first request, ADC2 data are transferred on the second request and so on.

In Triple ADC mode, ADC1 data are transferred on the first request, ADC2 data are transferred on the second request and ADC3 data are transferred on the third request; the sequence is repeated. So the DMA first transfers ADC1 data followed by ADC2 data followed by ADC3 data and so on.

DMA mode 1 is used in regular simultaneous triple mode.

Example:

Regular simultaneous triple mode: 3 consecutive DMA requests are generated (one for each converted data item)

1st request: $\text{ADC_CDR}[31:0] = \text{ADC1_DR}[15:0]$

2nd request: $\text{ADC_CDR}[31:0] = \text{ADC2_DR}[15:0]$

3rd request: $\text{ADC_CDR}[31:0] = \text{ADC3_DR}[15:0]$

4th request: $\text{ADC_CDR}[31:0] = \text{ADC1_DR}[15:0]$

- **DMA mode 2:** On each DMA request (two data items are available) two half-words representing two ADC-converted data items are transferred as a word.

In Dual ADC mode, both ADC2 and ADC1 data are transferred on the first request (ADC2 data take the upper half-word and ADC1 data take the lower half-word) and so on.

In Triple ADC mode, three DMA requests are generated. On the first request, both ADC2 and ADC1 data are transferred (ADC2 data take the upper half-word and ADC1 data take the lower half-word). On the second request, both ADC1 and ADC3 data are transferred (ADC1 data take the upper half-word and ADC3 data take the lower half-word). On the third request, both ADC3 and ADC2 data are transferred (ADC3 data take the upper half-word and ADC2 data take the lower half-word) and so on.

DMA mode 2 is used in interleaved mode and in regular simultaneous mode (for Dual ADC mode only).

Example:

- a) Interleaved dual mode: a DMA request is generated each time 2 data items are available:

1st request: $\text{ADC_CDR}[31:0] = \text{ADC2_DR}[15:0] | \text{ADC1_DR}[15:0]$

2nd request: $\text{ADC_CDR}[31:0] = \text{ADC2_DR}[15:0] | \text{ADC1_DR}[15:0]$

- b) Interleaved triple mode: a DMA request is generated each time 2 data items are available

1st request: $\text{ADC_CDR}[31:0] = \text{ADC2_DR}[15:0] | \text{ADC1_DR}[15:0]$

2nd request: $\text{ADC_CDR}[31:0] = \text{ADC1_DR}[15:0] | \text{ADC3_DR}[15:0]$

3rd request: $\text{ADC_CDR}[31:0] = \text{ADC3_DR}[15:0] | \text{ADC2_DR}[15:0]$

4th request: $\text{ADC_CDR}[31:0] = \text{ADC2_DR}[15:0] | \text{ADC1_DR}[15:0]$

- **DMA mode 3:** This mode is similar to the DMA mode 2. The only differences are that on each DMA request (two data items are available) two bytes representing two ADC converted data items are transferred as a half-word. The data transfer order is similar to that of the DMA mode 2.

DMA mode 3 is used in interleaved mode in 6-bit and 8-bit resolutions.

Example:

- a) Interleaved dual mode: a DMA request is generated each time 2 data items are available
 - 1st request: $\text{ADC_CDR}[15:0] = \text{ADC2_DR}[7:0] | \text{ADC1_DR}[7:0]$
 - 2nd request: $\text{ADC_CDR}[15:0] = \text{ADC2_DR}[7:0] | \text{ADC1_DR}[7:0]$
- b) Interleaved triple mode: a DMA request is generated each time 2 data items are available
 - 1st request: $\text{ADC_CDR}[15:0] = \text{ADC2_DR}[7:0] | \text{ADC1_DR}[7:0]$
 - 2nd request: $\text{ADC_CDR}[15:0] = \text{ADC1_DR}[7:0] | \text{ADC3_DR}[15:0]$
 - 3rd request: $\text{ADC_CDR}[15:0] = \text{ADC3_DR}[7:0] | \text{ADC2_DR}[7:0]$
 - 4th request: $\text{ADC_CDR}[15:0] = \text{ADC2_DR}[7:0] | \text{ADC1_DR}[7:0]$

Overrun detection: If an overrun is detected on one of the concerned ADCs (ADC1 and ADC2 in dual and triple modes, ADC3 in triple mode only), the DMA requests are no longer issued to ensure that all the data transferred to the RAM are valid. It may happen that the EOC bit corresponding to one ADC remains set because the data register of this ADC contains valid data.

14.9.1 Injected simultaneous mode

This mode converts an injected group of channels. The external trigger source comes from the injected group multiplexer of ADC1 (selected by the JEXTSEL[3:0] bits in the ADC1_CR2 register). A simultaneous trigger is provided to ADC2 and ADC3.

Note:

Do not convert the same channel on the two/three ADCs (no overlapping sampling times for the two/three ADCs when converting the same channel).

In simultaneous mode, one must convert sequences with the same length or ensure that the interval between triggers is longer than the longer of the 2 sequences (Dual ADC mode) /3 sequences (Triple ADC mode). Otherwise, the ADC with the shortest sequence may restart while the ADC with the longest sequence is completing the previous conversions.

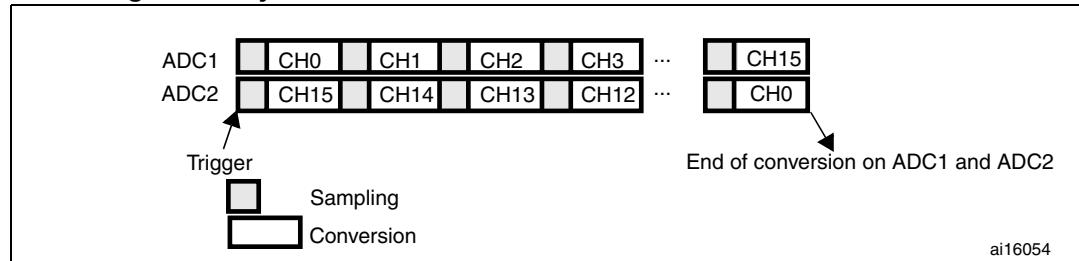
Regular conversions can be performed on one or all ADCs. In that case, they are independent of each other and are interrupted when an injected event occurs. They are resumed at the end of the injected conversion group.

Dual ADC mode

At the end of conversion event on ADC1 or ADC2:

- The converted data are stored into the ADC_JDRx registers of each ADC interface.
- A JEOC interrupt is generated (if enabled on one of the two ADC interfaces) when the ADC1/ADC2's injected channels have all been converted.

Figure 81. Injected simultaneous mode on 4 channels: dual ADC mode



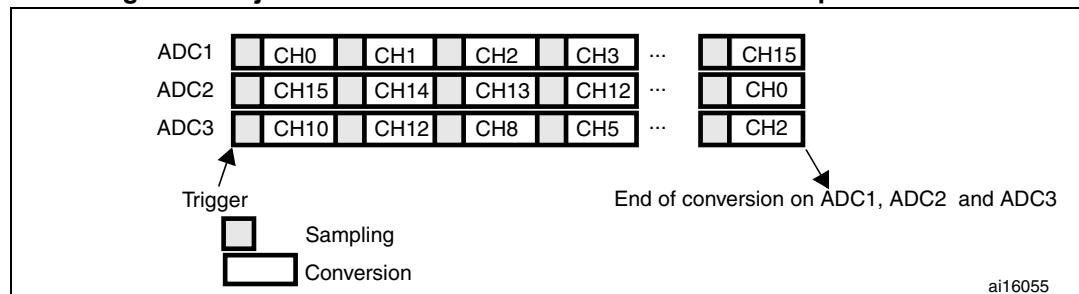
ai16054

Triple ADC mode

At the end of conversion event on ADC1, ADC2 or ADC3:

- The converted data are stored into the ADC_JDRx registers of each ADC interface.
- A JEOC interrupt is generated (if enabled on one of the three ADC interfaces) when the ADC1/ADC2/ADC3's injected channels have all been converted.

Figure 82. Injected simultaneous mode on 4 channels: triple ADC mode



ai16055

14.9.2 Regular simultaneous mode

This mode is performed on a regular group of channels. The external trigger source comes from the regular group multiplexer of ADC1 (selected by the EXTSEL[3:0] bits in the ADC1_CR2 register). A simultaneous trigger is provided to ADC2 and ADC3.

Note: *Do not convert the same channel on the two/three ADCs (no overlapping sampling times for the two/three ADCs when converting the same channel).*

In regular simultaneous mode, one must convert sequences with the same length or ensure that the interval between triggers is longer than the long conversion time of the 2 sequences (Dual ADC mode) /3 sequences (Triple ADC mode). Otherwise, the ADC with the shortest sequence may restart while the ADC with the longest sequence is completing the previous conversions.

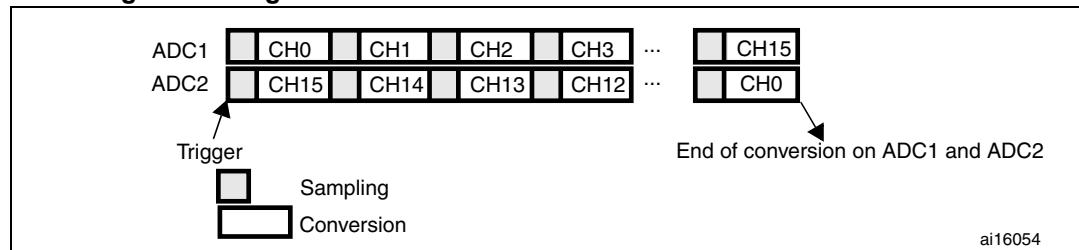
Injected conversions must be disabled.

Dual ADC mode

At the end of conversion event on ADC1 or ADC2:

- A 32-bit DMA transfer request is generated (if DMA[1:0] bits in the ADC_CCR register are equal to 0b10). This request transfers the ADC2 converted data stored in the upper half-word of the ADC_CDR 32-bit register to the SRAM and then the ADC1 converted data stored in the lower half-word of ADC_CCR to the SRAM.
- An EOC interrupt is generated (if enabled on one of the two ADC interfaces) when the ADC1/ADC2's regular channels have all been converted.

Figure 83. Regular simultaneous mode on 16 channels: dual ADC mode



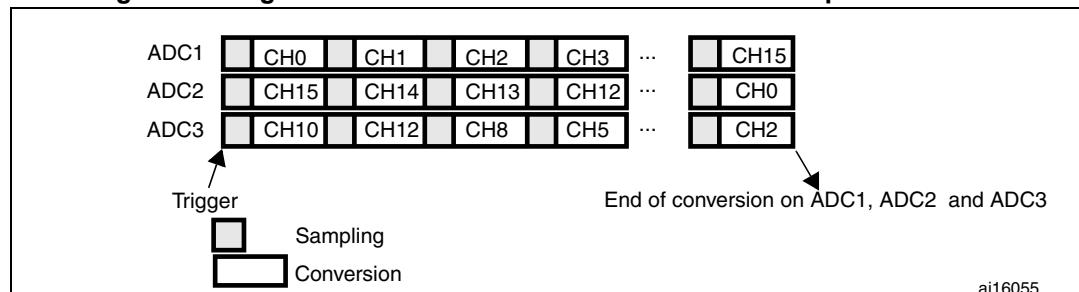
ai16054

Triple ADC mode

At the end of conversion event on ADC1, ADC2 or ADC3:

- Three 32-bit DMA transfer requests are generated (if DMA[1:0] bits in the ADC_CCR register are equal to 0b01). Three transfers then take place from the ADC_CDR 32-bit register to SRAM: first the ADC1 converted data, then the ADC2 converted data and finally the ADC3 converted data. The process is repeated for each new three conversions.
- An EOC interrupt is generated (if enabled on one of the three ADC interfaces) when the ADC1/ADC2/ADC3's regular channels have all been converted.

Figure 84. Regular simultaneous mode on 16 channels: triple ADC mode



ai16055

14.9.3 Interleaved mode

This mode can be started only on a regular group (usually one channel). The external trigger source comes from the regular channel multiplexer of ADC1.

Dual ADC mode

After an external trigger occurs:

- ADC1 starts immediately
- ADC2 starts after a delay of several-ADC clock cycles

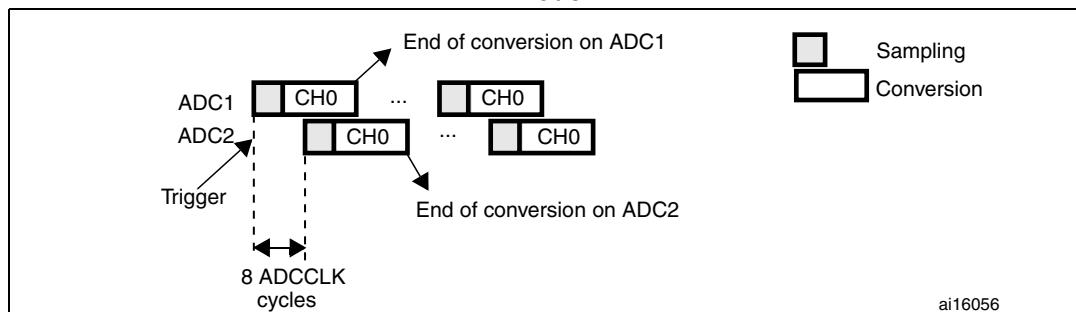
The minimum delay which separates 2 conversions in interleaved mode is configured in the DELAY bits in the ADC_CCR register. However, an ADC cannot start a conversion if the complementary ADC is still sampling its input (only one ADC can sample the input signal at a given time). In this case, the delay becomes the sampling time + 2 ADC clock cycles. For instance, if $\text{DELAY} = 5$ clock cycles and the sampling takes 15 clock cycles on both ADCs, then 17 clock cycles will separate conversions on ADC1 and ADC2).

If the CONT bit is set on both ADC1 and ADC2, the selected regular channels of both ADCs are continuously converted.

Note: *If the conversion sequence is interrupted (for instance when DMA end of transfer occurs), the multi-ADC sequencer must be reset by configuring it in independent mode first (bits DUAL[4:0] = 00000) before reprogramming the interleaved mode.*

After an EOC interrupt is generated by ADC2 (if enabled through the EOCIE bit) a 32-bit DMA transfer request is generated (if the DMA[1:0] bits in ADC_CCR are equal to 0b10). This request first transfers the ADC2 converted data stored in the upper half-word of the ADC_CDR 32-bit register into SRAM, then the ADC1 converted data stored in the register's lower half-word into SRAM.

Figure 85. Interleaved mode on 1 channel in continuous conversion mode: dual ADC mode



Triple ADC mode

After an external trigger occurs:

- ADC1 starts immediately and
- ADC2 starts after a delay of several ADC clock cycles
- ADC3 starts after a delay of several ADC clock cycles referred to the ADC2 conversion

The minimum delay which separates 2 conversions in interleaved mode is configured in the DELAY bits in the ADC_CCR register. However, an ADC cannot start a conversion if the complementary ADC is still sampling its input (only one ADC can sample the input signal at a given time). In this case, the delay becomes the sampling time + 2 ADC clock cycles. For instance, if $\text{DELAY} = 5$ clock cycles and the sampling takes 15 clock cycles on the three ADCs, then 17 clock cycles will separate the conversions on ADC1, ADC2 and ADC3).

If the CONT bit is set on ADC1, ADC2 and ADC3, the selected regular channels of all ADCs are continuously converted.

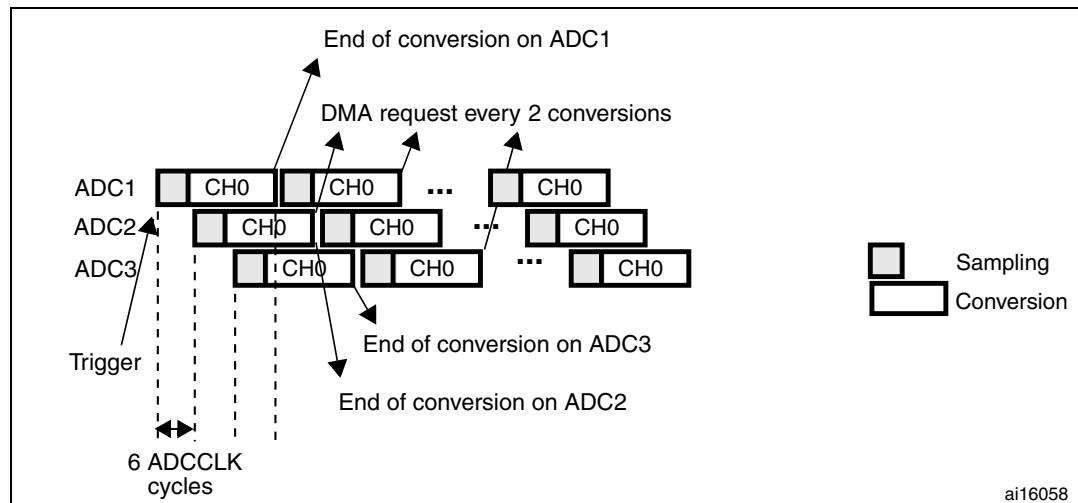
Note: *If the conversion sequence is interrupted (for instance when DMA end of transfer occurs), the multi-ADC sequencer must be reset by configuring it in independent mode first (bits DUAL[4:0] = 00000) before reprogramming the interleaved mode.*

In this mode a DMA request is generated each time 2 data items are available, (if the DMA[1:0] bits in the ADC_CCR register are equal to 0b10). The request first transfers the

first converted data stored in the lower half-word of the ADC_CDR 32-bit register to SRAM, then it transfers the second converted data stored in ADC_CDR's upper half-word to SRAM. The sequence is the following:

- 1st request: ADC_CDR[31:0] = ADC2_DR[15:0] | ADC1_DR[15:0]
- 2nd request: ADC_CDR[31:0] = ADC1_DR[15:0] | ADC3_DR[15:0]
- 3rd request: ADC_CDR[31:0] = ADC3_DR[15:0] | ADC2_DR[15:0]
- 4th request: ADC_CDR[31:0] = ADC2_DR[15:0] | ADC1_DR[15:0], ...

Figure 86. Interleaved mode on 1 channel in continuous conversion mode: triple ADC mode



14.9.4 Alternate trigger mode

This mode can be started only on an injected group. The source of external trigger comes from the injected group multiplexer of ADC1.

Note: *Regular conversions can be enabled on one or all ADCs. In this case the regular conversions are independent of each other. A regular conversion is interrupted when the ADC has to perform an injected conversion. It is resumed when the injected conversion is finished.*

If the conversion sequence is interrupted (for instance when DMA end of transfer occurs), the multi-ADC sequencer must be reset by configuring it in independent mode first (bits DUAL[4:0] = 00000) before reprogramming the interleaved mode.

The time interval between 2 trigger events must be greater than or equal to 1 ADC clock period. The minimum time interval between 2 trigger events that start conversions on the same ADC is the same as in the single ADC mode.

Dual ADC mode

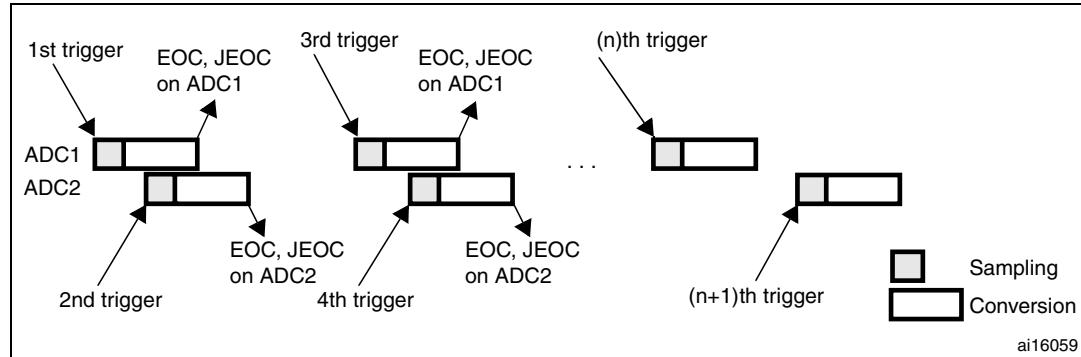
- When the 1st trigger occurs, all injected ADC1 channels in the group are converted
- When the 2nd trigger occurs, all injected ADC2 channels in the group are converted
- and so on

A JEOC interrupt, if enabled, is generated after all injected ADC1 channels in the group have been converted.

A JEOC interrupt, if enabled, is generated after all injected ADC2 channels in the group have been converted.

If another external trigger occurs after all injected channels in the group have been converted then the alternate trigger process restarts by converting the injected ADC1 channels in the group.

Figure 87. Alternate trigger: injected group of each ADC



If the injected discontinuous mode is enabled for both ADC1 and ADC2:

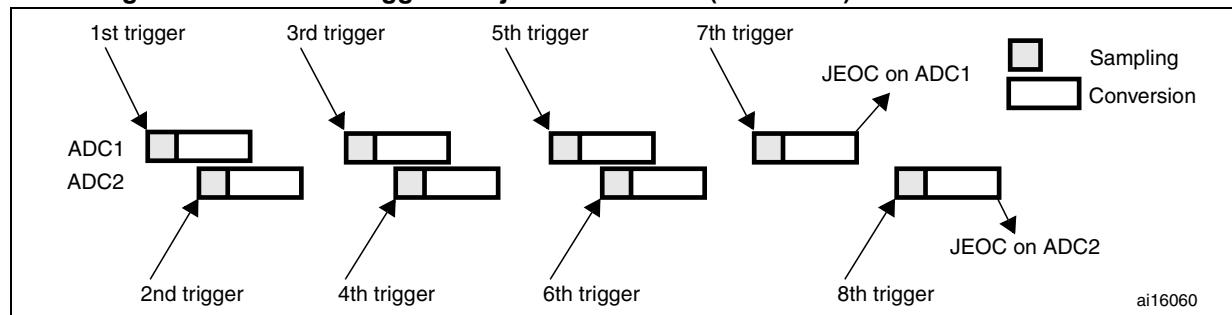
- When the 1st trigger occurs, the first injected ADC1 channel is converted.
- When the 2nd trigger occurs, the first injected ADC2 channel are converted
- and so on

A JEOC interrupt, if enabled, is generated after all injected ADC1 channels in the group have been converted.

A JEOC interrupt, if enabled, is generated after all injected ADC2 channels in the group have been converted.

If another external trigger occurs after all injected channels in the group have been converted then the alternate trigger process restarts.

Figure 88. Alternate trigger: 4 injected channels (each ADC) in discontinuous mode



Triple ADC mode

- When the 1st trigger occurs, all injected ADC1 channels in the group are converted.
- When the 2nd trigger occurs, all injected ADC2 channels in the group are converted.
- When the 3rd trigger occurs, all injected ADC3 channels in the group are converted.
- and so on

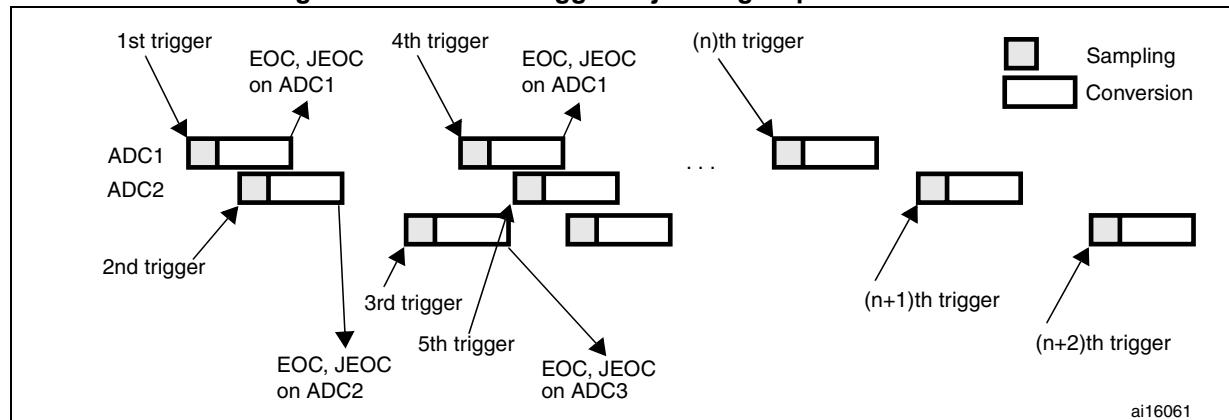
A JEOC interrupt, if enabled, is generated after all injected ADC1 channels in the group have been converted.

A JEOC interrupt, if enabled, is generated after all injected ADC2 channels in the group have been converted.

A JEOC interrupt, if enabled, is generated after all injected ADC3 channels in the group have been converted.

If another external trigger occurs after all injected channels in the group have been converted then the alternate trigger process restarts by converting the injected ADC1 channels in the group.

Figure 89. Alternate trigger: injected group of each ADC



14.9.5 Combined regular/injected simultaneous mode

It is possible to interrupt the simultaneous conversion of a regular group to start the 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 long conversion time of the 2 sequences (Dual ADC mode) /3 sequences (Triple ADC mode). Otherwise, the ADC with the shortest sequence may restart while the ADC with the longest sequence is completing the previous conversions.*

14.9.6 Combined regular simultaneous + alternate trigger mode

It is possible to interrupt the simultaneous conversion of a regular group to start the alternate trigger conversion of an injected group. [Figure 90](#) shows the behavior of an alternate trigger interrupting a simultaneous regular conversion.

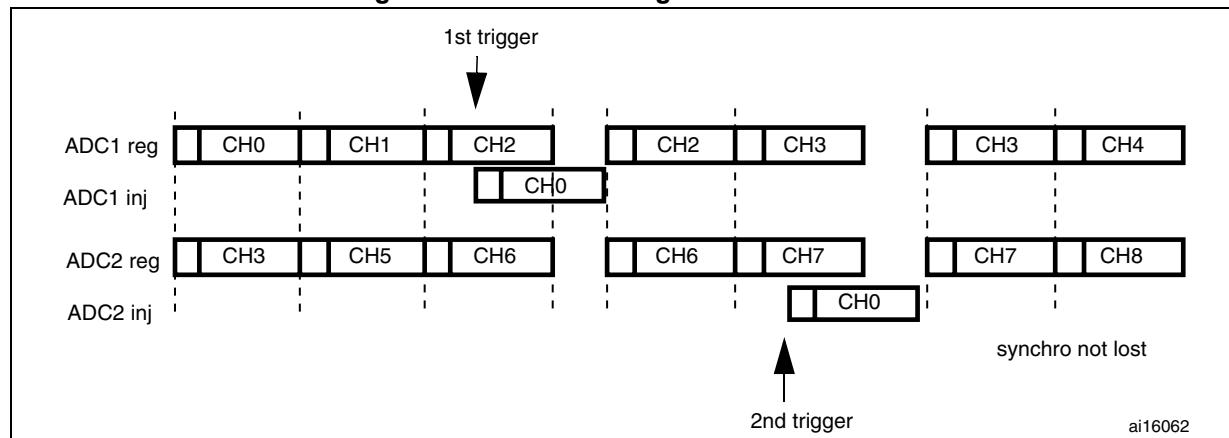
The injected alternate conversion is immediately started after the injected event. If regular conversion is already running, in order to ensure synchronization after the injected conversion, the regular conversion of all (master/slave) ADCs is stopped and resumed synchronously at the end of the injected 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 long conversion time of the 2 sequences (Dual ADC mode) /3 sequences (Triple ADC mode).*

Otherwise, the ADC with the shortest sequence may restart while the ADC with the longest sequence is completing the previous conversions.

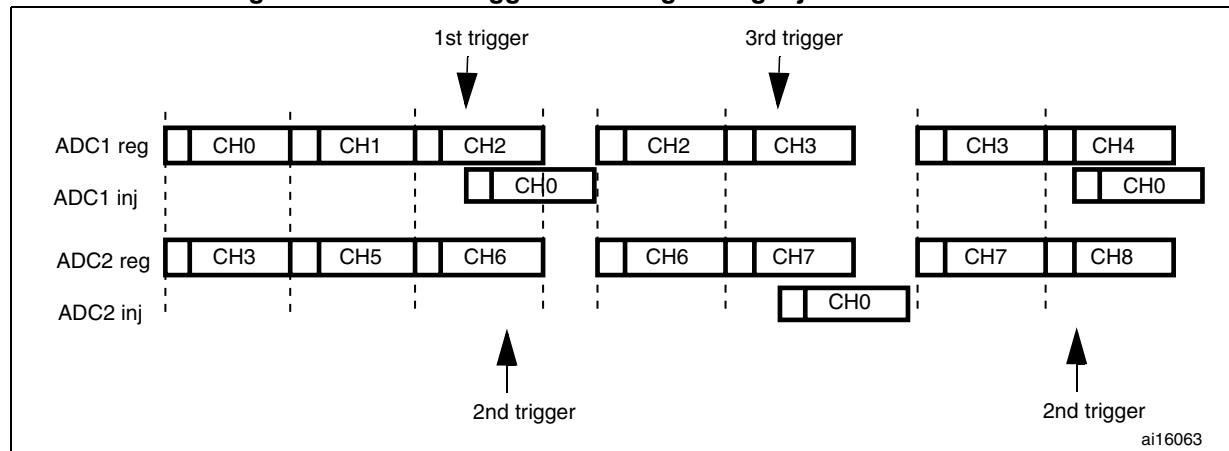
If the conversion sequence is interrupted (for instance when DMA end of transfer occurs), the multi-ADC sequencer must be reset by configuring it in independent mode first (bits DUAL[4:0] = 00000) before reprogramming the interleaved mode.

Figure 90. Alternate + regular simultaneous



If a trigger occurs during an injected conversion that has interrupted a regular conversion, it is ignored. [Figure 91](#) shows the behavior in this case (2nd trigger is ignored).

Figure 91. Case of trigger occurring during injected conversion



14.10 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 same input channel, ADC1_IN18, as VBAT: ADC1_IN18 is used to convert the sensor output voltage or VBAT into a digital value. Only one conversion, temperature sensor or VBAT, must be selected at a time. When the temperature sensor and the VBAT conversion are set simultaneously, only the VBAT conversion is performed.

[Figure 92](#) shows the block diagram of the temperature sensor.

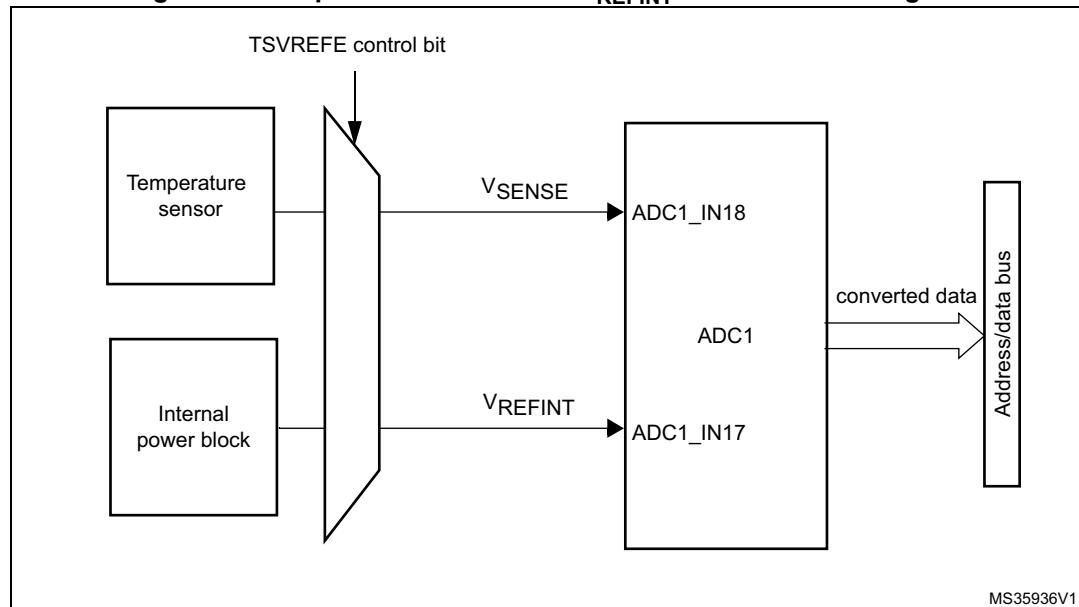
When not in use, the sensor can be put in power down mode.

Note: The TSVREFE bit must be set to enable the conversion of both internal channels: the ADC1_IN18 (temperature sensor) and the ADC1_IN17 (VREFINT).

Main features

- Supported temperature range: -40 to 125 °C
- Precision: ±1.5 °C

Figure 92. Temperature sensor and V_{REFINT} channel block diagram



1. V_{SENSE} is input to ADC1_IN18.

Reading the temperature

To use the sensor:

3. Select ADC1_IN18 input channel.
4. Select a sampling time greater than the minimum sampling time specified in the datasheet.
5. Set the TSVREFE bit in the ADC_CCR register to wake up the temperature sensor from power down mode
6. Start the ADC conversion by setting the SWSTART bit (or by external trigger)
7. Read the resulting V_{SENSE} data in the ADC data register
8. Calculate the temperature using the following formula:

$$\text{Temperature (in } ^\circ\text{C)} = \{(V_{\text{SENSE}} - V_{25}) / \text{Avg_Slope}\} + 25$$

Where:

- V₂₅ = V_{SENSE} value for 25° C
- Avg_Slope = average slope of the temperature vs. V_{SENSE} curve (given in mV/°C or µV/°C)

Refer to the datasheet electrical characteristics section for the actual values of V₂₅ 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.*

The temperature sensor output voltage changes linearly with temperature. The offset of this linear function depends on each chip due to process variation (up to 45 °C from one chip to another).

The internal temperature sensor is more suited for applications that detect temperature variations instead of absolute temperatures. If accurate temperature reading is required, an external temperature sensor should be used.

14.11 Battery charge monitoring

The VBAT bit in the ADC_CCR register is used to switch to the battery voltage. As the V_{BAT} voltage could be higher than V_{DDA} , to ensure the correct operation of the ADC, the V_{BAT} pin is internally connected to a bridge divider.

When the VBAT is set, the bridge is automatically enabled to connect:

- VBAT/4 to the ADC1_IN18 input channel

Note: *The VBAT and temperature sensor are connected to the same ADC internal channel (ADC1_IN18). Only one conversion, either temperature sensor or VBAT, must be selected at a time. When both conversion are enabled simultaneously, only the VBAT conversion is performed.*

14.12 ADC interrupts

An interrupt can be produced on the end of conversion for regular and injected groups, when the analog watchdog status bit is set and when the overrun 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 channels of an injected group)
- START (Start of conversion for channels of a regular group)

Table 90. ADC interrupts

Interrupt event	Event flag	Enable control bit
End of conversion of a regular group	EOC	EOCIE
End of conversion of an injected group	JEOC	JEOCIE
Analog watchdog status bit is set	AWD	AWDIE
Overrun	OVR	OVRIE

14.13 ADC registers

Refer to [Section 1.2 on page 53](#) for a list of abbreviations used in register descriptions.

The peripheral registers must be written at word level (32 bits). Read accesses can be done by bytes (8 bits), half-words (16 bits) or words (32 bits).

14.13.1 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
Res.	Res.	Res.	Res.	Res.	Res.										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	OVR	STRT	JSTRT	JEOC	EOC	AWD									
										rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

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

Bit 5 **OVR**: Overrun

This bit is set by hardware when data are lost (either in single mode or in dual/triple mode). It is cleared by software. Overrun detection is enabled only when DMA = 1 or EOCS = 1.

- 0: No overrun occurred
- 1: Overrun has occurred

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 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 the conversion of all injected channels in the group. It is cleared by software.

- 0: Conversion is not complete
- 1: Conversion complete

Bit 1 **EOC**: Regular channel end of conversion

This bit is set by hardware at the end of the conversion of a regular group of channels. It is cleared by software or by reading the ADC_DR register.

- 0: Conversion not complete (EOCS=0), or sequence of conversions not complete (EOCS=1)
- 1: Conversion complete (EOCS=0), or sequence of conversions complete (EOCS=1)

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

14.13.2 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
Res.	Res.	Res.	Res.	Res.	OVRIE	RES		AWDEN	JAWDEN	Res.	Res.	Res.	Res.	Res.	Res.
					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 31:27 Reserved, must be kept at reset value.

Bit 26 **OVRIE**: Overrun interrupt enable

This bit is set and cleared by software to enable/disable the Overrun interrupt.

0: Overrun interrupt disabled

1: Overrun interrupt enabled. An interrupt is generated when the OVR bit is set.

Bits 25:24 **RES[1:0]**: Resolution

These bits are written by software to select the resolution of the conversion.

00: 12-bit (minimum 15 ADCCLK cycles)

01: 10-bit (minimum 13 ADCCLK cycles)

10: 8-bit (minimum 11 ADCCLK cycles)

11: 6-bit (minimum 9 ADCCLK cycles)

Bit 23 **AWDEN**: Analog watchdog enable on regular channels

This bit is set and cleared 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 and cleared by software.

0: Analog watchdog disabled on injected channels

1: Analog watchdog enabled on injected channels

Bits 21:16 Reserved, must be kept at reset value.

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 is set and cleared by software to enable/disable discontinuous mode on the injected channels of a group.

0: Discontinuous mode on injected channels disabled

1: Discontinuous mode on injected channels enabled

Bit 11 **DISCEN:** Discontinuous mode on regular channels

This bit is 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 is 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 is 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 the 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 interrupt is generated if the EOCS bit is set:

- At the end of each regular group sequence if the EOCS bit is cleared to 0
- At the end of each regular channel conversion if the EOCS bit is set to 1

Note: A JEOC interrupt is generated only on the end of conversion of the last channel if the 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.

- 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.

Note: 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

10010: ADC analog input Channel18

Other values reserved

14.13.3 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
Res.	SWSTART	EXTEN		EXTSEL[3:0]				Res.	JSWSTART	JEXTEN		JEXTSEL[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	ALIGN	EOCS	DDS	DMA	Res.	Res.	Res.	Res.	Res.	Res.	CONT	ADON
				rw	rw	rw	rw							rw	rw

Bit 31 Reserved, must be kept at reset value.

Bit 30 **SWSTART**: Start conversion of regular channels

This bit is set by software to start conversion and cleared by hardware as soon as the conversion starts.

0: Reset state

1: Starts conversion of regular channels

Note: This bit can be set only when ADON = 1 otherwise no conversion is launched.

Bits 29:28 **EXTEN**: External trigger enable for regular channels

These bits are set and cleared by software to select the external trigger polarity and enable the trigger of a regular group.

00: Trigger detection disabled

01: Trigger detection on the rising edge

10: Trigger detection on the falling edge

11: Trigger detection on both the rising and falling edges

Bits 27:24 **EXTSEL[3:0]**: External event select for regular group

These bits select the external event used to trigger the start of conversion of a regular group:

0000: Timer 1 CH1

0001: Timer 1 CH2

0010: Timer 1 CH3

0011: Timer 2 CH2

0100: Timer 5 TRGO

0101: Timer 4 CH4

0110: Timer 3 CH4

0111: Timer 8 TRGO

1000: Timer 8 TRGO(2)

1001: Timer 1 TRGO

1010: Timer 1 TRGO(2)

1011: Timer 2 TRGO

1100: Timer 4 TRGO

1101: Timer 6 TRGO

1110: Reserved

1111: EXTI line11

Bit 23 Reserved, must be kept at reset value.

Bit 22 **JSWSTART**: Start conversion of injected channels

This bit is set by software and cleared by hardware as soon as the conversion starts.

0: Reset state

1: Starts conversion of injected channels

This bit can be set only when ADON = 1 otherwise no conversion is launched.

Bits 21:20 **JEXTEN**: External trigger enable for injected channels

These bits are set and cleared by software to select the external trigger polarity and enable the trigger of an injected group.

00: Trigger detection disabled

01: Trigger detection on the rising edge

10: Trigger detection on the falling edge

11: Trigger detection on both the rising and falling edges

Bits 19:16 **JEXTSEL[3:0]**: External event select for injected group

These bits select the external event used to trigger the start of conversion of an injected group.

0000: Timer 1 TRGO

0001: Timer 1 CH4

0010: Timer 2 TRGO

0011: Timer 2 CH1

0100: Timer 3 CH4

0101: Timer4 TRGO

0110: Reserved

0111: Timer 8 CH4

1000: Timer 1 TRGO(2)

1001: Timer 8 TRGO

1010: Timer 8 TRGO(2)

1011: Timer 3 CH3

1100: Timer 5 TRGO

1101: Timer 3 CH1

1110: Timer 6 TRGO

1111: Reserved

Bits 15:12 Reserved, must be kept at reset value.

Bit 11 **ALIGN**: Data alignment

This bit is set and cleared by software. Refer to [Figure 77](#) and [Figure 78](#).

0: Right alignment

1: Left alignment

Bit 10 **EOCS**: End of conversion selection

This bit is set and cleared by software.

0: The EOC bit is set at the end of each sequence of regular conversions. Overrun detection is enabled only if DMA=1.

1: The EOC bit is set at the end of each regular conversion. Overrun detection is enabled.

Bit 9 **DDS**: DMA disable selection (for single ADC mode)

This bit is set and cleared by software.

0: No new DMA request is issued after the last transfer (as configured in the DMA controller)

1: DMA requests are issued as long as data are converted and DMA=1

Bit 8 **DMA**: Direct memory access mode (for single ADC 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

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

Bit 1 **CONT**: Continuous conversion

This bit is set and cleared by software. If it is set, conversion takes place continuously until it is cleared.

0: Single conversion mode

1: Continuous conversion mode

Bit 0 **ADON**: A/D Converter ON / OFF

This bit is set and cleared by software.

0: Disable ADC conversion and go to power down mode

1: Enable ADC

14.13.4 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
Res.	Res.	Res.	Res.	Res.	SMP18[2:0]			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
SMP15_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: 27 Reserved, must be kept at reset value.

Bits 26:0 **SMPx[2:0]**: Channel x sampling time selection

These bits are written by software to select the sampling time individually for each channel.

During sampling cycles, the channel selection bits must remain unchanged.

- Note:
- 000: 3 cycles
 - 001: 15 cycles
 - 010: 28 cycles
 - 011: 56 cycles
 - 100: 84 cycles
 - 101: 112 cycles
 - 110: 144 cycles
 - 111: 480 cycles

14.13.5 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	
Res.	Res.	SMP9[2:0]				SMP8[2:0]				SMP7[2:0]				SMP6[2:0]		
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
SMP5_0	SMP4[2:0]				SMP3[2:0]				SMP2[2:0]				SMP1[2:0]		SMP0[2:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bits 31:30 Reserved, must be kept at reset value.

Bits 29:0 **SMPx[2:0]**: Channel x sampling time selection

These bits are written by software to select the sampling time individually for each channel.
During sample cycles, the channel selection bits must remain unchanged.

Note:
 000: 3 cycles
 001: 15 cycles
 010: 28 cycles
 011: 56 cycles
 100: 84 cycles
 101: 112 cycles
 110: 144 cycles
 111: 480 cycles

14.13.6 ADC injected channel data offset register x (ADC_JOFRx) (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
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	JOFFSETx[11:0]											
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:12 Reserved, must be kept at reset value.

Bits 11:0 **JOFFSETx[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_JDRx registers.

14.13.7 ADC watchdog higher threshold register (ADC_HTR)

Address offset: 0x24

Reset value: 0x0000 0FFF

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

Bits 31:12 Reserved, must be kept at reset value.

Bits 11:0 **HT[11:0]**: Analog watchdog higher threshold

These bits are written by software to define the higher threshold for the analog watchdog.

Note: *The software can write to these registers when an ADC conversion is ongoing. The programmed value will be effective when the next conversion is complete. Writing to this register is performed with a write delay that can create uncertainty on the effective time at which the new value is programmed.*

14.13.8 ADC watchdog lower 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
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	LT[11:0]											
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:12 Reserved, must be kept at reset value.

Bits 11:0 **LT[11:0]**: Analog watchdog lower threshold

These bits are written by software to define the lower threshold for the analog watchdog.

Note: *The software can write to these registers when an ADC conversion is ongoing. The programmed value will be effective when the next conversion is complete. Writing to this register is performed with a write delay that can create uncertainty on the effective time at which the new value is programmed.*

14.13.9 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
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	L[3:0]				SQ16[4:1]			
								rw	rw	rw	rw	rw	rw	rw	rw
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 at reset value.

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..18) 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

14.13.10 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	
Res.	Res.	SQ12[4:0]					SQ11[4:0]					SQ10[4:1]				
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
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 at reset value.

Bits 29:26 **SQ12[4:0]**: 12th conversion in regular sequence

These bits are written by software with the channel number (0..18) 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

14.13.11 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
Res.	Res.	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 at reset value.

Bits 29:25 **SQ6[4:0]**: 6th conversion in regular sequence

These bits are written by software with the channel number (0..18) 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

14.13.12 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
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	JL[1:0]	JSQ4[4:1]				
										rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
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 at reset value.

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 (when JL[1:0]=3, see note below)

These bits are written by software with the channel number (0..18) assigned as the 4th in the sequence to be converted.

Bits 14:10 **JSQ3[4:0]:** 3rd conversion in injected sequence (when JL[1:0]=3, see note below)

Bits 9:5 **JSQ2[4:0]:** 2nd conversion in injected sequence (when JL[1:0]=3, see note below)

Bits 4:0 **JSQ1[4:0]:** 1st conversion in injected sequence (when JL[1:0]=3, see note below)

Note: When JL[1:0]=3 (4 injected conversions in the sequencer), the ADC converts the channels in the following order: JSQ1[4:0], JSQ2[4:0], JSQ3[4:0], and JSQ4[4:0].

When JL=2 (3 injected conversions in the sequencer), the ADC converts the channels in the following order: JSQ2[4:0], JSQ3[4:0], and JSQ4[4:0].

When JL=1 (2 injected conversions in the sequencer), the ADC converts the channels in starting from JSQ3[4:0], and then JSQ4[4:0].

When JL=0 (1 injected conversion in the sequencer), the ADC converts only JSQ4[4:0] channel.

14.13.13 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
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
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 at reset value.

Bits 15:0 **JDATA[15:0]**: Injected data

These bits are read-only. They contain the conversion result from injected channel x. The data are left -or right-aligned as shown in [Figure 77](#) and [Figure 78](#).

14.13.14 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
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
DATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

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

Bits 15:0 **DATA[15:0]**: Regular data

These bits are read-only. They contain the conversion result from the regular channels. The data are left- or right-aligned as shown in [Figure 77](#) and [Figure 78](#).

14.13.15 ADC Common status register (ADC_CSR)

Address offset: 0x00 (this offset address is relative to ADC1 base address + 0x300)

Reset value: 0x0000 0000

This register provides an image of the status bits of the different ADCs. Nevertheless it is read-only and does not allow to clear the different status bits. Instead each status bit must be cleared by writing it to 0 in the corresponding ADC_SR register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OVR3	STRT3	JSTRT3	JEOC 3	EOC3	AWD3
										ADC3					
										r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	OVR2	STRT2	JSTRT2	JEOC2	EOC2	AWD2	Res.	Res.	OVR1	STRT1	JSTRT1	JEOC 1	EOC1	AWD1
		ADC2								ADC1					
		r	r	r	r	r	r			r	r	r	r	r	r

Bits 31:22 Reserved, must be kept at reset value.

Bit 21 **OVR3**: Overrun flag of ADC3

This bit is a copy of the OVR bit in the ADC3_SR register.

Bit 20 **STRT3**: Regular channel Start flag of ADC3

This bit is a copy of the STRT bit in the ADC3_SR register.

- Bit 19 **JSTRT3:** Injected channel Start flag of ADC3
 This bit is a copy of the JSTRT bit in the ADC3_SR register.
- Bit 18 **JEOC3:** Injected channel end of conversion of ADC3
 This bit is a copy of the JEOC bit in the ADC3_SR register.
- Bit 17 **EOC3:** End of conversion of ADC3
 This bit is a copy of the EOC bit in the ADC3_SR register.
- Bit 16 **AWD3:** Analog watchdog flag of ADC3
 This bit is a copy of the AWD bit in the ADC3_SR register.
- Bits 15:14 Reserved, must be kept at reset value.
- Bit 13 **OVR2:** Overrun flag of ADC2
 This bit is a copy of the OVR bit in the ADC2_SR register.
- Bit 12 **STRT2:** Regular channel Start flag of ADC2
 This bit is a copy of the STRT bit in the ADC2_SR register.
- Bit 11 **JSTRT2:** Injected channel Start flag of ADC2
 This bit is a copy of the JSTRT bit in the ADC2_SR register.
- Bit 10 **JEOC2:** Injected channel end of conversion of ADC2
 This bit is a copy of the JEOC bit in the ADC2_SR register.
- Bit 9 **EOC2:** End of conversion of ADC2
 This bit is a copy of the EOC bit in the ADC2_SR register.
- Bit 8 **AWD2:** Analog watchdog flag of ADC2
 This bit is a copy of the AWD bit in the ADC2_SR register.
- Bits 7:6 Reserved, must be kept at reset value.
- Bit 5 **OVR1:** Overrun flag of ADC1
 This bit is a copy of the OVR bit in the ADC1_SR register.
- Bit 4 **STRT1:** Regular channel Start flag of ADC1
 This bit is a copy of the STRT bit in the ADC1_SR register.
- Bit 3 **JSTRT1:** Injected channel Start flag of ADC1
 This bit is a copy of the JSTRT bit in the ADC1_SR register.
- Bit 2 **JEOC1:** Injected channel end of conversion of ADC1
 This bit is a copy of the JEOC bit in the ADC1_SR register.
- Bit 1 **EOC1:** End of conversion of ADC1
 This bit is a copy of the EOC bit in the ADC1_SR register.
- Bit 0 **AWD1:** Analog watchdog flag of ADC1
 This bit is a copy of the AWD bit in the ADC1_SR register.

14.13.16 ADC common control register (ADC_CCR)

Address offset: 0x04 (this offset address is relative to ADC1 base address + 0x300)

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TSVREFE	VBATE	Res.	Res.	Res.	Res.	ADCPRE	
								rw	rw					rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMA[1:0]	DDS	Res.	DELAY[3:0]				Res.	Res.	Res.	MULTI[4:0]					
rw	rw	rw	rw	rw	rw	rw				rw	rw	rw	rw	rw	rw

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

Bit 23 **TSVREFE**: Temperature sensor and V_{REFINT} enable

This bit is set and cleared by software to enable/disable the temperature sensor and the V_{REFINT} channel.

- 0: Temperature sensor and V_{REFINT} channel disabled
- 1: Temperature sensor and V_{REFINT} channel enabled

Note: VBATE must be disabled when TSVREFE is set. If both bits are set, only the VBAT conversion is performed.

Bit 22 **VBATE**: V_{BAT} enable

This bit is set and cleared by software to enable/disable the V_{BAT} channel.

- 0: V_{BAT} channel disabled
- 1: V_{BAT} channel enabled

Bits 21:18 Reserved, must be kept at reset value.

Bits 17:16 **ADCPRE**: ADC prescaler

Set and cleared by software to select the frequency of the clock to the ADC. The clock is common for all the ADCs.

Note: 00: PCLK2 divided by 2
 01: PCLK2 divided by 4
 10: PCLK2 divided by 6
 11: PCLK2 divided by 8

Bits 15:14 **DMA**: Direct memory access mode for multi ADC mode

This bit-field is set and cleared by software. Refer to the DMA controller section for more details.

- 00: DMA mode disabled
- 01: DMA mode 1 enabled (2 / 3 half-words one by one - 1 then 2 then 3)
- 10: DMA mode 2 enabled (2 / 3 half-words by pairs - 2&1 then 1&3 then 3&2)
- 11: DMA mode 3 enabled (2 / 3 bytes by pairs - 2&1 then 1&3 then 3&2)

Bit 13 **DDS**: DMA disable selection (for multi-ADC mode)

This bit is set and cleared by software.

- 0: No new DMA request is issued after the last transfer (as configured in the DMA controller). DMA bits are not cleared by hardware, however they must have been cleared and set to the wanted mode by software before new DMA requests can be generated.
- 1: DMA requests are issued as long as data are converted and DMA = 01, 10 or 11.

Bit 12 Reserved, must be kept at reset value.

Bits 11:8 **DELAY:** Delay between 2 sampling phases

Set and cleared by software. These bits are used in dual or triple interleaved modes.

0000: 5 * T_{ADCCLK}

0001: 6 * T_{ADCCLK}

0010: 7 * T_{ADCCLK}

...

1111: 20 * T_{ADCCLK}

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

Bits 4:0 **MULTI[4:0]:** Multi ADC mode selection

These bits are written by software to select the operating mode.

- All the ADCs independent:

00000: Independent mode

- 00001 to 01001: Dual mode, ADC1 and ADC2 working together, ADC3 is independent

00001: Combined regular simultaneous + injected simultaneous mode

00010: Combined regular simultaneous + alternate trigger mode

00011: Reserved

00101: Injected simultaneous mode only

00110: Regular simultaneous mode only

00111: interleaved mode only

01001: Alternate trigger mode only

- 10001 to 11001: Triple mode: ADC1, 2 and 3 working together

10001: Combined regular simultaneous + injected simultaneous mode

10010: Combined regular simultaneous + alternate trigger mode

10011: Reserved

10101: Injected simultaneous mode only

10110: Regular simultaneous mode only

10111: interleaved mode only

11001: Alternate trigger mode only

All other combinations are reserved and must not be programmed

Note: In multi mode, a change of channel configuration generates an abort that can cause a loss of synchronization. It is recommended to disable the multi ADC mode before any configuration change.

14.13.17 ADC common regular data register for dual and triple modes (ADC_CDR)

Address offset: 0x08 (this offset address is relative to ADC1 base address + 0x300)

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA2[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
DATA1[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 **DATA2[15:0]**: 2nd data item of a pair of regular conversions

- In dual mode, these bits contain the regular data of ADC2. Refer to [Dual ADC mode](#).
- In triple mode, these bits contain alternatively the regular data of ADC2, ADC1 and ADC3. Refer to [Triple ADC mode](#).

Bits 15:0 **DATA1[15:0]**: 1st data item of a pair of regular conversions

- In dual mode, these bits contain the regular data of ADC1. Refer to [Dual ADC mode](#).
- In triple mode, these bits contain alternatively the regular data of ADC1, ADC3 and ADC2. Refer to [Triple ADC mode](#).

14.13.18 ADC register map

The following table summarizes the ADC registers.

Table 91. ADC global register map

Offset	Register
0x000 - 0x04C	ADC1
0x050 - 0x0FC	Reserved
0x100 - 0x14C	ADC2
0x118 - 0x1FC	Reserved
0x200 - 0x24C	ADC3
0x250 - 0x2FC	Reserved
0x300 - 0x308	Common registers

Table 92. ADC register map and reset values for each ADC

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x00	ADC_SR	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	OVR	5
	Reset value	—	—	—	—	—	—	—	—	—	—	—	—	—	—	STRTR	4
0x04	ADC_CR1	Res	Res	Res	Res	OVRIE	RES[1:0]	AWDEN	JAWDEN	Res	Res	Res	Res	Res	DISC NUM [2:0]	DISCEN	Res
	Reset value	—	—	—	—	0	0	0	0	—	—	—	—	—	0 0 0	0 0 0	9
															JAUTO	Res	10
															AWD_SGL	Res	8
															SCAN	Res	7
															JEOCIE	Res	6
															AWDIE	Res	5
															EOCIE	0	4
															AWDCH[4:0]	JSTRT	3
															0 0 0 0 0	JEOC	2
															0 0 0 0 0	EOC	1
															0 0 0 0 0	AWD	0

Table 92. ADC register map and reset values for each ADC (continued)

Table 93. ADC register map and reset values (common ADC registers)

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_CSR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OVR	STRT	JSTRT	JEOC	EOC	AWD	Res.	Res.	Res.	Res.															
												0	0	0	0	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
	-																																			
0x04	ADC_CCR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
0x08	ADC_CDR	Regular DATA2[15:0]															Regular DATA1[15:0]															MULTI [4:0]				
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

15 Digital-to-analog converter (DAC)

15.1 DAC introduction

The DAC module is a 12-bit, voltage output digital-to-analog converter. The DAC can be configured in 8- or 12-bit mode and may be used in conjunction with the DMA controller. In 12-bit mode, the data could be left- or right-aligned. The DAC has two output channels, each with its own converter. In dual DAC channel mode, conversions could be done independently or simultaneously when both channels are grouped together for synchronous update operations. An input reference pin, V_{REF+} (shared with ADC) is available for better resolution.

15.2 DAC main features

- Two DAC converters: one output channel each
- Left or right data alignment in 12-bit mode
- Synchronized update capability
- Noise-wave generation
- Triangular-wave generation
- Dual DAC channel for independent or simultaneous conversions
- DMA capability for each channel
- DMA underrun error detection
- External triggers for conversion
- Input voltage reference, V_{REF+}

Figure 93 shows the block diagram of a DAC channel and *Table 94* gives the pin description.

Figure 93. DAC channel block diagram

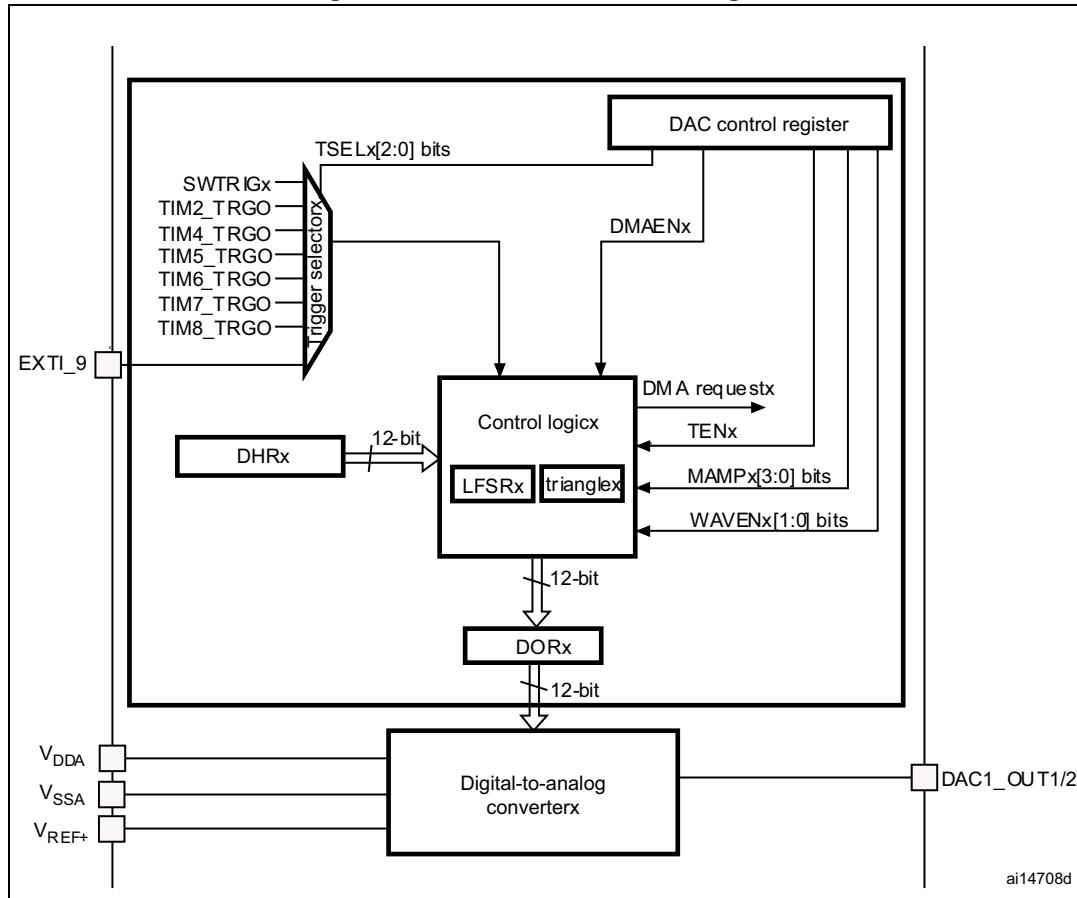


Table 94. DAC pins

Name	Signal type	Remarks
V _{REF+}	Input, analog reference positive	The higher/positive reference voltage for the DAC, $1.8 \text{ V} \leq V_{\text{REF}+} \leq V_{\text{DDA}}$
V _{DDA}	Input, analog supply	Analog power supply
V _{SSA}	Input, analog supply ground	Ground for analog power supply
DAC _{_OUTx}	Analog output signal	DAC channelx analog output

Note: Once the DAC channelx is enabled, the corresponding GPIO pin (PA4 or PA5) is automatically connected to the analog converter output (DAC_{_OUTx}). In order to avoid parasitic consumption, the PA4 or PA5 pin should first be configured to analog (AIN).

15.3 DAC functional description

15.3.1 DAC channel enable

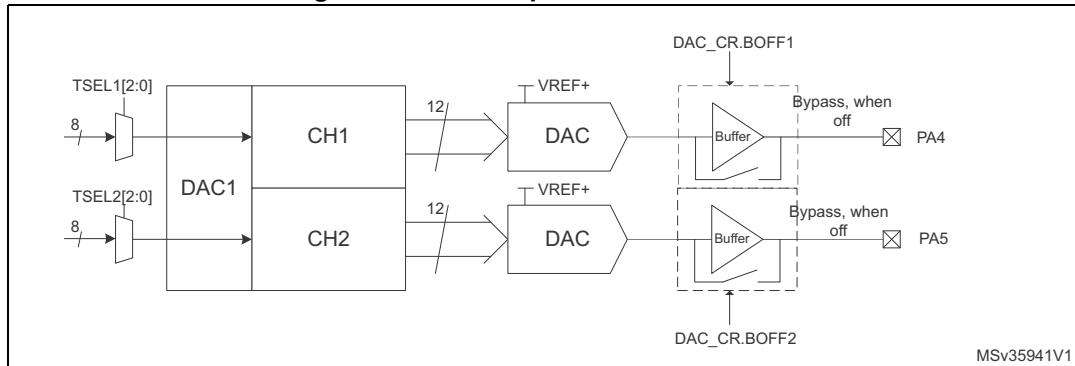
Each DAC channel can be powered on by setting its corresponding ENx bit in the DAC_CR register. The DAC channel is then enabled after a startup time t_{WAKEUP} .

Note: The ENx bit enables the analog DAC Channelx macrocell only. The DAC Channelx digital interface is enabled even if the ENx bit is reset.

15.3.2 DAC output buffer enable

The DAC integrates two output buffers that can be used to reduce the output impedance, and to drive external loads directly without having to add an external operational amplifier. Each DAC channel output buffer can be enabled and disabled using the corresponding BOFFx bit in the DAC_CR register.

Figure 94. DAC output buffer connection

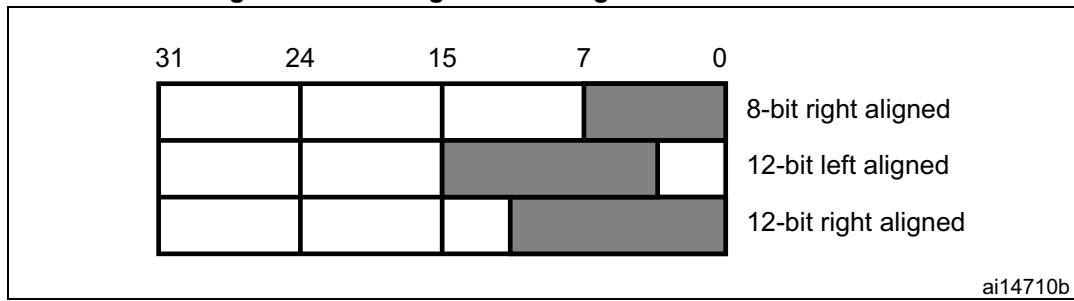


15.3.3 DAC data format

Depending on the selected configuration mode, the data have to be written into the specified register as described below:

- Single DAC channelx, there are three possibilities:
 - 8-bit right alignment: the software has to load data into the DAC_DHR8Rx [7:0] bits (stored into the DHRx[11:4] bits)
 - 12-bit left alignment: the software has to load data into the DAC_DHR12Lx [15:4] bits (stored into the DHRx[11:0] bits)
 - 12-bit right alignment: the software has to load data into the DAC_DHR12Rx [11:0] bits (stored into the DHRx[11:0] bits)

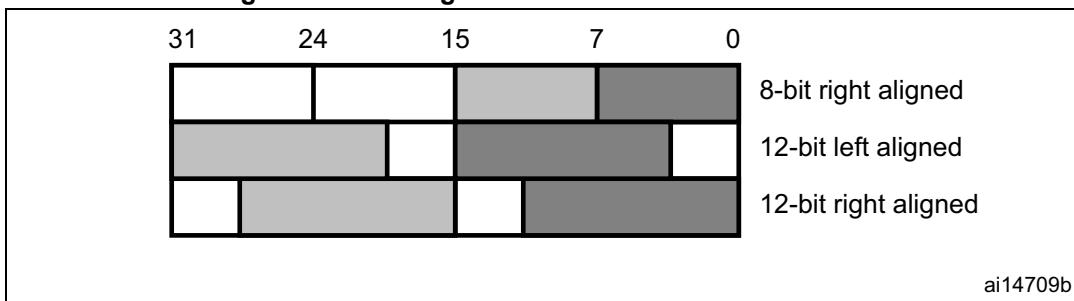
Depending on the loaded DAC_DHRyyxx register, the data written by the user is shifted and stored into the corresponding DHRx (data holding registerx, which are internal non-memory-mapped registers). The DHRx register is then loaded into the DORx register either automatically, by software trigger or by an external event trigger.

Figure 95. Data registers in single DAC channel mode

ai14710b

- Dual DAC channels, there are three possibilities:
 - 8-bit right alignment: data for DAC channel1 to be loaded into the DAC_DHR8RD [7:0] bits (stored into the DHR1[11:4] bits) and data for DAC channel2 to be loaded into the DAC_DHR8RD [15:8] bits (stored into the DHR2[11:4] bits)
 - 12-bit left alignment: data for DAC channel1 to be loaded into the DAC_DHR12LD [15:4] bits (stored into the DHR1[11:0] bits) and data for DAC channel2 to be loaded into the DAC_DHR12LD [31:20] bits (stored into the DHR2[11:0] bits)
 - 12-bit right alignment: data for DAC channel1 to be loaded into the DAC_DHR12RD [11:0] bits (stored into the DHR1[11:0] bits) and data for DAC channel2 to be loaded into the DAC_DHR12LD [27:16] bits (stored into the DHR2[11:0] bits)

Depending on the loaded DAC_DHRyyD register, the data written by the user is shifted and stored into DHR1 and DHR2 (data holding registers, which are internal non-memory-mapped registers). The DHR1 and DHR2 registers are then loaded into the DOR1 and DOR2 registers, respectively, either automatically, by software trigger or by an external event trigger.

Figure 96. Data registers in dual DAC channel mode

ai14709b

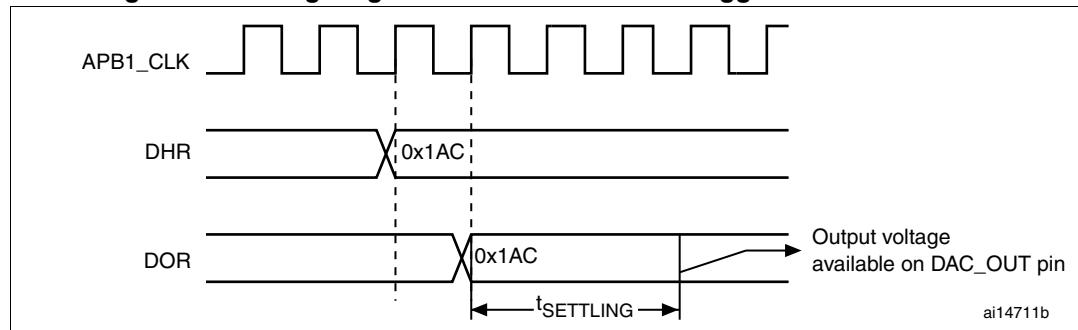
15.3.4 DAC conversion

The DAC_DORx cannot be written directly and any data transfer to the DAC channelx must be performed by loading the DAC_DHRx register (write to DAC_DHR8Rx, DAC_DHR12Lx, DAC_DHR12Rx, DAC_DHR8RD, DAC_DHR12LD or DAC_DHR12RD).

Data stored in the DAC_DHRx register are automatically transferred to the DAC_DORx register after one APB1 clock cycle, if no hardware trigger is selected (TENx bit in DAC_CR register is reset). However, when a hardware trigger is selected (TENx bit in DAC_CR register is set) and a trigger occurs, the transfer is performed three APB1 clock cycles later.

When DAC_DORx is loaded with the DAC_DHRx contents, the analog output voltage becomes available after a time $t_{SETTLING}$ that depends on the power supply voltage and the analog output load.

Figure 97. Timing diagram for conversion with trigger disabled TEN = 0



15.3.5 DAC output voltage

Digital inputs are converted to output voltages on a linear conversion between 0 and V_{REF+} .

The analog output voltages on each DAC channel pin are determined by the following equation:

$$\text{DACOutput} = V_{REF} \times \frac{\text{DOR}}{4096}$$

15.3.6 DAC trigger selection

If the TENx control bit is set, conversion can then be triggered by an external event (timer counter, external interrupt line). The TSELx[2:0] control bits determine which out of 8 possible events will trigger conversion as shown in [Table 95](#).

Table 95. External triggers

Source	Type	TSEL[2:0]
Timer 6 TRGO event	Internal signal from on-chip timers	000
Timer 8 TRGO event		001
Timer 7 TRGO event		010
Timer 5 TRGO event		011
Timer 2 TRGO event		100
Timer 4 TRGO event		101
EXTI line9	External pin	110
SWTRIG	Software control bit	111

Each time a DAC interface detects a rising edge on the selected timer TRGO output, or on the selected external interrupt line 9, the last data stored into the DAC_DHRx register are transferred into the DAC_DORx register. The DAC_DORx register is updated three APB1 cycles after the trigger occurs.

If the software trigger is selected, the conversion starts once the SWTRIG bit is set. SWTRIG is reset by hardware once the DAC_DORx register has been loaded with the DAC_DHRx register contents.

Note: TSELx[2:0] bit cannot be changed when the ENx bit is set.

When software trigger is selected, the transfer from the DAC_DHRx register to the DAC_DORx register takes only one APB1 clock cycle.

15.3.7 DMA request

Each DAC channel has a DMA capability. Two DMA channels are used to service DAC channel DMA requests.

A DAC DMA request is generated when an external trigger (but not a software trigger) occurs while the DMAENx bit is set. The value of the DAC_DHRx register is then transferred into the DAC_DORx register.

In dual mode, if both DMAENx bits are set, two DMA requests are generated. If only one DMA request is needed, you should set only the corresponding DMAENx bit. In this way, the application can manage both DAC channels in dual mode by using one DMA request and a unique DMA channel.

DMA underrun

The DAC DMA request is not queued so that if a second external trigger arrives before the acknowledgement for the first external trigger is received (first request), then no new request is issued and the DMA channelx underrun flag DMAUDRx in the DAC_SR register is set, reporting the error condition. DMA data transfers are then disabled and no further DMA request is treated. The DAC channelx continues to convert old data.

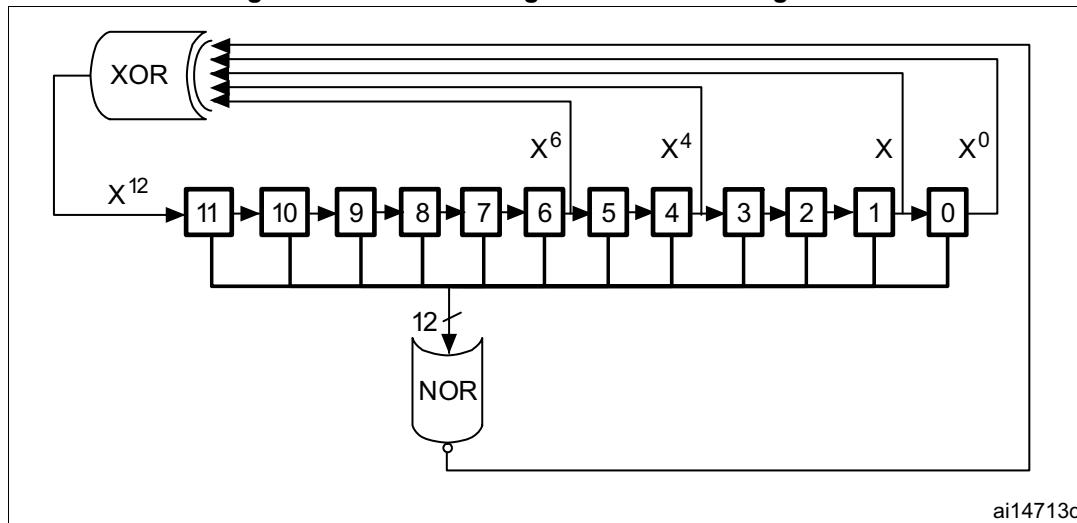
The software should clear the DMAUDRx flag by writing “1”, clear the DMAEN bit of the used DMA stream and re-initialize both DMA and DAC channelx to restart the transfer correctly. The software should modify the DAC trigger conversion frequency or lighten the DMA workload to avoid a new DMA underrun. Finally, the DAC conversion could be resumed by enabling both DMA data transfer and conversion trigger.

For each DAC channelx, an interrupt is also generated if its corresponding DMAUDRIEx bit in the DAC_CR register is enabled.

15.3.8 Noise generation

In order to generate a variable-amplitude pseudonoise, an LFSR (linear feedback shift register) is available. DAC noise generation is selected by setting WAVEEx[1:0] to “01”. The preloaded value in LFSR is 0xAAA. This register is updated three APB1 clock cycles after each trigger event, following a specific calculation algorithm.

Figure 98. DAC LFSR register calculation algorithm

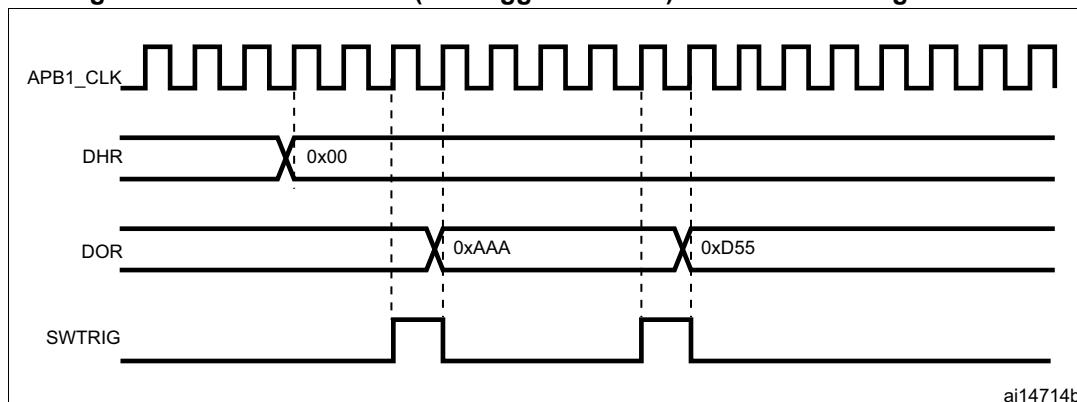


The LFSR value, that may be masked partially or totally by means of the MAMPx[3:0] bits in the DAC_CR register, is added up to the DAC_DHRx contents without overflow and this value is then stored into the DAC_DORx register.

If LFSR is 0x0000, a ‘1’ is injected into it (antilock-up mechanism).

It is possible to reset LFSR wave generation by resetting the WAVE[1:0] bits.

Figure 99. DAC conversion (SW trigger enabled) with LFSR wave generation



Note: The DAC trigger must be enabled for noise generation by setting the TENx bit in the DAC_CR register.

15.3.9 Triangle-wave generation

It is possible to add a small-amplitude triangular waveform on a DC or slowly varying signal. DAC triangle-wave generation is selected by setting WAVE[1:0] to “10”. The amplitude is configured through the MAMPx[3:0] bits in the DAC_CR register. An internal triangle counter is incremented three APB1 clock cycles after each trigger event. The value of this counter is then added to the DAC_DHRx register without overflow and the sum is stored into the DAC_DORx register. The triangle counter is incremented as long as it is less than the maximum amplitude defined by the MAMPx[3:0] bits. Once the configured amplitude is reached, the counter is decremented down to 0, then incremented again and so on.

It is possible to reset triangle wave generation by resetting the WAVEx[1:0] bits.

Figure 100. DAC triangle wave generation

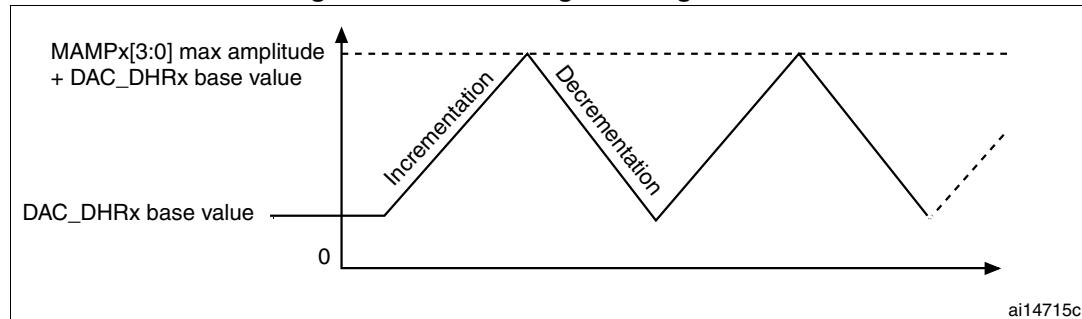
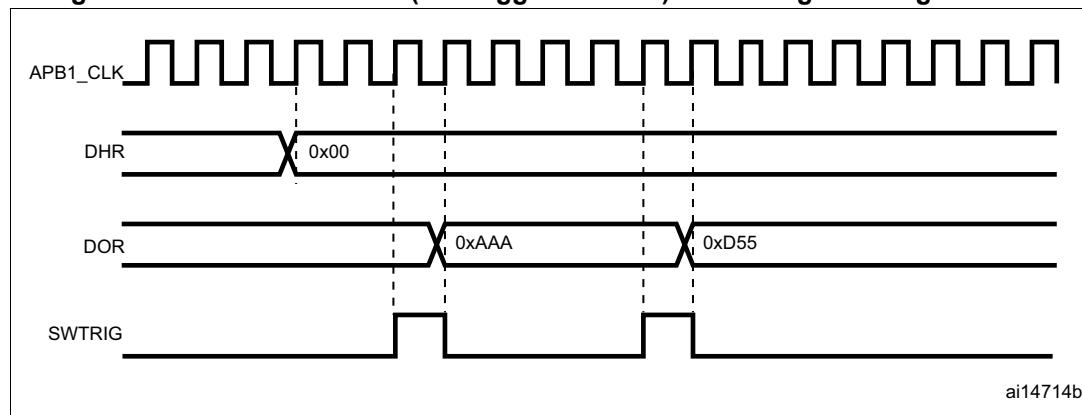


Figure 101. DAC conversion (SW trigger enabled) with triangle wave generation



Note: The DAC trigger must be enabled for noise generation by setting the TENx bit in the DAC_CR register.

The MAMPx[3:0] bits must be configured before enabling the DAC, otherwise they cannot be changed.

15.4 Dual DAC channel conversion

To efficiently use the bus bandwidth in applications that require the two DAC channels at the same time, three dual registers are implemented: DHR8RD, DHR12RD and DHR12LD. A unique register access is then required to drive both DAC channels at the same time.

Eleven possible conversion modes are possible using the two DAC channels and these dual registers. All the conversion modes can nevertheless be obtained using separate DHRx registers if needed.

All modes are described in the paragraphs below.

15.4.1 Independent trigger without wave generation

To configure the DAC in this conversion mode, the following sequence is required:

- Set the two DAC channel trigger enable bits TEN1 and TEN2
- Configure different trigger sources by setting different values in the TSEL1[2:0] and TSEL2[2:0] bits
- Load the dual DAC channel data into the desired DHR register (DAC_DHR12RD, DAC_DHR12LD or DAC_DHR8RD)

When a DAC channel1 trigger arrives, the DHR1 register is transferred into DAC_DOR1 (three APB1 clock cycles later).

When a DAC channel2 trigger arrives, the DHR2 register is transferred into DAC_DOR2 (three APB1 clock cycles later).

15.4.2 Independent trigger with single LFSR generation

To configure the DAC in this conversion mode, the following sequence is required:

- Set the two DAC channel trigger enable bits TEN1 and TEN2
- Configure different trigger sources by setting different values in the TSEL1[2:0] and TSEL2[2:0] bits
- Configure the two DAC channel WAVEx[1:0] bits as “01” and the same LFSR mask value in the MAMPx[3:0] bits
- Load the dual DAC channel data into the desired DHR register (DHR12RD, DHR12LD or DHR8RD)

When a DAC channel1 trigger arrives, the LFSR1 counter, with the same mask, is added to the DHR1 register and the sum is transferred into DAC_DOR1 (three APB1 clock cycles later). Then the LFSR1 counter is updated.

When a DAC channel2 trigger arrives, the LFSR2 counter, with the same mask, is added to the DHR2 register and the sum is transferred into DAC_DOR2 (three APB1 clock cycles later). Then the LFSR2 counter is updated.

15.4.3 Independent trigger with different LFSR generation

To configure the DAC in this conversion mode, the following sequence is required:

- Set the two DAC channel trigger enable bits TEN1 and TEN2
- Configure different trigger sources by setting different values in the TSEL1[2:0] and TSEL2[2:0] bits
- Configure the two DAC channel WAVEx[1:0] bits as “01” and set different LFSR masks values in the MAMP1[3:0] and MAMP2[3:0] bits
- Load the dual DAC channel data into the desired DHR register (DAC_DHR12RD, DAC_DHR12LD or DAC_DHR8RD)

When a DAC channel1 trigger arrives, the LFSR1 counter, with the mask configured by MAMP1[3:0], is added to the DHR1 register and the sum is transferred into DAC_DOR1 (three APB1 clock cycles later). Then the LFSR1 counter is updated.

When a DAC channel2 trigger arrives, the LFSR2 counter, with the mask configured by MAMP2[3:0], is added to the DHR2 register and the sum is transferred into DAC_DOR2 (three APB1 clock cycles later). Then the LFSR2 counter is updated.

15.4.4 Independent trigger with single triangle generation

To configure the DAC in this conversion mode, the following sequence is required:

- Set the two DAC channel trigger enable bits TEN1 and TEN2
- Configure different trigger sources by setting different values in the TSEL1[2:0] and TSEL2[2:0] bits
- Configure the two DAC channel WAVEx[1:0] bits as “1x” and the same maximum amplitude value in the MAMPx[3:0] bits
- Load the dual DAC channel data into the desired DHR register (DAC_DHR12RD, DAC_DHR12LD or DAC_DHR8RD)

When a DAC channel1 trigger arrives, the DAC channel1 triangle counter, with the same triangle amplitude, is added to the DHR1 register and the sum is transferred into DAC_DOR1 (three APB1 clock cycles later). The DAC channel1 triangle counter is then updated.

When a DAC channel2 trigger arrives, the DAC channel2 triangle counter, with the same triangle amplitude, is added to the DHR2 register and the sum is transferred into DAC_DOR2 (three APB1 clock cycles later). The DAC channel2 triangle counter is then updated.

15.4.5 Independent trigger with different triangle generation

To configure the DAC in this conversion mode, the following sequence is required:

- Set the two DAC channel trigger enable bits TEN1 and TEN2
- Configure different trigger sources by setting different values in the TSEL1[2:0] and TSEL2[2:0] bits
- Configure the two DAC channel WAVEx[1:0] bits as “1x” and set different maximum amplitude values in the MAMP1[3:0] and MAMP2[3:0] bits
- Load the dual DAC channel data into the desired DHR register (DAC_DHR12RD, DAC_DHR12LD or DAC_DHR8RD)

When a DAC channel1 trigger arrives, the DAC channel1 triangle counter, with a triangle amplitude configured by MAMP1[3:0], is added to the DHR1 register and the sum is transferred into DAC_DOR1 (three APB1 clock cycles later). The DAC channel1 triangle counter is then updated.

When a DAC channel2 trigger arrives, the DAC channel2 triangle counter, with a triangle amplitude configured by MAMP2[3:0], is added to the DHR2 register and the sum is transferred into DAC_DOR2 (three APB1 clock cycles later). The DAC channel2 triangle counter is then updated.

15.4.6 Simultaneous software start

To configure the DAC in this conversion mode, the following sequence is required:

- Load the dual DAC channel data to the desired DHR register (DAC_DHR12RD, DAC_DHR12LD or DAC_DHR8RD)

In this configuration, one APB1 clock cycle later, the DHR1 and DHR2 registers are transferred into DAC_DOR1 and DAC_DOR2, respectively.

15.4.7 Simultaneous trigger without wave generation

To configure the DAC in this conversion mode, the following sequence is required:

- Set the two DAC channel trigger enable bits TEN1 and TEN2
- Configure the same trigger source for both DAC channels by setting the same value in the TSEL1[2:0] and TSEL2[2:0] bits
- Load the dual DAC channel data to the desired DHR register (DAC_DHR12RD, DAC_DHR12LD or DAC_DHR8RD)

When a trigger arrives, the DHR1 and DHR2 registers are transferred into DAC_DOR1 and DAC_DOR2, respectively (after three APB1 clock cycles).

15.4.8 Simultaneous trigger with single LFSR generation

To configure the DAC in this conversion mode, the following sequence is required:

- Set the two DAC channel trigger enable bits TEN1 and TEN2
- Configure the same trigger source for both DAC channels by setting the same value in the TSEL1[2:0] and TSEL2[2:0] bits
- Configure the two DAC channel WAVEx[1:0] bits as “01” and the same LFSR mask value in the MAMPx[3:0] bits
- Load the dual DAC channel data to the desired DHR register (DHR12RD, DHR12LD or DHR8RD)

When a trigger arrives, the LFSR1 counter, with the same mask, is added to the DHR1 register and the sum is transferred into DAC_DOR1 (three APB1 clock cycles later). The LFSR1 counter is then updated. At the same time, the LFSR2 counter, with the same mask, is added to the DHR2 register and the sum is transferred into DAC_DOR2 (three APB1 clock cycles later). The LFSR2 counter is then updated.

15.4.9 Simultaneous trigger with different LFSR generation

To configure the DAC in this conversion mode, the following sequence is required:

- Set the two DAC channel trigger enable bits TEN1 and TEN2
- Configure the same trigger source for both DAC channels by setting the same value in the TSEL1[2:0] and TSEL2[2:0] bits
- Configure the two DAC channel WAVEx[1:0] bits as “01” and set different LFSR mask values using the MAMP1[3:0] and MAMP2[3:0] bits
- Load the dual DAC channel data into the desired DHR register (DAC_DHR12RD, DAC_DHR12LD or DAC_DHR8RD)

When a trigger arrives, the LFSR1 counter, with the mask configured by MAMP1[3:0], is added to the DHR1 register and the sum is transferred into DAC_DOR1 (three APB1 clock cycles later). The LFSR1 counter is then updated.

At the same time, the LFSR2 counter, with the mask configured by MAMP2[3:0], is added to the DHR2 register and the sum is transferred into DAC_DOR2 (three APB1 clock cycles later). The LFSR2 counter is then updated.

15.4.10 Simultaneous trigger with single triangle generation

To configure the DAC in this conversion mode, the following sequence is required:

- Set the two DAC channel trigger enable bits TEN1 and TEN2
- Configure the same trigger source for both DAC channels by setting the same value in the TSEL1[2:0] and TSEL2[2:0] bits
- Configure the two DAC channel WAVEx[1:0] bits as “1x” and the same maximum amplitude value using the MAMPx[3:0] bits
- Load the dual DAC channel data into the desired DHR register (DAC_DHR12RD, DAC_DHR12LD or DAC_DHR8RD)

When a trigger arrives, the DAC channel1 triangle counter, with the same triangle amplitude, is added to the DHR1 register and the sum is transferred into DAC_DOR1 (three APB1 clock cycles later). The DAC channel1 triangle counter is then updated.

At the same time, the DAC channel2 triangle counter, with the same triangle amplitude, is added to the DHR2 register and the sum is transferred into DAC_DOR2 (three APB1 clock cycles later). The DAC channel2 triangle counter is then updated.

15.4.11 Simultaneous trigger with different triangle generation

To configure the DAC in this conversion mode, the following sequence is required:

- Set the two DAC channel trigger enable bits TEN1 and TEN2
- Configure the same trigger source for both DAC channels by setting the same value in the TSEL1[2:0] and TSEL2[2:0] bits
- Configure the two DAC channel WAVEx[1:0] bits as “1x” and set different maximum amplitude values in the MAMP1[3:0] and MAMP2[3:0] bits
- Load the dual DAC channel data into the desired DHR register (DAC_DHR12RD, DAC_DHR12LD or DAC_DHR8RD)

When a trigger arrives, the DAC channel1 triangle counter, with a triangle amplitude configured by MAMP1[3:0], is added to the DHR1 register and the sum is transferred into DAC_DOR1 (three APB1 clock cycles later). Then the DAC channel1 triangle counter is updated.

At the same time, the DAC channel2 triangle counter, with a triangle amplitude configured by MAMP2[3:0], is added to the DHR2 register and the sum is transferred into DAC_DOR2 (three APB1 clock cycles later). Then the DAC channel2 triangle counter is updated.

15.5 DAC registers

Refer to [Section 1 on page 53](#) for a list of abbreviations used in register descriptions.

The peripheral registers have to be accessed by words (32 bits).

15.5.1 DAC control register (DAC_CR)

Address offset: 0x000

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Res.	Res.	DMAU DRIE2	DMA EN2	MAMP2[3:0]				WAVE2[1:0]		TSEL2[2:0]				TEN2	BOFF2	EN2
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	Res.	DMAU DRIE1	DMA EN1	MAMP1[3:0]				WAVE1[1:0]		TSEL1[2:0]				TEN1	BOFF1	EN1
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bits 31:30 Reserved, must be kept at reset value.

Bit 29 **DMAUDRIE2**: DAC channel2 DMA underrun interrupt enable

This bit is set and cleared by software.

0: DAC channel2 DMA underrun interrupt disabled

1: DAC channel2 DMA underrun interrupt enabled

Bit 28 **DMAEN2**: DAC channel2 DMA enable

This bit is set and cleared by software.

0: DAC channel2 DMA mode disabled

1: DAC channel2 DMA mode enabled

Bits 27:24 **MAMP2[3:0]**: DAC channel2 mask/amplitude selector

These bits are written by software to select mask in wave generation mode or amplitude in triangle generation mode.

0000: Unmask bit0 of LFSR/ triangle amplitude equal to 1

0001: Unmask bits[1:0] of LFSR/ triangle amplitude equal to 3

0010: Unmask bits[2:0] of LFSR/ triangle amplitude equal to 7

0011: Unmask bits[3:0] of LFSR/ triangle amplitude equal to 15

0100: Unmask bits[4:0] of LFSR/ triangle amplitude equal to 31

0101: Unmask bits[5:0] of LFSR/ triangle amplitude equal to 63

0110: Unmask bits[6:0] of LFSR/ triangle amplitude equal to 127

0111: Unmask bits[7:0] of LFSR/ triangle amplitude equal to 255

1000: Unmask bits[8:0] of LFSR/ triangle amplitude equal to 511

1001: Unmask bits[9:0] of LFSR/ triangle amplitude equal to 1023

1010: Unmask bits[10:0] of LFSR/ triangle amplitude equal to 2047

≥ 1011: Unmask bits[11:0] of LFSR/ triangle amplitude equal to 4095

Bits 23:22 **WAVE2[1:0]**: DAC channel2 noise/triangle wave generation enable

These bits are set/reset by software.

00: wave generation disabled

01: Noise wave generation enabled

1x: Triangle wave generation enabled

Note: Only used if bit TEN2 = 1 (DAC channel2 trigger enabled)

Bits 21:19 **TSEL2[2:0]**: DAC channel2 trigger selection

- These bits select the external event used to trigger DAC channel2
- 000: Timer 6 TRGO event
 - 001: Timer 8 TRGO event
 - 010: Timer 7 TRGO event
 - 011: Timer 5 TRGO event
 - 100: Timer 2 TRGO event
 - 101: Timer 4 TRGO event
 - 110: External line9
 - 111: Software trigger

Note: Only used if bit TEN2 = 1 (DAC channel2 trigger enabled).

Bit 18 **TEN2**: DAC channel2 trigger enable

- This bit is set and cleared by software to enable/disable DAC channel2 trigger
- 0: DAC channel2 trigger disabled and data written into the DAC_DHRx register are transferred one APB1 clock cycle later to the DAC_DOR2 register
 - 1: DAC channel2 trigger enabled and data from the DAC_DHRx register are transferred three APB1 clock cycles later to the DAC_DOR2 register

Note: When software trigger is selected, the transfer from the DAC_DHRx register to the DAC_DOR2 register takes only one APB1 clock cycle.

Bit 17 **BOFF2**: DAC channel2 output buffer disable

- This bit is set and cleared by software to enable/disable DAC channel2 output buffer.
- 0: DAC channel2 output buffer enabled
 - 1: DAC channel2 output buffer disabled

Bit 16 **EN2**: DAC channel2 enable

- This bit is set and cleared by software to enable/disable DAC channel2.
- 0: DAC channel2 disabled
 - 1: DAC channel2 enabled

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

Bit 13 **DMAUDRIE1**: DAC channel1 DMA Underrun Interrupt enable

- This bit is set and cleared by software.
- 0: DAC channel1 DMA Underrun Interrupt disabled
 - 1: DAC channel1 DMA Underrun Interrupt enabled

Bit 12 **DMAEN1**: DAC channel1 DMA enable

- This bit is set and cleared by software.
- 0: DAC channel1 DMA mode disabled
 - 1: DAC channel1 DMA mode enabled

Bits 11:8 **MAMP1[3:0]**: DAC channel1 mask/amplitude selector

These bits are written by software to select mask in wave generation mode or amplitude in triangle generation mode.

- 0000: Unmask bit0 of LFSR/ triangle amplitude equal to 1
- 0001: Unmask bits[1:0] of LFSR/ triangle amplitude equal to 3
- 0010: Unmask bits[2:0] of LFSR/ triangle amplitude equal to 7
- 0011: Unmask bits[3:0] of LFSR/ triangle amplitude equal to 15
- 0100: Unmask bits[4:0] of LFSR/ triangle amplitude equal to 31
- 0101: Unmask bits[5:0] of LFSR/ triangle amplitude equal to 63
- 0110: Unmask bits[6:0] of LFSR/ triangle amplitude equal to 127
- 0111: Unmask bits[7:0] of LFSR/ triangle amplitude equal to 255
- 1000: Unmask bits[8:0] of LFSR/ triangle amplitude equal to 511
- 1001: Unmask bits[9:0] of LFSR/ triangle amplitude equal to 1023
- 1010: Unmask bits[10:0] of LFSR/ triangle amplitude equal to 2047
- ≥ 1011 : Unmask bits[11:0] of LFSR/ triangle amplitude equal to 4095

Bits 7:6 **WAVE1[1:0]**: DAC channel1 noise/triangle wave generation enable

These bits are set and cleared by software.

- 00: wave generation disabled
- 01: Noise wave generation enabled
- 1x: Triangle wave generation enabled

Note: Only used if bit TEN1 = 1 (DAC channel1 trigger enabled).

Bits 5:3 **TSEL1[2:0]**: DAC channel1 trigger selection

These bits select the external event used to trigger DAC channel1.

- 000: Timer 6 TRGO event
- 001: Timer 8 TRGO event
- 010: Timer 7 TRGO event
- 011: Timer 5 TRGO event
- 100: Timer 2 TRGO event
- 101: Timer 4 TRGO event
- 110: External line9
- 111: Software trigger

Note: Only used if bit TEN1 = 1 (DAC channel1 trigger enabled).

Bit 2 **TEN1**: DAC channel1 trigger enable

This bit is set and cleared by software to enable/disable DAC channel1 trigger.

- 0: DAC channel1 trigger disabled and data written into the DAC_DHRx register are transferred one APB1 clock cycle later to the DAC_DOR1 register
- 1: DAC channel1 trigger enabled and data from the DAC_DHRx register are transferred three APB1 clock cycles later to the DAC_DOR1 register

Note: When software trigger is selected, the transfer from the DAC_DHRx register to the DAC_DOR1 register takes only one APB1 clock cycle.

Bit 1 **BOFF1**: DAC channel1 output buffer disable

This bit is set and cleared by software to enable/disable DAC channel1 output buffer.

- 0: DAC channel1 output buffer enabled
- 1: DAC channel1 output buffer disabled

Bit 0 **EN1**: DAC channel1 enable

This bit is set and cleared by software to enable/disable DAC channel1.

- 0: DAC channel1 disabled
- 1: DAC channel1 enabled

15.5.2 DAC software trigger register (DAC_SWTRIGR)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SWTRIG2	SWTRIG1													
														w	w

Bits 31:2 Reserved, must be kept at reset value.

Bit 1 **SWTRIG2**: DAC channel2 software trigger

This bit is set and cleared by software to enable/disable the software trigger.

0: Software trigger disabled

1: Software trigger enabled

Note: This bit is cleared by hardware (one APB1 clock cycle later) once the DAC_DHR2 register value has been loaded into the DAC_DOR2 register.

Bit 0 **SWTRIG1**: DAC channel1 software trigger

This bit is set and cleared by software to enable/disable the software trigger.

0: Software trigger disabled

1: Software trigger enabled

Note: This bit is cleared by hardware (one APB1 clock cycle later) once the DAC_DHR1 register value has been loaded into the DAC_DOR1 register.

15.5.3 DAC channel1 12-bit right-aligned data holding register (DAC_DHR12R1)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	DACC1DHR[11:0]											
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:12 Reserved, must be kept at reset value.

Bits 11:0 **DACC1DHR[11:0]**: DAC channel1 12-bit right-aligned data

These bits are written by software which specifies 12-bit data for DAC channel1.

15.5.4 DAC channel1 12-bit left aligned data holding register (DAC_DHR12L1)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DACC1DHR[11:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 15:4 **DACC1DHR[11:0]**: DAC channel1 12-bit left-aligned data

These bits are written by software which specifies 12-bit data for DAC channel1.

Bits 3:0 Reserved, must be kept at reset value.

15.5.5 DAC channel1 8-bit right aligned data holding register (DAC_DHR8R1)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	DACC1DHR[7:0]														
								rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 7:0 **DACC1DHR[7:0]**: DAC channel1 8-bit right-aligned data

These bits are written by software which specifies 8-bit data for DAC channel1.

15.5.6 DAC channel2 12-bit right aligned data holding register (DAC_DHR12R2)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DACC2DHR[11:0]															
Res.	Res.	Res.	Res.		rw										

Bits 31:12 Reserved, must be kept at reset value.

Bits 11:0 **DACC2DHR[11:0]**: DAC channel2 12-bit right-aligned data

These bits are written by software which specifies 12-bit data for DAC channel2.

15.5.7 DAC channel2 12-bit left aligned data holding register (DAC_DHR12L2)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DACC2DHR[11:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 15:4 **DACC2DHR[11:0]**: DAC channel2 12-bit left-aligned data

These bits are written by software which specify 12-bit data for DAC channel2.

Bits 3:0 Reserved, must be kept at reset value.

15.5.8 DAC channel2 8-bit right-aligned data holding register (DAC_DHR8R2)

Address offset: 0x1C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DACC2DHR[7:0]															
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		rw						

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

Bits 7:0 **DACC2DHR[7:0]**: DAC channel2 8-bit right-aligned data

These bits are written by software which specifies 8-bit data for DAC channel2.

15.5.9 Dual DAC 12-bit right-aligned data holding register (DAC_DHR12RD)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16			
Res.	Res.	Res.	Res.	DACC2DHR[11:0]														
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
Res.	Res.	Res.	Res.	DACC1DHR[11:0]														
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bits 31:28 Reserved, must be kept at reset value.

Bits 27:16 **DACC2DHR[11:0]**: DAC channel2 12-bit right-aligned data

These bits are written by software which specifies 12-bit data for DAC channel2.

Bits 15:12 Reserved, must be kept at reset value.

Bits 11:0 **DACC1DHR[11:0]**: DAC channel1 12-bit right-aligned data

These bits are written by software which specifies 12-bit data for DAC channel1.

15.5.10 DUAL DAC 12-bit left aligned data holding register (DAC_DHR12LD)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16			
DACC2DHR[11:0]															Res.	Res.	Res.	Res.
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
DACC1DHR[11:0]															Res.	Res.	Res.	Res.
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw							

Bits 31:20 **DACC2DHR[11:0]**: DAC channel2 12-bit left-aligned data

These bits are written by software which specifies 12-bit data for DAC channel2.

Bits 19:16 Reserved, must be kept at reset value.

Bits 15:4 **DACC1DHR[11:0]**: DAC channel1 12-bit left-aligned data

These bits are written by software which specifies 12-bit data for DAC channel1.

Bits 3:0 Reserved, must be kept at reset value.

15.5.11 DUAL DAC 8-bit right aligned data holding register (DAC_DHR8RD)

Address offset: 0x28

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DACC2DHR[7:0]								DACC1DHR[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 15:8 **DACC2DHR[7:0]**: DAC channel2 8-bit right-aligned data

These bits are written by software which specifies 8-bit data for DAC channel2.

Bits 7:0 **DACC1DHR[7:0]**: DAC channel1 8-bit right-aligned data

These bits are written by software which specifies 8-bit data for DAC channel1.

15.5.12 DAC channel1 data output register (DAC_DOR1)

Address offset: 0x2C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	DACC1DOR[11:0]											
				r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:12 Reserved, must be kept at reset value.

Bits 11:0 **DACC1DOR[11:0]**: DAC channel1 data output

These bits are read-only, they contain data output for DAC channel1.

15.5.13 DAC channel2 data output register (DAC_DOR2)

Address offset: 0x30

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	DACC2DOR[11:0]											
				r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:12 Reserved, must be kept at reset value.

Bits 11:0 **DACC2DOR[11:0]**: DAC channel2 data output

These bits are read-only, they contain data output for DAC channel2.

15.5.14 DAC status register (DAC_SR)

Address offset: 0x34

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	DMAUDR2	Res.												
		rc_w1													
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	DMAUDR1	Res.												
		rc_w1													

Bits 31:30 Reserved, must be kept at reset value.

Bit 29 **DMAUDR2**: DAC channel2 DMA underrun flag

This bit is set by hardware and cleared by software (by writing it to 1).

0: No DMA underrun error condition occurred for DAC channel2

1: DMA underrun error condition occurred for DAC channel2 (the currently selected trigger is driving DAC channel2 conversion at a frequency higher than the DMA service capability rate)

Bits 28:14 Reserved, must be kept at reset value.

Bit 13 **DMAUDR1**: DAC channel1 DMA underrun flag

This bit is set by hardware and cleared by software (by writing it to 1).

0: No DMA underrun error condition occurred for DAC channel1

1: DMA underrun error condition occurred for DAC channel1 (the currently selected trigger is driving DAC channel1 conversion at a frequency higher than the DMA service capability rate)

Bits 12:0 Reserved, must be kept at reset value.

15.5.15 DAC register map

Table 96 summarizes the DAC registers.

Table 96. DAC register map

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

16 True random number generator (RNG)

16.1 Introduction

The RNG is a true random number generator that continuously provides 32-bit entropy samples, based on an analog noise source. It can be used by the application as a live entropy source to build a NIST compliant Deterministic Random Bit Generator (DRBG).

The RNG true random number generator has been validated according to the German AIS-31 standard.

16.2 RNG main features

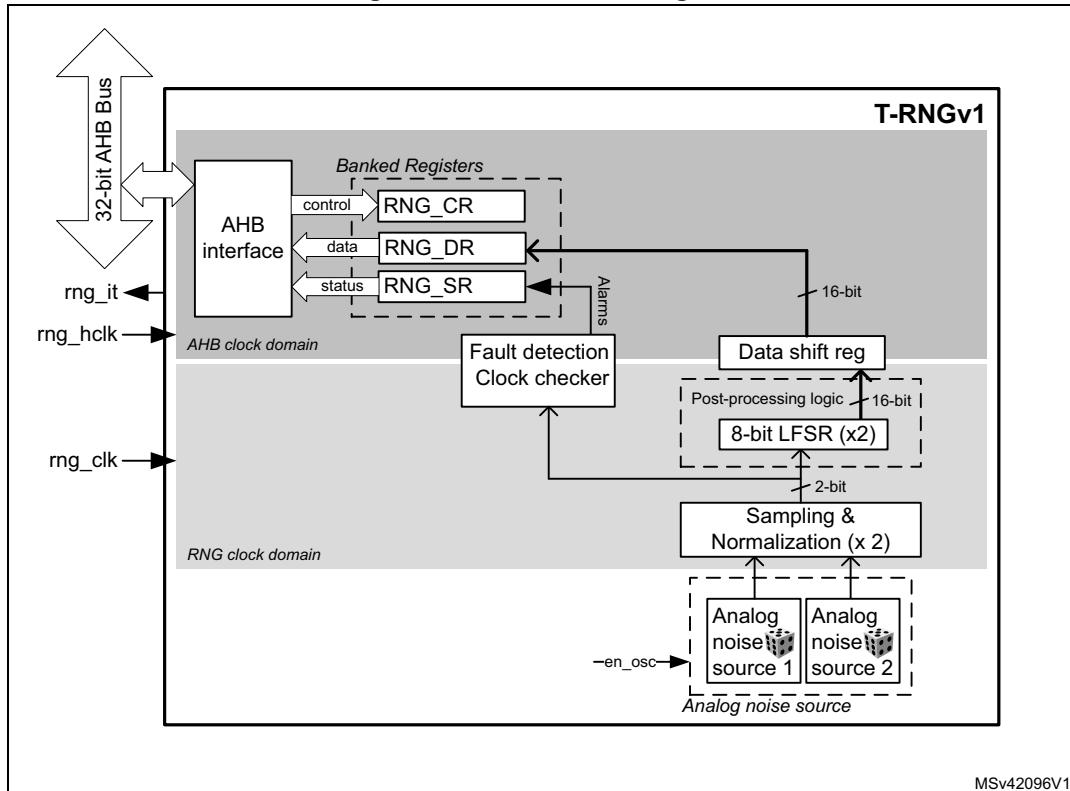
- The RNG delivers 32-bit true random numbers, produced by an analog entropy source post-processed with linear-feedback shift registers (LFSR).
- It is validated according to the AIS-31 pre-defined class PTG.2 evaluation methodology, which is part of the German Common Criteria (CC) scheme.
- It produces one 32-bit random samples every 42 RNG clock cycles (dedicated clock).
- It allows embedded continuous basic health tests with associated error management
 - Includes too low sampling clock detection and repetition count tests.
- It can be disabled to reduce power consumption.
- It has an AMBA AHB slave peripheral, accessible through 32-bit word single accesses only (else an AHB bus error is generated). Warning! any write not equal to 32 bits might corrupt the register content.

16.3 RNG functional description

16.3.1 RNG block diagram

Figure 102 shows the RNG block diagram.

Figure 102. RNG block diagram



MSv42096V1

16.3.2 RNG internal signals

Table 97 describes a list of useful-to-know internal signals available at the RNG level, not at the STM32 product level (on pads).

Table 97. RNG internal input/output signals

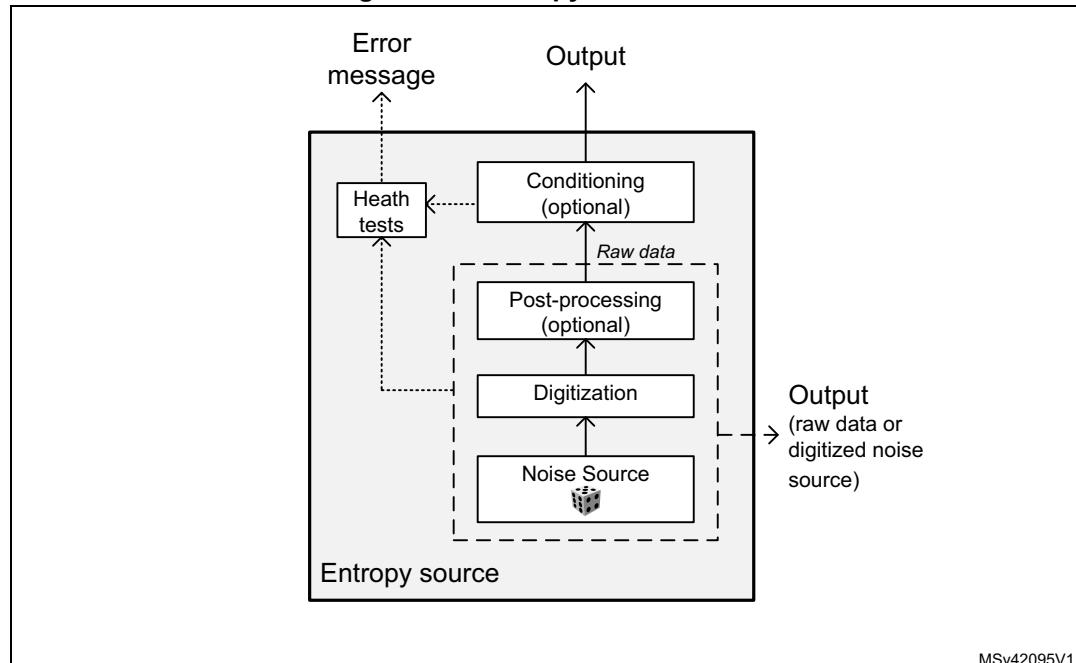
Signal name	Signal type	Description
rng_it	Digital output	RNG global interrupt request
rng_hclk	Digital input	AHB clock
rng_clk	Digital input	RNG dedicated clock, asynchronous to rng_hclk

16.3.3 Random number generation

The true random number generator (RNG) delivers truly random data through its AHB interface at deterministic intervals. The RNG implements the entropy source model pictured on [Figure 103](#), and provides three main functions to the application:

- Collects the bitstring output of the entropy source box
- Obtains samples of the noise source for validation purpose
- Collects error messages from continuous health tests

Figure 103. Entropy source model



MSv42095V1

The main components of the RNG are:

- A source of physical randomness (analog noise source)
- A digitization stage for this analog noise source
- A stage delivering post-processed noise source (raw data)
- An output buffer for the raw data. If further cryptographic conditioning is required by the application it will need to be performed by software.
- An optional output for the digitized noise source (unbuffered, on digital pads)
- Basic health tests on the digitized noise source

All those components are detailed below.

Noise source

The noise source is the component that contains the non-deterministic, entropy-providing activity that is ultimately responsible for the uncertainty associated with the bitstring output by the entropy source. It is composed of:

- Two analog noise sources, each based on three XORed free-running ring oscillator outputs. It is possible to disable those analog oscillators to save power, as described in

Section 16.4: RNG low-power usage.

- A sampling stage of these outputs clocked by a dedicated clock input (**rng_clk**), delivering a 2-bit raw data output.

This noise source sampling is independent to the AHB interface clock frequency (**rng_hclk**).

Note: In [Section 16.7: Entropy source validation](#) recommended RNG clock frequencies are given.

Post processing

The sample values obtained from a true random noise source consist of 2-bit bitstrings. Because this noise source output is biased, the RNG implements a post-processing component that reduces that bias to a tolerable level.

The RNG post-processing consists of two stages, applied to each noise source bits:

- The RNG takes half of the bits from the sampled noise source, and half of the bits from inverted sampled noise source. Thus, if the source generates more ‘1’ than ‘0’ (or the opposite), it is filtered
- A linear feedback shift register (LFSR) performs a whitening process, producing 8-bit strings.

This component is clocked by the RNG clock.

The times required between two random number generations, and between the RNG initialization and availability of first sample are described in [Section 16.6: RNG processing time](#).

Output buffer

The RNG_DR data output register can store up to two 16-bit words which have been output from the post-processing component (LFSR). In order to read back 32-bit random samples it is required to wait 42 RNG clock cycles.

Whenever a random number is available through the RNG_DR register the DRDY flag transitions from “0” to “1”. This flag remains high until output buffer becomes empty after reading one word from the RNG_DR register.

Note: When interrupts are enabled an interrupt is generated when this data ready flag transitions from “0” to “1”. Interrupt is then cleared automatically by the RNG as explained above.

Health checks

This component ensures that the entire entropy source (with its noise source) starts then operates as expected, obtaining assurance that failures are caught quickly and with a high probability and reliability.

The RNG implements the following health check features:

1. Behavior tests, applied to the entropy source *at run-time*
 - Repetition count test, flagging an error when:
 - a) One of the noise source has provided more than 64 consecutive bits at a constant value ("0" or "1")
 - b) One of the noise sources has delivered more than 32 consecutive occurrence of two bits patterns ("01" or "10")
 - 2. Vendor specific continuous test
 - Real-time "too slow" sampling clock detector, flagging an error when one RNG clock cycle is smaller than AHB clock cycle divided by 16.

The CECS and SECS status bits in the RNG_SR register indicate when an error condition is detected, as detailed in [Section 16.3.7: Error management](#).

Note: An interrupt can be generated when an error is detected.

16.3.4 RNG initialization

When a hardware reset occurs the following chain of events occurs:

1. The analog noise source is enabled, and logic starts sampling the analog output after four RNG clock cycles, filling LFSR shift register and associated 16-bit post-processing shift register.
2. The output buffer is refilled automatically according to the RNG usage.

The associated initialization time can be found in [Section 16.6: RNG processing time](#).

16.3.5 RNG operation

Normal operations

To run the RNG using interrupts the following steps are recommended:

1. Enable the interrupts by setting the IE bit in the RNG_CR register. At the same time enable the RNG by setting the bit RNGEN=1.
2. An interrupt is now generated when a random number is ready or when an error occurs. Therefore at each interrupt, check that:
 - No error occurred. The SEIS and CEIS bits should be set to '0' in the RNG_SR register.
 - A random number is ready. The DRDY bit must be set to '1' in the RNG_SR register.
 - If above two conditions are true the content of the RNG_DR register can be read.

To run the RNG in polling mode following steps are recommended:

1. Enable the random number generation by setting the RNGEN bit to “1” in the RNG_CR register.
2. Read the RNG_SR register and check that:
 - No error occurred (the SEIS and CEIS bits should be set to ‘0’)
 - A random number is ready (the DRDY bit should be set to ‘1’)
3. If above conditions are true read the content of the RNG_DR register.

Note: When data is not ready (DRDY=“0”) RNG_DR returns zero.

Low-power operations

If the power consumption is a concern to the application, low-power strategies can be used, as described in [Section 16.4: RNG low-power usage on page 451](#).

Software post-processing

If a NIST approved DRBG with 128 bits of security strength is required an approved random generator software must be built around the RNG true random number generator.

16.3.6 RNG clocking

The RNG runs on two different clocks: the AHB bus clock and a dedicated RNG clock.

The AHB clock is used to clock the AHB banked registers and the post-processing component. The RNG clock is used for noise source sampling. Recommended clock configurations are detailed in [Section 16.7: Entropy source validation](#).

Caution: When the CED bit in the RNG_CR register is set to “0”, the RNG clock frequency **must be higher** than AHB clock frequency divided by 16, otherwise the clock checker will flag a clock error (CECS or CEIS in the RNG_SR register) and the RNG will stop producing random numbers.

See [Section 16.3.1: RNG block diagram](#) for details (AHB and RNG clock domains).

16.3.7 Error management

In parallel to random number generation an health check block verifies the correct noise source behavior and the frequency of the RNG source clock as detailed in this section. Associated error state is also described.

Clock error detection

When the clock error detection is enabled (CED = 0) and if the RNG clock frequency is too low, the RNG stops generating random numbers and sets to “1” both the **CEIS** and **CECS** bits to indicate that a clock error occurred. In this case, the application should check that the RNG clock is configured correctly (see [Section 16.3.6: RNG clocking](#)) and then it must clear the CEIS bit interrupt flag. As soon as the RNG clock operates correctly, the CECS bit will be automatically cleared.

The RNG operates only when the CECS flag is set to “0”. However note that the clock error has no impact on the previously generated random numbers, and the RNG_DR register contents can still be used.

Noise source error detection

When a noise source (or seed) error occurs, the RNG stops generating random numbers and sets to “1” both **SEIS** and **SECS** bits to indicate that a seed error occurred. If a value is available in the RNG_DR register, it must not be used as it may not have enough entropy.

In order to fully recover from a seed error application must clear the SEIS bit by writing it to “0”, then clear and set the RNGEN bit to reinitialize and restart the RNG.

16.4 RNG low-power usage

If power consumption is a concern, the RNG can be disabled as soon as the DRDY bit is set to “1” by setting the RNGEN bit to “0” in the RNG_CR register. The 32-bit random value stored in the RNG_DR register will be still be available. If a new random is needed the application will need to re-enable the RNG and wait for 42+4 RNG clock cycles.

When disabling the RNG the user deactivates all the analog seed generators, whose power consumption is given in the datasheet electrical characteristics section.

16.5 RNG interrupts

In the RNG an interrupt can be produced on the following events:

- Data ready flag
- Seed error, see [Section 16.3.7: Error management](#)
- Clock error, see [Section 16.3.7: Error management](#)

Dedicated interrupt enable control bits are available as shown in [Table 98](#)

Table 98. RNG interrupt requests

Interrupt event	Event flag	Enable control bit
Data ready flag	DRDY	IE
Seed error flag	SEIS	IE
Clock error flag	CEIS	IE

The user can enable or disable the above interrupt sources individually by changing the mask bits or the general interrupt control bit IE in the RNG_CR register. The status of the individual interrupt sources can be read from the RNG_SR register.

Note:

Interrupts are generated only when RNG is enabled.

16.6 RNG processing time

The RNG can produce one 32-bit random numbers every 42 RNG clock cycles.

After enabling or re-enabling the RNG using the RNGEN bit it takes 46 RNG clock cycles before random data are available.

16.7 Entropy source validation

16.7.1 Introduction

In order to assess the amount of entropy available from the RNG, STMicroelectronics has tested the peripheral against AIS-31 PTG.2 set of tests. The results can be provided on demand or the customer can reproduce the measurements using the AIS reference software. The customer could also test the RNG against an older NIST SP800-22 set of tests.

16.7.2 Validation conditions

STMicroelectronics has validated the RNG true random number generator in the following conditions:

- RNG clock rng_clk= 48 MHz (CED bit = '0' in RNG_CR register) and rng_clk= 400kHz (CED bit="1" in RNG_CR)
- AHB clock rng_hclk= 60 MHz

16.7.3 Data collection

If raw data needs to be read instead of pre-processed data the developer is invited to contact STMicroelectronics to receive the correct procedure to follow.

16.8 RNG registers

The RNG is associated with a control register, a data register and a status register.

16.8.1 RNG control register (RNG_CR)

Address offset: 0x0000

Reset value: 0x0000 0000

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

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

Bit 5 **CED**: Clock error detection

- 0: Clock error detection is enable
- 1: Clock error detection is disable

The clock error detection cannot be enabled nor disabled on-the-fly when the RNG is enabled, i.e. to enable or disable CED the RNG must be disabled.

Bit 4 Reserved, must be kept at reset value.

Bit 3 **IE**: Interrupt Enable

- 0: RNG Interrupt is disabled
- 1: RNG Interrupt is enabled. An interrupt is pending as soon as DRDY='1', SEIS='1' or CEIS='1' in the RNG_SR register.

Bit 2 **RNGEN**: True random number generator enable

- 0: True random number generator is disabled. Analog noise sources are powered off and logic clocked by the RNG clock is gated.
- 1: True random number generator is enabled.

Bits 1:0 Reserved, must be kept at reset value.

16.8.2 RNG status register (RNG_SR)

Address offset: 0x004

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SEIS	CEIS	Res.	Res.	SECS	CECS	DRDY								
								rc_w0	rc_w0			r	r	r	

Bits 31:7 Reserved, must be kept at reset value.

Bit 6 **SEIS:** Seed error interrupt status

This bit is set at the same time as SECS. It is cleared by writing it to '0'.

0: No faulty sequence detected

1: At least one faulty sequence has been detected. See **SECS** bit description for details.

An interrupt is pending if IE = '1' in the RNG_CR register.

Bit 5 **CEIS:** Clock error interrupt status

This bit is set at the same time as CECS. It is cleared by writing it to '0'.

0: The RNG clock is correct ($f_{RNGCLK} > f_{HCLK}/16$)

1: The RNG has been detected too slow ($f_{RNGCLK} < f_{HCLK}/16$)

An interrupt is pending if IE = '1' in the RNG_CR register.

Bits 4:3 Reserved, must be kept at reset value.

Bit 2 **SECS:** Seed error current status

0: No faulty sequence has currently been detected. If the SEIS bit is set, this means that a faulty sequence was detected and the situation has been recovered.

1: One of the noise source has provided more than 64 consecutive bits at a constant value ("0" or "1"), or more than 32 consecutive occurrence of two bits patterns ("01" or "10")

Bit 1 **CECS:** Clock error current status

0: The RNG clock is correct ($f_{RNGCLK} > f_{HCLK}/16$). If the CEIS bit is set, this means that a slow clock was detected and the situation has been recovered.

1: The RNG clock is too slow ($f_{RNGCLK} < f_{HCLK}/16$).

Note: CECS bit is valid only if the CED bit in the RNG_CR register is set to "0".

Bit 0 **DRDY:** Data Ready

0: The RNG_DR register is not yet valid, no random data is available.

1: The RNG_DR register contains valid random data.

Once the RNG_DR register has been read, this bit returns to '0' until a new random value is generated.

If IE='1' in the RNG_CR register, an interrupt is generated when DRDY='1'.

16.8.3 RNG data register (RNG_DR)

Address offset: 0x008

Reset value: 0x0000 0000

The RNG_DR register is a read-only register that delivers a 32-bit random value when read. After being read this register delivers a new random value after 42 periods of RNG clock if the output FIFO is empty.

The content of this register is valid when DRDY='1', even if RNGEN='0'.

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

Bits 31:0 **RNDATA[31:0]: Random data**

32-bit random data which are valid when DRDY='1'. When DRDY='0' RNDATA value is zero.

16.8.4 RNG register map

Table 99 gives the RNG register map and reset values.

Table 99. RNG register map and reset map

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
0x000	RNG_CR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SEIS	CEIS	0	CED	5						
	Reset value																									0	0	Res.	Res.	4						
0x004	RNG_SR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	0	0	IE	3						
	Reset value																									0	0	0	0	0	0	0				
0x008	RNG_DR	RNDATA[31:0]																																		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

17 AES hardware accelerator (AES)

17.1 Introduction

The AES hardware accelerator (AES) encrypts or decrypts data, using an algorithm and implementation fully compliant with the advanced encryption standard (AES) defined in Federal information processing standards (FIPS) publication 197.

Multiple chaining modes are supported (ECB, CBC, CTR, GCM, GMAC, CCM), for key sizes of 128 or 256 bits.

The AES accelerator is a 32-bit AHB peripheral. It supports DMA single transfers for incoming and outgoing data (two DMA channels required).

The AES peripheral provides hardware acceleration to AES cryptographic algorithms packaged in STM32 cryptographic library.

AES is an AMBA AHB slave peripheral, accessible through 32-bit word single accesses only (otherwise an AHB bus error is generated and write accesses are ignored).

17.2 AES main features

- Compliance with NIST “Advanced encryption standard (AES), FIPS publication 197” from November 2001
- 128-bit data block processing
- Support for cipher key lengths of 128-bit and 256-bit
- Encryption and decryption with multiple chaining modes:
 - Electronic codebook (ECB) mode
 - Cipher block chaining (CBC) mode
 - Counter (CTR) mode
 - Galois counter mode (GCM)
 - Galois message authentication code (GMAC) mode
 - Counter with CBC-MAC (CCM) mode
- 51 or 75 clock cycle latency in ECB mode for processing one 128-bit block of data with, respectively, 128-bit or 256-bit key
- Integrated key scheduler with its key derivation stage (ECB or CBC decryption only)
- AMBA AHB slave peripheral, accessible through 32-bit word single accesses only
- 256-bit register for storing the cryptographic key (eight 32-bit registers)
- 128-bit register for storing initialization vector (four 32-bit registers)
- 32-bit buffer for data input and output
- Automatic data flow control with support of single-transfer direct memory access (DMA) using two channels (one for incoming data, one for processed data)
- Data-swapping logic to support 1-, 8-, 16- or 32-bit data
- Possibility for software to suspend a message if AES needs to process another message with a higher priority, then resume the original message

17.3 AES implementation

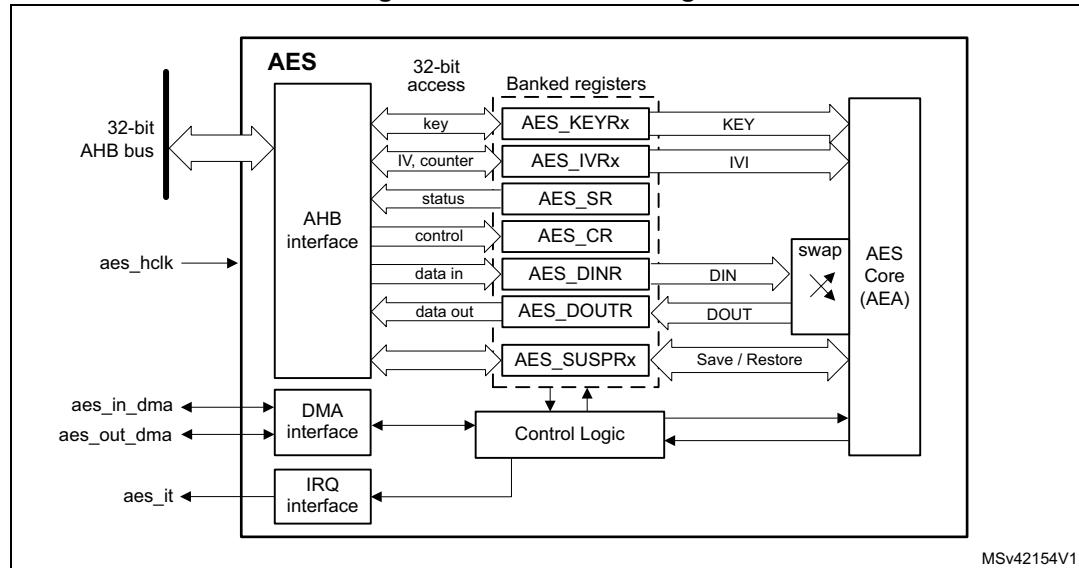
The device has a single instance of AES peripheral.

17.4 AES functional description

17.4.1 AES block diagram

Figure 104 shows the block diagram of AES.

Figure 104. AES block diagram



17.4.2 AES internal signals

Table 100 describes the user relevant internal signals interfacing the AES peripheral.

Table 100. AES internal input/output signals

Signal name	Signal type	Description
aes_hclk	digital input	AHB bus clock
aes_it	digital output	AES interrupt request
aes_in_dma	digital input/output	Input DMA single request/acknowledge
aes_out_dma	digital input/output	Output DMA single request/acknowledge

17.4.3 AES cryptographic core

Overview

The AES cryptographic core consists of the following components:

- AES algorithm (AEA)
- multiplier over a binary Galois field (GF2mul)
- key input
- initialization vector (IV) input
- chaining algorithm logic (XOR, feedback/counter, mask)

The AES core works on 128-bit data blocks (four words) with 128-bit or 256-bit key length. Depending on the chaining mode, the AES requires zero or one 96-bit initialization vector IV (and a 32-bit counter field).

The AES features the following modes of operation:

- **Mode 1:**
Plaintext encryption using a key stored in the AES_KEYRx registers
- **Mode 2:**
ECB or CBC decryption key preparation. It must be used prior to selecting Mode 3 with ECB or CBC chaining modes. The key prepared for decryption is stored automatically in the AES_KEYRx registers. Now the AES peripheral is ready to switch to Mode 3 for executing data decryption.
- **Mode 3:**
Ciphertext decryption using a key stored in the AES_KEYRx registers. When ECB and CBC chaining modes are selected, the key must be prepared beforehand, through Mode 2.
- **Mode 4:**
ECB or CBC ciphertext single decryption using the key stored in the AES_KEYRx registers (the initial key is derived automatically).

Note: Mode 2 and mode 4 are only used when performing ECB and CBC decryption.

When Mode 4 is selected only one decryption can be done, therefore usage of Mode 2 and Mode 3 is recommended instead.

The operating mode is selected by programming the MODE[1:0] bitfield of the AES_CR register. It may be done only when the AES peripheral is disabled.

Typical data processing

Typical usage of the AES is described in [Section 17.4.4: AES procedure to perform a cipher operation on page 464](#).

Note: The outputs of the intermediate AEA stages are never revealed outside the cryptographic boundary, with the exclusion of the IVI bitfield.

Chaining modes

The following chaining modes are supported by AES, selected through the CHMOD[2:0] bitfield of the AES_CR register:

- Electronic code book (ECB)
- Cipher block chaining (CBC)
- Counter (CTR)
- Galois counter mode (GCM)
- Galois message authentication code (GMAC)
- Counter with CBC-MAC (CCM)

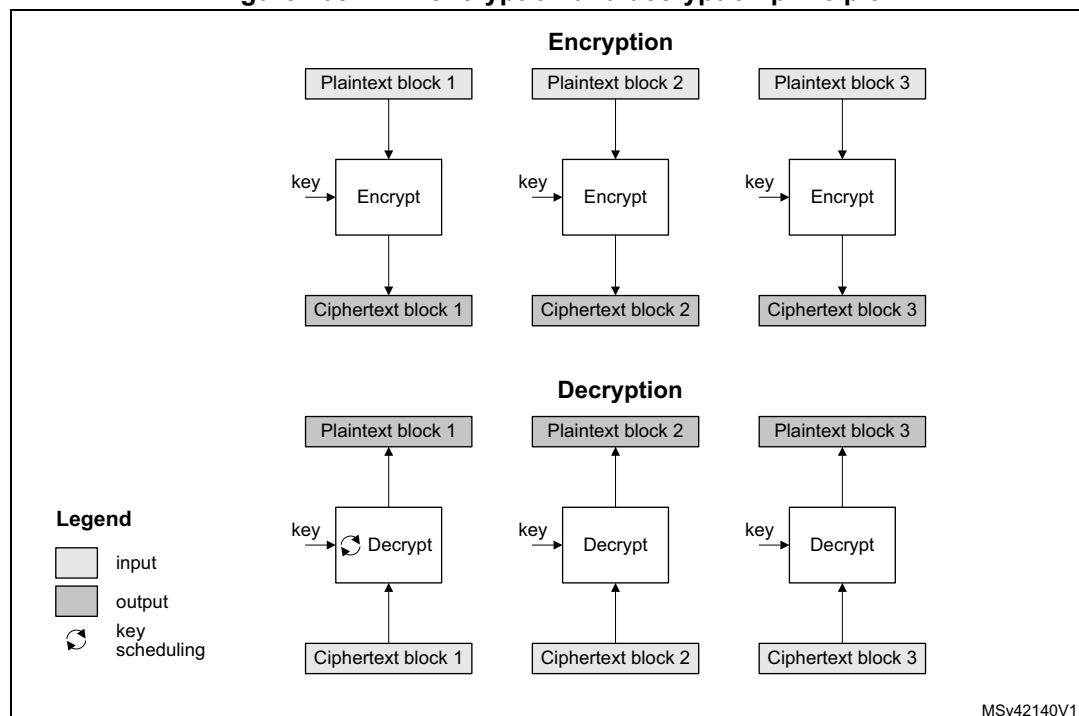
Note: The chaining mode may be changed only when AES is disabled (bit EN of the AES_CR register set).

Principle of each AES chaining mode is provided in the following subsections.

Detailed information is in dedicated sections, starting from [Section 17.4.8: AES basic chaining modes \(ECB, CBC\)](#).

Electronic codebook (ECB) mode

Figure 105. ECB encryption and decryption principle

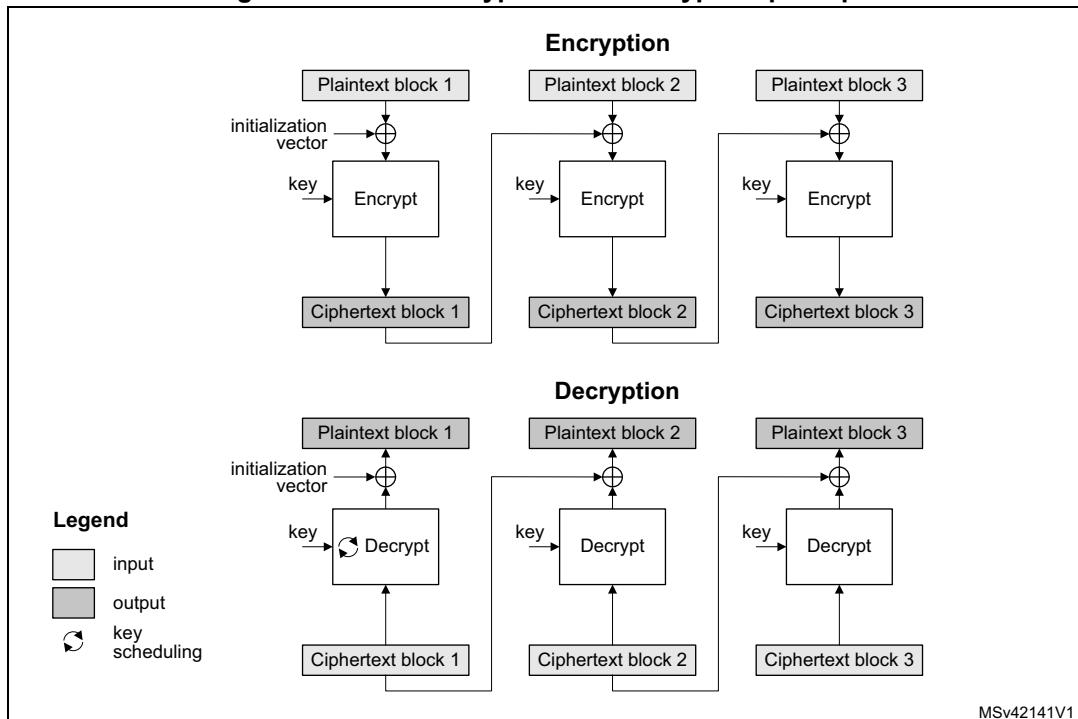


ECB is the simplest mode of operation. There are no chaining operations, and no special initialization stage. The message is divided into blocks and each block is encrypted or decrypted separately.

Note: For decryption, a special key scheduling is required before processing the first block.

Cipher block chaining (CBC) mode

Figure 106. CBC encryption and decryption principle

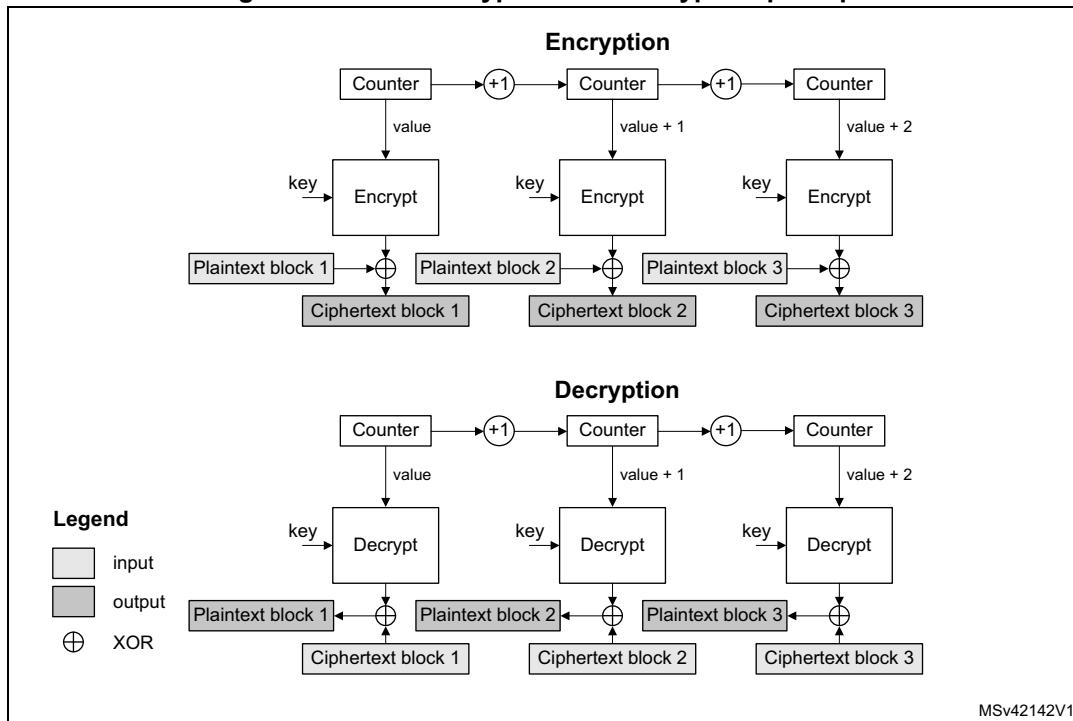


In CBC mode the output of each block chains with the input of the following block. To make each message unique, an initialization vector is used during the first block processing.

Note: For decryption, a special key scheduling is required before processing the first block.

Counter (CTR) mode

Figure 107. CTR encryption and decryption principle

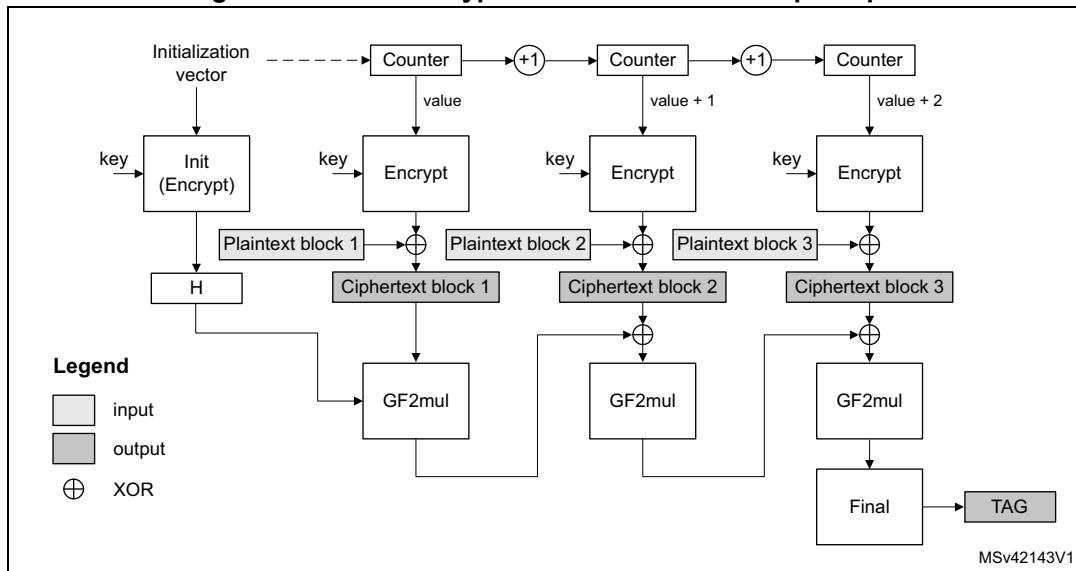


The CTR mode uses the AES core to generate a key stream. The keys are then XORed with the plaintext to obtain the ciphertext as specified in NIST Special Publication 800-38A, *Recommendation for Block Cipher Modes of Operation*.

Note: Unlike with ECB and CBC modes, no key scheduling is required for the CTR decryption, since in this chaining scheme the AES core is always used in encryption mode for producing the key stream, or counter blocks.

Galois/counter mode (GCM)

Figure 108. GCM encryption and authentication principle

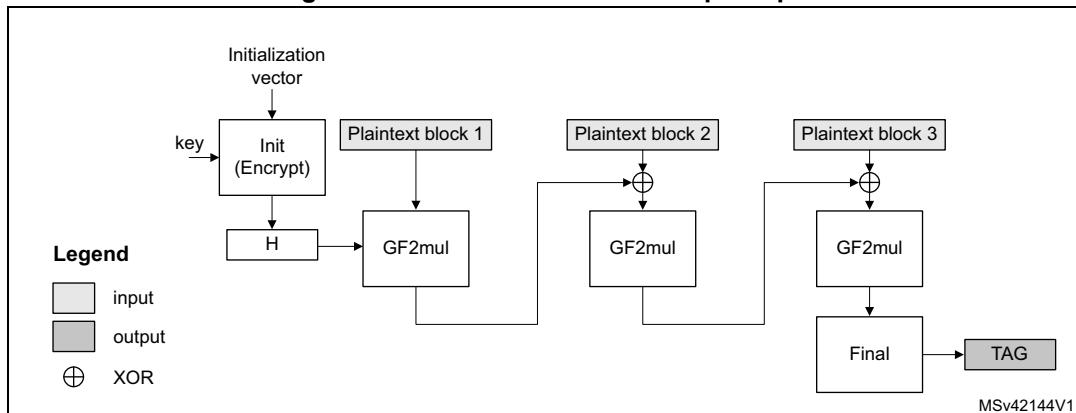


In Galois/counter mode (GCM), the plaintext message is encrypted while a message authentication code (MAC) is computed in parallel, thus generating the corresponding ciphertext and its MAC (also known as authentication tag). It is defined in *NIST Special Publication 800-38D, Recommendation for Block Cipher Modes of Operation - Galois/Counter Mode (GCM) and GMAC*.

GCM mode is based on AES in counter mode for confidentiality. It uses a multiplier over a fixed finite field for computing the message authentication code. It requires an initial value and a particular 128-bit block at the end of the message.

Galois message authentication code (GMAC) principle

Figure 109. GMAC authentication principle

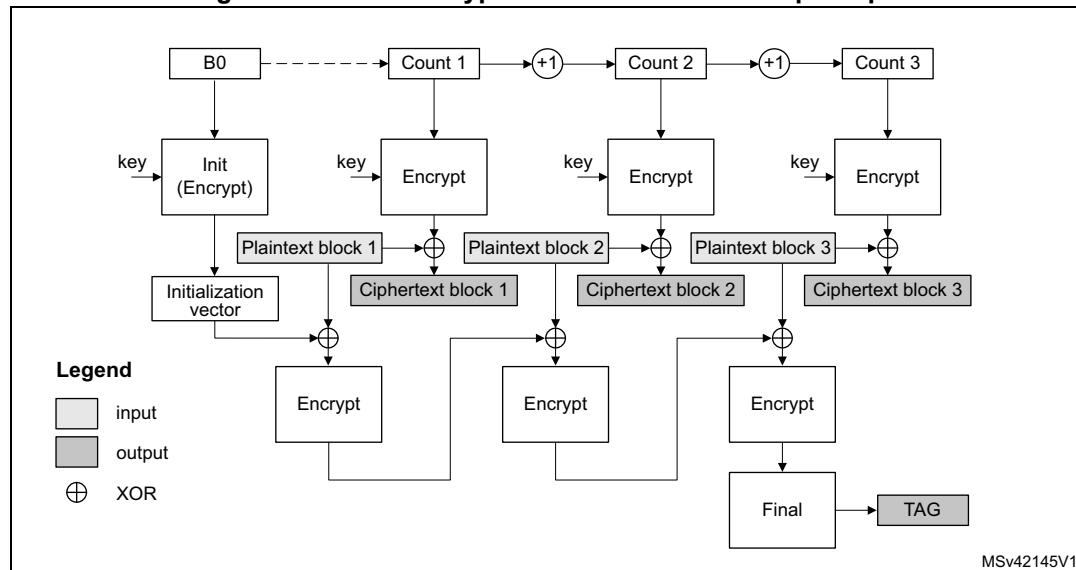


Galois message authentication code (GMAC) allows authenticating a message and generating the corresponding message authentication code (MAC). It is defined in *NIST Special Publication 800-38D, Recommendation for Block Cipher Modes of Operation - Galois/Counter Mode (GCM) and GMAC*.

GMAC is similar to GCM, except that it is applied on a message composed only by plaintext authenticated data (that is, only header, no payload).

Counter with CBC-MAC (CCM) principle

Figure 110. CCM encryption and authentication principle



In Counter with cipher block chaining-message authentication code (CCM) mode, the plaintext message is encrypted while a message authentication code (MAC) is computed in parallel, thus generating the corresponding ciphertext and the corresponding MAC (also known as tag). It is described by NIST in *Special Publication 800-38C, Recommendation for Block Cipher Modes of Operation - The CCM Mode for Authentication and Confidentiality*.

CCM mode is based on AES in counter mode for confidentiality and it uses CBC for computing the message authentication code. It requires an initial value.

Like GCM, the CCM chaining mode can be applied on a message composed only by plaintext authenticated data (that is, only header, no payload). Note that this way of using CCM is not called CMAC (it is not similar to GCM/GMAC), and its usage is not recommended by NIST.

17.4.4 AES procedure to perform a cipher operation

Introduction

A typical cipher operation is explained below. Detailed information is provided in sections starting from [Section 17.4.8: AES basic chaining modes \(ECB, CBC\)](#).

The flowcharts shown in [Figure 111](#) and [Figure 112](#) describe the way STM32 cryptographic library implements the AES algorithm. AES accelerates the execution of the AES-128 and AES-256 cryptographic algorithms in ECB, CBC, CTR, CCM, and GCM operating modes.

Note: *For more details on the cryptographic library, refer to the UM1924 user manual “STM32 crypto library” available from www.st.com.*

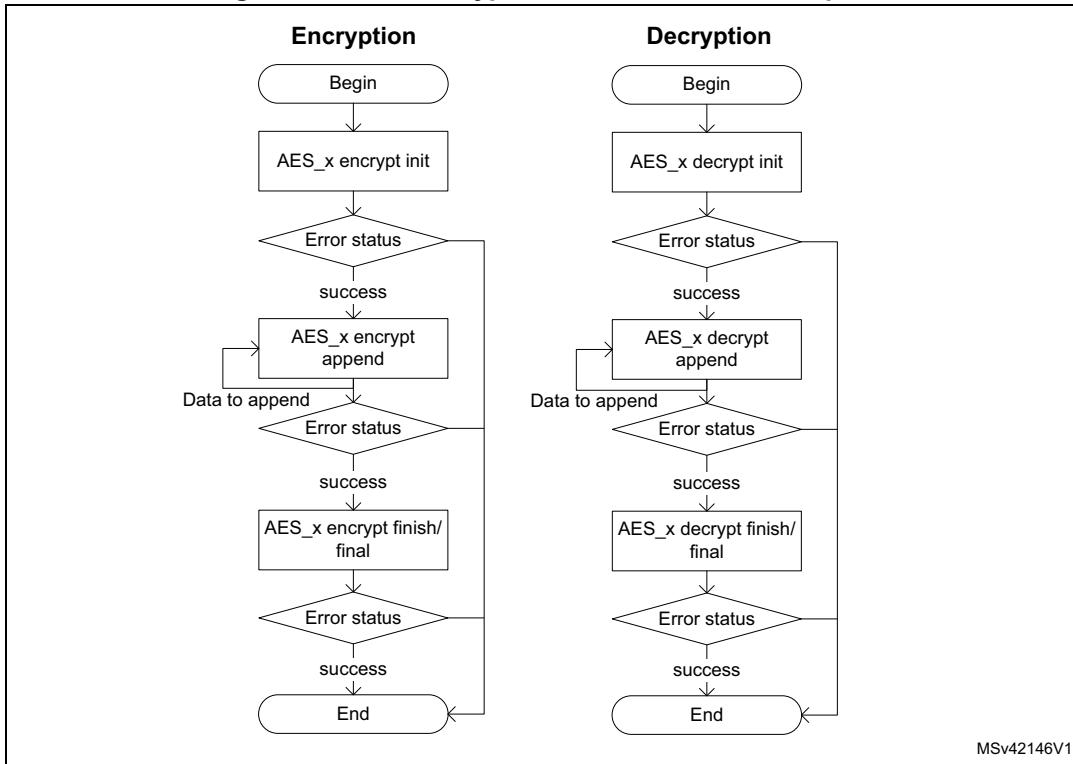
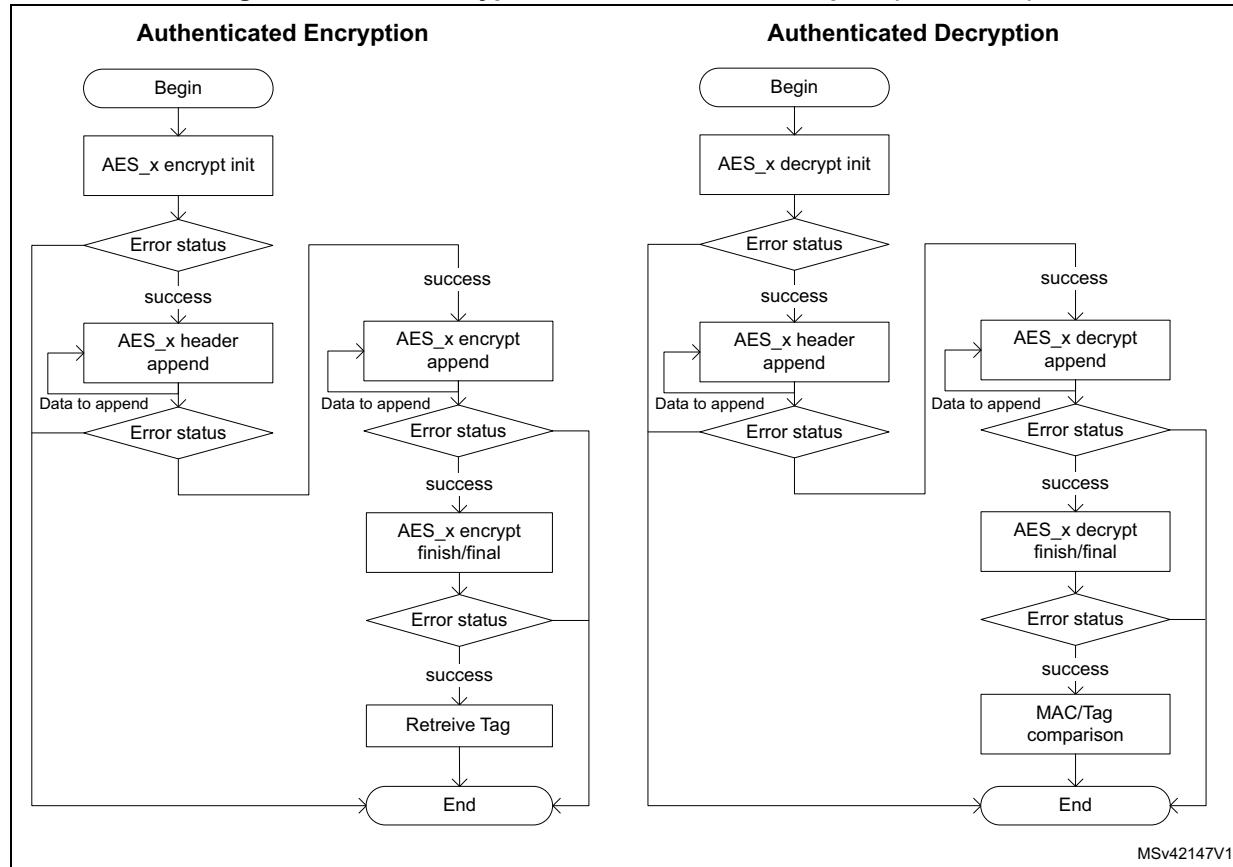
Figure 111. STM32 cryptolib AES flowchart examples

Figure 112. STM32 cryptolib AES flowchart examples (continued)



Initialization of AES

To initialize AES, first disable it by clearing the EN bit of the AES_CR register. Then perform the following steps in any order:

- Configure the AES mode, by programming the MODE[1:0] bitfield of the AES_CR register.
 - For encryption, Mode 1 must be selected (MODE[1:0] = 00).
 - For decryption, Mode 3 must be selected (MODE[1:0] = 10), unless ECB or CBC chaining modes are used. In this latter case, an initial key derivation of the encryption key must be performed, as described in [Section 17.4.5: AES decryption key preparation](#).
- Select the chaining mode, by programming the CHMOD[2:0] bitfield of the AES_CR register
- Configure the key size (128-bit or 256-bit), with the KEYSIZE bitfield of the AES_CR register.
- Write a symmetric key into the AES_KEYRx registers (4 or 8 registers depending on the key size).
- Configure the data type (1-, 8-, 16- or 32-bit), with the DATATYPE[1:0] bitfield in the AES_CR register.
- When it is required (for example in CBC or CTR chaining modes), write the initialization vectors into the AES_IVRx register.

Data append

This section describes different ways of appending data for processing, where the size of data to process is not a multiple of 128 bits.

For ECB, CBC and GCM encryption mode, refer to [Section 17.4.6: AES ciphertext stealing and data padding](#). The second-last and the last block management in these cases is more complex than in the sequence described in this section.

Data append through polling

This method uses flag polling to control the data append.

For all other cases, the data is appended through the following sequence:

1. Enable the AES peripheral by setting the EN bit of the AES_CR register.
2. Repeat the following sub-sequence until the payload is entirely processed:
 - a) Write four input data words into the AES_DINR register.
 - b) Wait until the status flag CCF is set in the AES_SR, then read the four data words from the AES_DOUTR register.
 - c) Clear the CCF flag, by setting the CCFC bit of the AES_CR register.
 - d) If the data block just processed is the second-last block of the message and the significant data in the last block to process is inferior to 128 bits, pad the remainder of the last block with zeros
3. Discard the data that is not part of the payload, then disable the AES peripheral by clearing the EN bit of the AES_CR register.

Note: *Up to three wait cycles are automatically inserted between two consecutive writes to the AES_DINR register, to allow sending the key to the AES processor.*

Data append using interrupt

The method uses interrupt from the AES peripheral to control the data append, through the following sequence:

1. Enable interrupts from AES by setting the CCFIE bit of the AES_CR register.
2. Enable the AES peripheral by setting the EN bit of the AES_CR register.
3. Write first four input data words into the AES_DINR register.
4. Handle the data in the AES interrupt service routine, upon interrupt:
 - a) Read four output data words from the AES_DOUTR register.
 - b) Clear the CCF flag and thus the pending interrupt, by setting the CCFC bit of the AES_CR register
 - c) If the data block just processed is the second-last block of an message and the significant data in the last block to process is inferior to 128 bits, pad the remainder of the last block with zeros. Then proceed with point 4e).
 - d) If the data block just processed is the last block of the message, discard the data that is not part of the payload, then disable the AES peripheral by clearing the EN bit of the AES_CR register and quit the interrupt service routine.
 - e) Write next four input data words into the AES_DINR register and quit the interrupt service routine.

Note: *AES is tolerant of delays between consecutive read or write operations, which allows, for example, an interrupt from another peripheral to be served between two AES computations.*

Data append using DMA

With this method, all the transfers and processing are managed by DMA and AES. To use the method, proceed as follows:

1. Prepare the last four-word data block (if the data to process does not fill it completely), by padding the remainder of the block with zeros.
2. Configure the DMA controller so as to transfer the data to process from the memory to the AES peripheral input and the processed data from the AES peripheral output to the memory, as described in [Section 17.4.16: AES DMA interface](#). Configure the DMA controller so as to generate an interrupt on transfer completion.
3. Enable the AES peripheral by setting the EN bit of the AES_CR register
4. Enable DMA requests by setting the DMAINEN and DMAOUTEN bits of the AES_CR register.
5. Upon DMA interrupt indicating the transfer completion, get the AES-processed data from the memory.

Note: *The CCF flag has no use with this method, because the reading of the AES_DOUTR register is managed by DMA automatically, without any software action, at the end of the computation phase.*

17.4.5 AES decryption key preparation

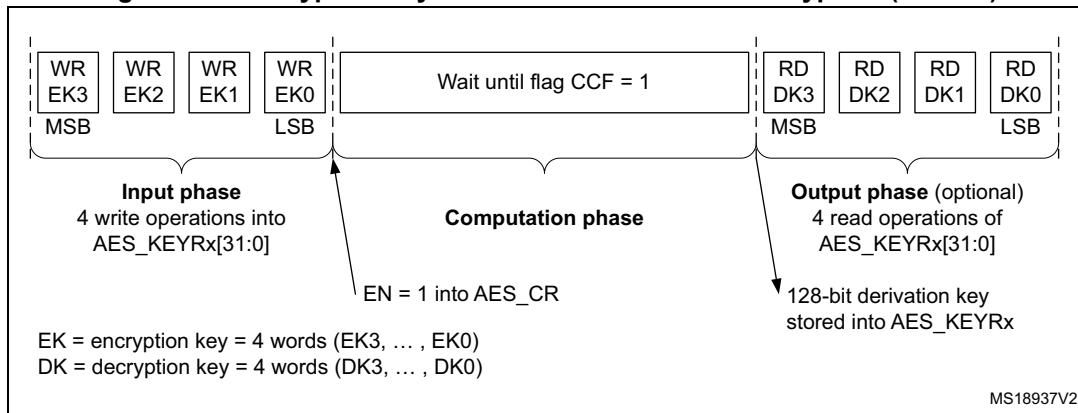
For an ECB or CBC decryption, a key for the first round of decryption must be derived from the key of the last round of encryption. This is why a complete key schedule of encryption is required before performing the decryption. This key preparation is **not required** for AES decryption in modes other than ECB or CBC.

Recommended method is to select the Mode 2 by setting to 01 the MODE[1:0] bitfield of the AES_CR (key process only), then proceed with the decryption by setting MODE[1:0] to 10 (Mode 3, decryption only). Mode 2 usage is described below:

1. Disable the AES peripheral by clearing the EN bit of the AES_CR register.
2. Select Mode 2 by setting to 01 the MODE[1:0] bitfield of the AES_CR. The CHMOD[2:0] bitfield is not significant in this case because this key derivation mode is independent of the chaining algorithm selected.
3. Set key length to 128 or 256 bits, via KEYSIZE bit of AES_CR register.
4. Write the AES_KEYRx registers (128 or 256 bits) with encryption key, as shown in [Figure 113](#). Writes to the AES_IVRx registers have no effect.
5. Enable the AES peripheral, by setting the EN bit of the AES_CR register.
6. Wait until the CCF flag is set in the AES_SR register.
7. Derived key is available in AES core, ready to use for decryption. Application can also read the AES_KEYRx register to obtain the derived key if needed, as shown in [Figure 113](#) (the processed key is loaded automatically into the AES_KEYRx registers).

Note: *The AES is disabled by hardware when the derivation key is available.*

To restart a derivation key computation, repeat steps 4, 5, 6 and 7.

Figure 113. Encryption key derivation for ECB/CBC decryption (Mode 2)

If the software stores the initial key prepared for decryption, it is enough to do the key schedule operation only once for all the data to be decrypted with a given cipher key.

Note: *The operation of the key preparation lasts 80 or 109 clock cycles, depending on the key size (128- or 256-bit).*

Note: *Alternative key preparation is to select Mode 4 by setting to 11 the MODE[1:0] bitfield of the `AES_CR` register. In this case Mode 3 cannot be used.*

17.4.6 AES ciphertext stealing and data padding

When using AES in ECB or CBC modes to manage messages the size of which is not a multiple of the block size (128 bits), ciphertext stealing techniques are used, such as those described in NIST Special Publication 800-38A, *Recommendation for Block Cipher Modes of Operation: Three Variants of Ciphertext Stealing for CBC Mode*. Since the AES peripheral on the device does not support such techniques, **the last two blocks** of input data must be handled in a special way by the application.

Note: *Ciphertext stealing techniques are not documented in this reference manual.*

Similarly, when AES is used in other modes than ECB or CBC, an incomplete input data block (that is, block with input data shorter than 128 bits) must be padded with zeros prior to encryption (that is, extra bits must be appended to the trailing end of the data string). After decryption, the extra bits must be discarded. As AES does not implement automatic data padding operation to **the last block**, the application must follow the recommendation given in [Section 17.4.4: AES procedure to perform a cipher operation on page 464](#) to manage messages the size of which is not a multiple of 128 bits.

Note: *Padding data are swapped in a similar way as normal data, according to the DATATYPE[1:0] field of the `AES_CR` register (see [Section 17.4.13: .AES data registers and data swapping on page 490](#) for details).*

A workaround is required in order to properly compute authentication tags for **GCM encryption**, when the input data in the last block is **inferior to 128 bits**. During GCM encryption payload phase and before inserting a last plaintext block smaller than 128 bits, then application must apply the following steps:

1. Disable the AES peripheral by clearing the EN bit of the AES_CR register
2. Change the mode to CTR by writing 010 to the CHMOD[2:0] bitfield of the AES_CR register.
3. Pad the last block (smaller than 128 bits) with zeros to have a complete block of 128 bits, then write it into AES_DINR register.
4. Upon encryption completion, read the 128-bit ciphertext from the AES_DOUTR register and store it as intermediate data.
5. Change again the mode to GCM by writing 011 to the CHMOD[2:0] bitfield of the AES_CR register.
6. Select Final phase by writing 11 to the GCMPH[1:0] bitfield of the AES_CR register.
7. In the intermediate data, set to zero the bits corresponding to the padded bits of the last block of payload, then insert the resulting data into AES_DINR register.
8. Wait for operation completion, and read data on AES_DOUTR. This data is to be discarded.
9. Apply the normal Final phase as described in [Section 17.4.10: AES Galois/counter mode \(GCM\) on page 478](#)

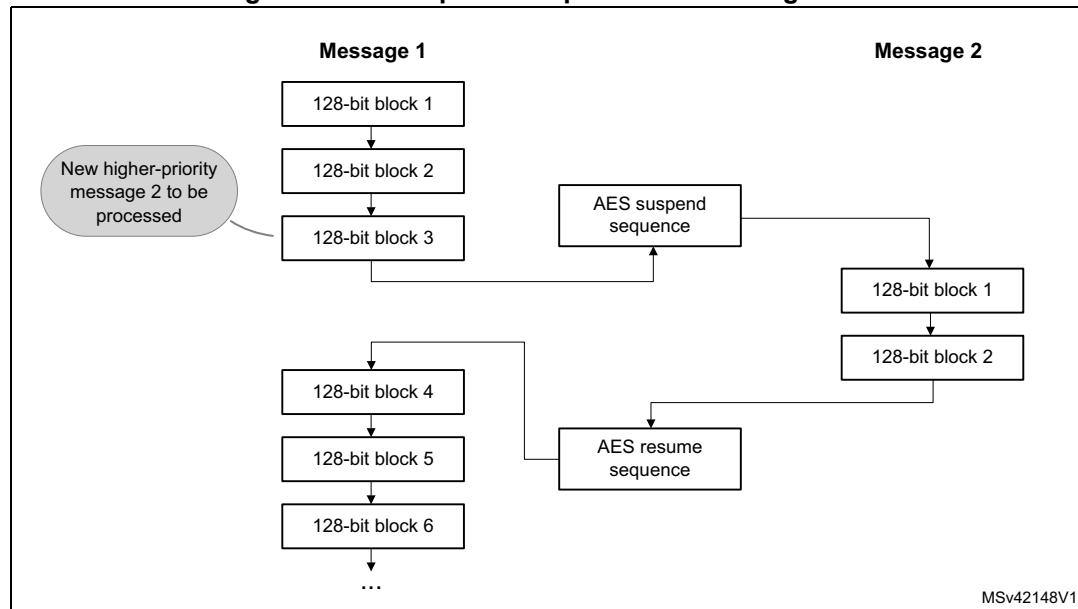
17.4.7 AES task suspend and resume

A message can be suspended if another message with a higher priority must be processed. When this highest priority message is sent, the suspended message can resume in both encryption or decryption mode.

Suspend/resume operations do not break the chaining operation and the message processing can resume as soon as AES is enabled again to receive the next data block.

[Figure 114](#) gives an example of suspend/resume operation: Message 1 is suspended in order to send a shorter and higher-priority Message 2.

Figure 114. Example of suspend mode management



A detailed description of suspend/resume operations is in the sections dedicated to each AES mode.

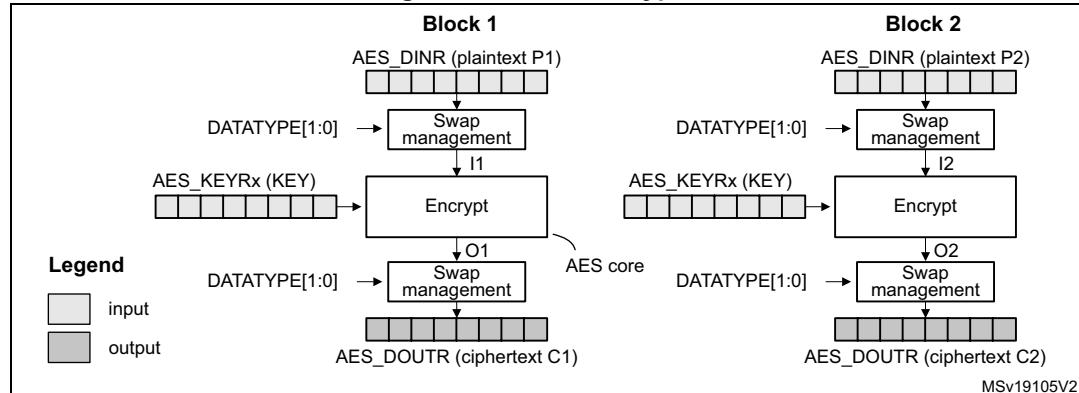
17.4.8 AES basic chaining modes (ECB, CBC)

Overview

This section gives a brief explanation of the four basic operation modes provided by the AES computing core: ECB encryption, ECB decryption, CBC encryption and CBC decryption. For detailed information, refer to the FIPS publication 197 from November 26, 2001.

Figure 115 illustrates the electronic codebook (ECB) encryption.

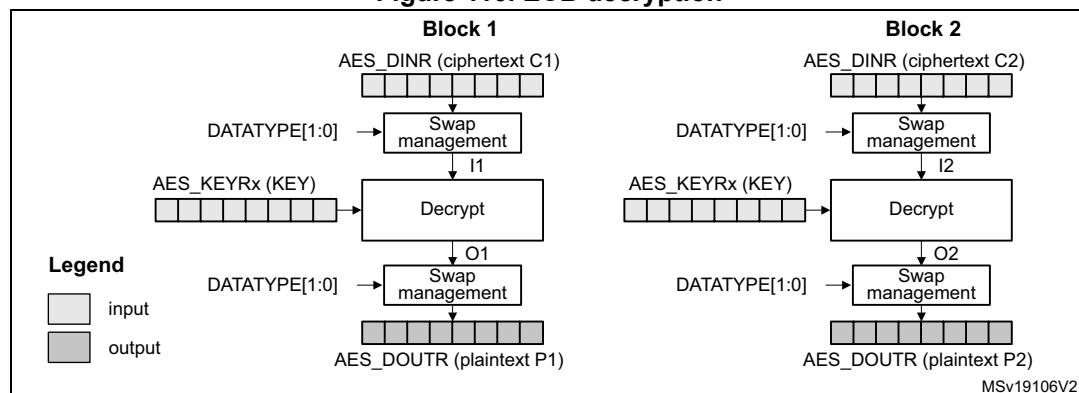
Figure 115. ECB encryption



In ECB encrypt mode, the 128-bit plaintext input data block Px in the AES_DINR register first goes through bit/byte/half-word swapping. The swap result Ix is processed with the AES core set in encrypt mode, using a 128- or 256-bit key. The encryption result Ox goes through bit/byte/half-word swapping, then is stored in the AES_DOUTR register as 128-bit ciphertext output data block Cx. The ECB encryption continues in this way until the last complete plaintext block is encrypted.

Figure 116 illustrates the electronic codebook (ECB) decryption.

Figure 116. ECB decryption

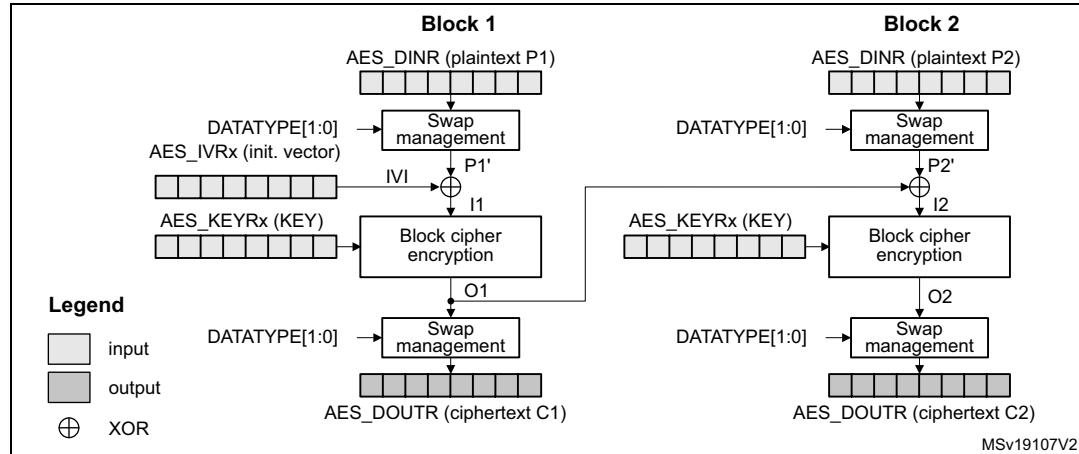


To perform an AES decryption in the ECB mode, the secret key has to be prepared by collecting the last-round encryption key (which requires to first execute the complete key schedule for encryption), and using it as the first-round key for the decryption of the ciphertext. This preparation is supported by the AES core.

In ECB decrypt mode, the 128-bit ciphertext input data block C1 in the AES_DINR register first goes through bit/byte/half-word swapping. The keying sequence is reversed compared to that of the ECB encryption. The swap result I1 is processed with the AES core set in decrypt mode, using the formerly prepared decryption key. The decryption result goes through bit/byte/half-word swapping, then is stored in the AES_DOUTR register as 128-bit plaintext output data block P1. The ECB decryption continues in this way until the last complete ciphertext block is decrypted.

Figure 117 illustrates the cipher block chaining (CBC) encryption mode.

Figure 117. CBC encryption

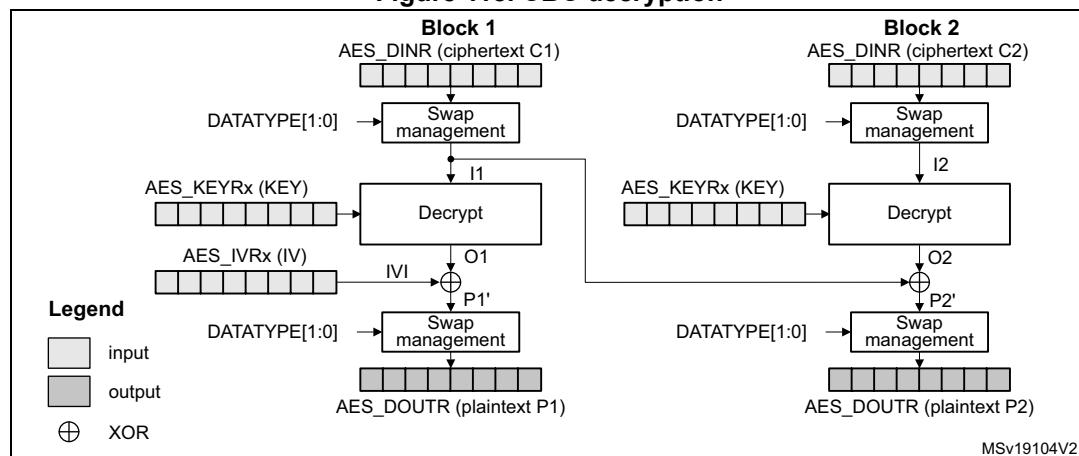


In CBC encrypt mode, the first plaintext input block, after bit/byte/half-word swapping (P1'), is XOR-ed with a 128-bit IVI bitfield (initialization vector and counter), producing the I1 input data for encrypt with the AES core, using a 128- or 256-bit key. The resulting 128-bit output block O1, after swapping operation, is used as ciphertext C1. The O1 data is then XOR-ed with the second-block plaintext data P2' to produce the I2 input data for the AES core to produce the second block of ciphertext data. The chaining of data blocks continues in this way until the last plaintext block in the message is encrypted.

If the message size is not a multiple of 128 bits, the final partial data block is encrypted in the way explained in [Section 17.4.6: AES ciphertext stealing and data padding](#).

Figure 118 illustrates the cipher block chaining (CBC) decryption mode.

Figure 118. CBC decryption



In CBC decrypt mode, like in ECB decrypt mode, the secret key must be prepared to perform an AES decryption.

After the key preparation process, the decryption goes as follows: the first 128-bit ciphertext block (after the swap operation) is used directly as the AES core input block I1 for decrypt operation, using the 128-bit or 256-bit key. Its output O1 is XOR-ed with the 128-bit IV1 field (that must be identical to that used during encryption) to produce the first plaintext block P1.

The second ciphertext block is processed in the same way as the first block, except that the I1 data from the first block is used in place of the initialization vector.

The decryption continues in this way until the last complete ciphertext block is decrypted.

If the message size is not a multiple of 128 bits, the final partial data block is decrypted in the way explained in [Section 17.4.6: AES ciphertext stealing and data padding](#).

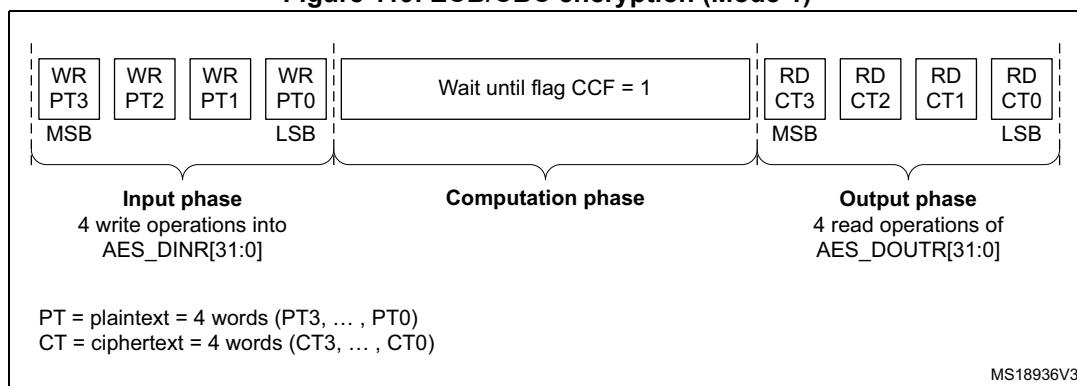
For more information on data swapping, refer to [Section 17.4.13: .AES data registers and data swapping](#).

ECB/CBC encryption sequence

The sequence of events to perform an ECB/CBC encryption (more detail in [Section 17.4.4](#)):

1. Disable the AES peripheral by clearing the EN bit of the AES_CR register.
2. Select the Mode 1 by to 00 the MODE[1:0] bitfield of the AES_CR register and select ECB or CBC chaining mode by setting the CHMOD[2:0] bitfield of the AES_CR register to 000 or 001, respectively. Data type can also be defined, using DATATYPE[1:0] bitfield.
3. Select 128- or 256-bit key length through the KEYSIZE bit of the AES_CR register.
4. Write the AES_KEYRx registers (128 or 256 bits) with encryption key. Fill the AES_IVRx registers with the initialization vector data if CBC mode has been selected.
5. Enable the AES peripheral by setting the EN bit of the AES_CR register.
6. Write the AES_DINR register four times to input the plaintext (MSB first), as shown in [Figure 119](#).
7. Wait until the CCF flag is set in the AES_SR register.
8. Read the AES_DOUTR register four times to get the ciphertext (MSB first) as shown in [Figure 119](#). Then clear the CCF flag by setting the CCFC bit of the AES_CR register.
9. Repeat steps 6,7,8 to process all the blocks with the same encryption key.

Figure 119. ECB/CBC encryption (Mode 1)

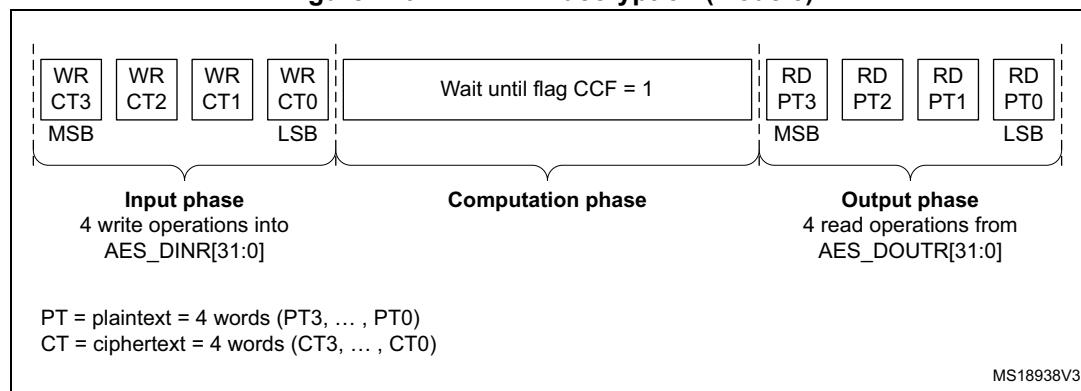


ECB/CBC decryption sequence

The sequence of events to perform an AES ECB/CBC decryption is as follows (more detail in [Section 17.4.4](#)):

1. Follow the steps described in [Section 17.4.5: AES decryption key preparation on page 468](#), in order to prepare the decryption key in AES core.
2. Disable the AES peripheral by clearing the EN bit of the AES_CR register.
3. Select the Mode 3 by setting to 10 the MODE[1:0] bitfield of the AES_CR register and select ECB or CBC chaining mode by setting the CHMOD[2:0] bitfield of the AES_CR register to 000 or 001, respectively. Data type can also be defined, using DATATYPE[1:0] bitfield.
4. Select key length of 128 or 256 bits via KEYSIZE bitfield of the AES_CR register.
5. Write the AES_IVRx registers with the initialization vector (required in CBC mode only).
6. Enable AES by setting the EN bit of the AES_CR register.
7. Write the AES_DINR register four times to input the cipher text (MSB first), as shown in [Figure 120](#).
8. Wait until the CCF flag is set in the AES_SR register.
9. Read the AES_DOUTR register four times to get the plain text (MSB first), as shown in [Figure 120](#). Then clear the CCF flag by setting the CCFC bit of the AES_CR register.
10. Repeat steps 7,8,9 to process all the blocks encrypted with the same key.

Figure 120. ECB/CBC decryption (Mode 3)



Suspend/resume operations in ECB/CBC modes

To suspend the processing of a message, proceed as follows:

1. If DMA is used, stop the AES DMA transfers to the IN FIFO by clearing the DMAINEN bit of the AES_CR register.
2. If DMA is not used, read four times the AES_DOUTR register to save the last processed block. If DMA is used, wait until the CCF flag is set in the AES_SR register

- then stop the DMA transfers from the OUT FIFO by clearing the DMAOUTEN bit of the AES_CR register.
3. If DMA is not used, poll the CCF flag of the AES_SR register until it becomes 1 (computation completed).
 4. Clear the CCF flag by setting the CCFC bit of the AES_CR register.
 5. Save initialization vector registers (only required in CBC mode as AES_IVRx registers are altered during the data processing).
 6. Disable the AES peripheral by clearing the bit EN of the AES_CR register.
 7. Save the current AES configuration in the memory (except AES initialization vector values).
 8. If DMA is used, save the DMA controller status (pointers for IN and OUT data transfers, number of remaining bytes, and so on).

Note:

In point 7, the derived key information stored in AES_KEYRx registers can optionally be saved in memory if the interrupted process is a decryption. Otherwise those registers do not need to be saved as the original key value is known by the application

To resume the processing of a message, proceed as follows:

1. If DMA is used, configure the DMA controller so as to complete the rest of the FIFO IN and FIFO OUT transfers.
2. Ensure that AES is disabled (the EN bit of the AES_CR must be 0).
3. Restore the AES_CR and AES_KEYRx register setting, using the values of the saved configuration. In case of decryption, derived key information can be written in AES_KEYRx register instead of the original key value.
4. Prepare the decryption key as described in [Section 17.4.5: AES decryption key preparation](#) (only required for ECB or CBC decryption). This step is not necessary if derived key information has been loaded in AES_KEYRx registers.
5. Restore AES_IVRx registers using the saved configuration (only required in CBC mode).
6. Enable the AES peripheral by setting the EN bit of the AES_CR register.
7. If DMA is used, enable AES DMA transfers by setting the DMAINEN and DMAOUTEN bits of the AES_CR register.

Alternative single ECB/CBC decryption using Mode 4

The sequence of events to perform a single round of ECB/CBC decryption using Mode 4 is:

1. Disable the AES peripheral by clearing the EN bit of the AES_CR register.
2. Select the Mode 4 by setting to 11 the MODE[1:0] bitfield of the AES_CR register and select ECB or CBC chaining mode by setting the CHMOD[2:0] bitfield of the AES_CR register to 000 or 001, respectively.
3. Select key length of 128 or 256 bits via KEYSIZE bitfield of the AES_CR register.
4. Write the AES_KEYRx registers with the encryption key. Write the AES_IVRx registers if the CBC mode is selected.
5. Enable the AES peripheral by setting the EN bit of the AES_CR register.
6. Write the AES_DINR register four times to input the cipher text (MSB first).
7. Wait until the CCF flag is set in the AES_SR register.
8. Read the AES_DOUTR register four times to get the plain text (MSB first). Then clear the CCF flag by setting the CCFC bit of the AES_CR register.

Note: When mode 4 is selected mode 3 cannot be used.

In mode 4, the AES_KEYRx registers contain the encryption key during all phases of the processing. No derivation key is stored in these registers. It is stored internally in AES.

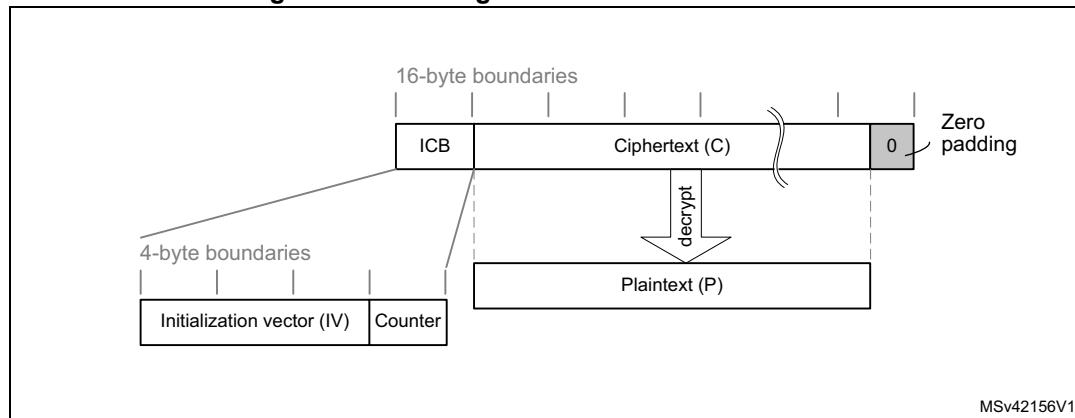
17.4.9 AES counter (CTR) mode

Overview

The counter mode (CTR) uses AES as a key-stream generator. The generated keys are then XOR-ed with the plaintext to obtain the ciphertext.

CTR chaining is defined in NIST Special Publication 800-38A, *Recommendation for Block Cipher Modes of Operation*. A typical message construction in CTR mode is given in [Figure 121](#).

Figure 121. Message construction in CTR mode



The structure of this message is:

- A 16-byte initial counter block (ICB), composed of two distinct fields:
 - **Initialization vector (IV)**: a 96-bit value that must be unique for each encryption cycle with a given key.
 - **Counter**: a 32-bit big-endian integer that is incremented each time a block processing is completed. The initial value of the counter should be set to 1.
- The plaintext P is encrypted as ciphertext C, with a known length. This length can be non-multiple of 16 bytes, in which case a plaintext padding is required.

CTR encryption and decryption

[Figure 122](#) and [Figure 123](#) describe the CTR encryption and decryption process, respectively, as implemented in the AES peripheral. The CTR mode is selected by writing 010 to the CHMOD[2:0] bitfield of AES_CR register.

Figure 122. CTR encryption

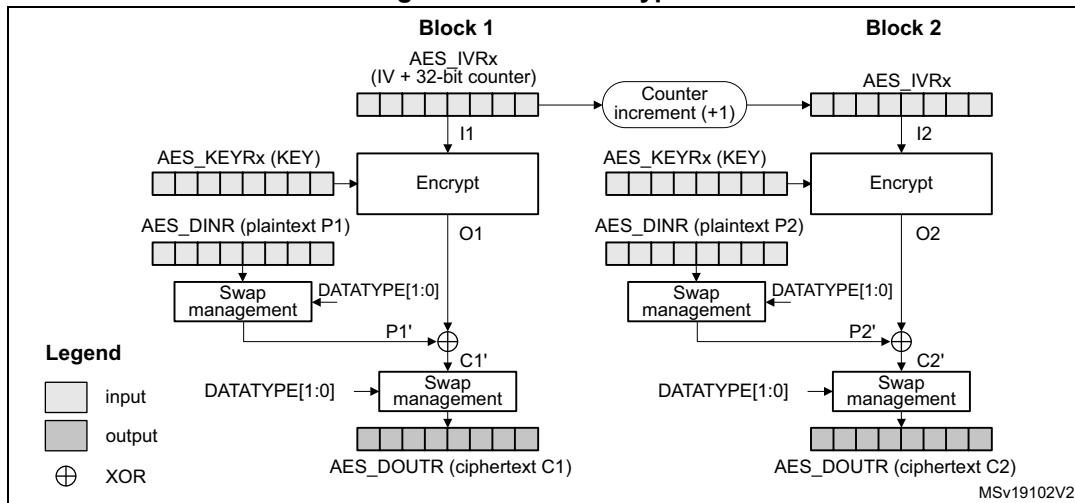
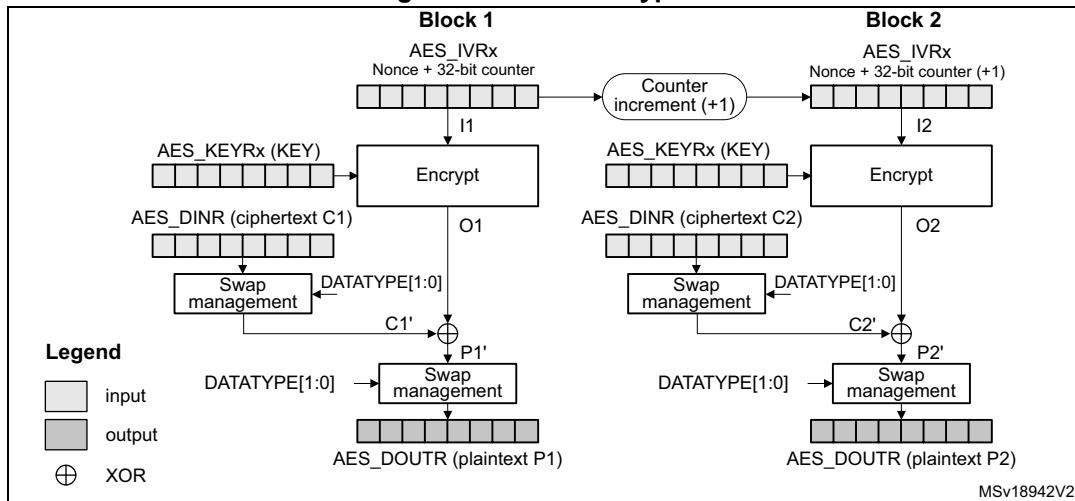


Figure 123. CTR decryption



In CTR mode, the cryptographic core output (also called keystream) O_x is XOR-ed with relevant input block (P_x' for encryption, C_x' for decryption), to produce the correct output block (C_x' for encryption, P_x' for decryption). Initialization vectors in AES must be initialized as shown in [Table 101](#).

Table 101. CTR mode initialization vector definition

AES_IVR3[31:0]	AES_IVR2[31:0]	AES_IVR1[31:0]	AES_IVR0[31:0]
Nonce[31:0]	Nonce[63:32]	Nonce[95:64]	32-bit counter = 0x0001

Unlike in CBC mode that uses the AES_IVRx registers only once when processing the first data block, in CTR mode AES_IVRx registers are used for processing each data block, and the AES peripheral increments the counter bits of the initialization vector (leaving the nonce bits unchanged).

CTR decryption does not differ from CTR encryption, since the core always encrypts the current counter block to produce the key stream that is then XOR-ed with the plaintext (CTR).

encryption) or ciphertext (CTR decryption) input. In CTR mode, the MODE[1:0] bitfield settings 11, 10 or 00 default all to encryption mode, and the setting 01 (key derivation) is forbidden.

The sequence of events to perform an encryption or a decryption in CTR chaining mode:

1. Ensure that AES is disabled (the EN bit of the AES_CR must be 0).
2. Select CTR chaining mode by setting to 010 the CHMOD[2:0] bitfield of the AES_CR register. Set MODE[1:0] bitfield to any value other than 01.
3. Initialize the AES_KEYRx registers, and load the AES_IVRx registers as described in [Table 101](#).
4. Set the EN bit of the AES_CR register, to start encrypting the current counter (EN is automatically reset when the calculation finishes).
5. If it is the last block, pad the data with zeros to have a complete block, if needed.
6. Append data in AES, and read the result. The three possible scenarios are described in [Section 17.4.4: AES procedure to perform a cipher operation](#).
7. Repeat the previous step till the second-last block is processed. For the last block, apply the two previous steps and discard the bits that are not part of the payload (if the size of the significant data in the last input block is less than 16 bytes).

Suspend/resume operations in CTR mode

Like for the CBC mode, it is possible to interrupt a message to send a higher priority message, and resume the message that was interrupted. Detailed CBC suspend/resume sequence is described in [Section 17.4.8: AES basic chaining modes \(ECB, CBC\)](#).

Note: Like for CBC mode, the AES_IVRx registers must be reloaded during the resume operation.

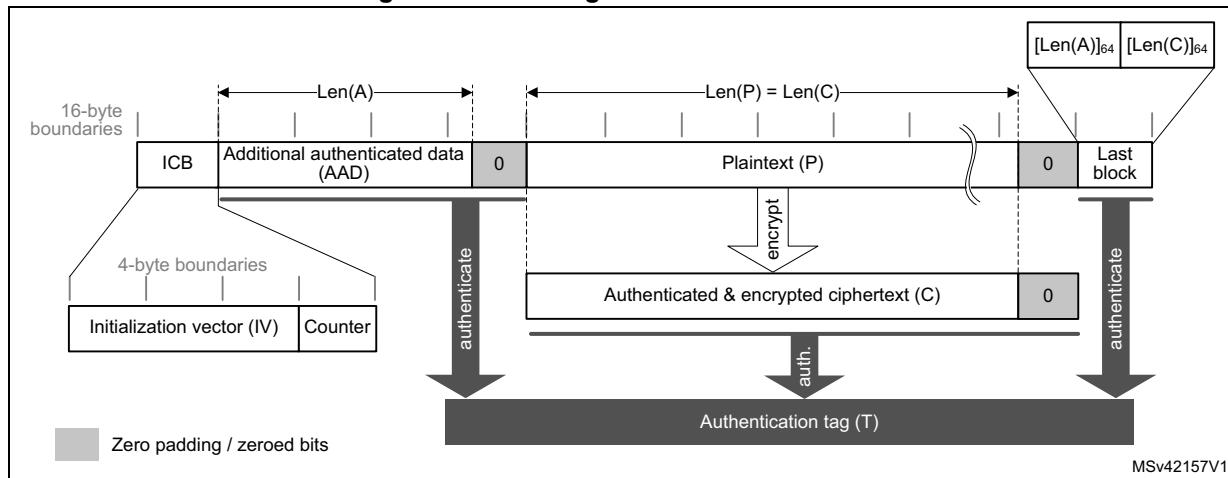
17.4.10 AES Galois/counter mode (GCM)

Overview

The AES Galois/counter mode (GCM) allows encrypting and authenticating a plaintext message into the corresponding ciphertext and tag (also known as message authentication code). To ensure confidentiality, GCM algorithm is based on AES counter mode. It uses a multiplier over a fixed finite field to generate the tag.

GCM chaining is defined in NIST Special Publication 800-38D, *Recommendation for Block Cipher Modes of Operation - Galois/Counter Mode (GCM) and GMAC*. A typical message construction in GCM mode is given in [Figure 124](#).

Figure 124. Message construction in GCM



The message has the following structure:

- **16-byte initial counter block (ICB)**, composed of two distinct fields:
 - **Initialization vector (IV)**: a 96-bit value that must be unique for each encryption cycle with a given key. Note that the GCM standard supports IVs with less than 96 bits, but in this case strict rules apply.
 - **Counter**: a 32-bit big-endian integer that is incremented each time a block processing is completed. According to NIST specification, the counter value is 0x2 when processing the first block of payload.
- **Authenticated header AAD** (also known as additional authentication data) has a known length **Len(A)** that may be a non-multiple of 16 bytes, and must not exceed $2^{64} - 1$ bits. This part of the message is only authenticated, not encrypted.
- **Plaintext message P** is both authenticated and encrypted as ciphertext C, with a known length **Len(P)** that may be non-multiple of 16 bytes, and cannot exceed $2^{32} - 2$ 128-bit blocks.
- **Last block** contains the AAD header length (bits [32:63]) and the payload length (bits [96:127]) information, as shown in [Table 102](#).

The GCM standard specifies that ciphertext C has the same bit length as the plaintext P.

When a part of the message (AAD or P) has a length that is a non-multiple of 16-bytes a special padding scheme is required.

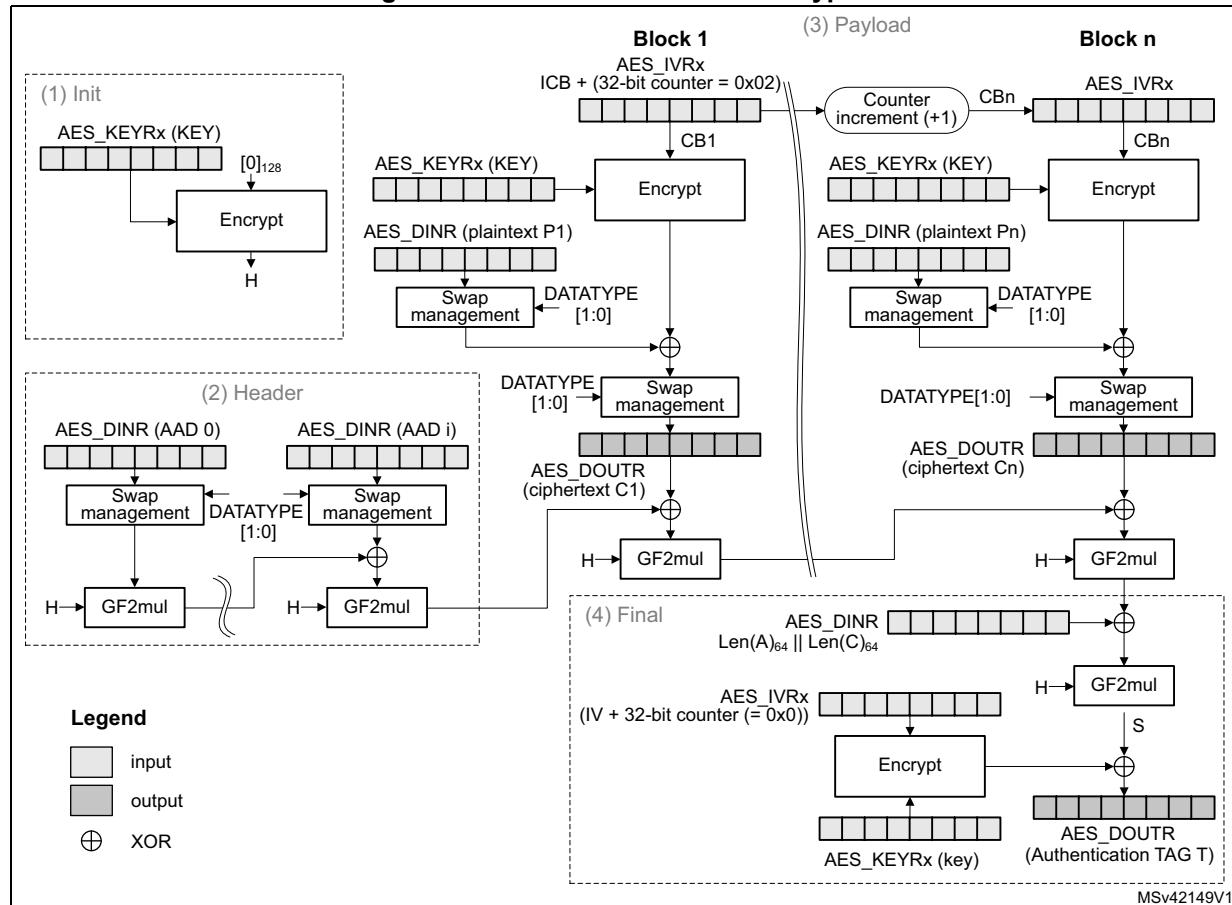
Table 102. GCM last block definition

Endianness	Bit[0] ----- Bit[31]	Bit[32]----- Bit[63]	Bit[64] ----- Bit[95]	Bit[96] ----- Bit[127]
Input data	0x0	AAD length[31:0]	0x0	Payload length[31:0]

GCM processing

[Figure 125](#) describes the GCM implementation in the AES peripheral. The GCM is selected by writing 011 to the CHMOD[2:0] bitfield of the AES_CR register.

Figure 125. GCM authenticated encryption



The mechanism for the confidentiality of the plaintext in GCM mode is similar to that in the Counter mode, with a particular increment function (denoted 32-bit increment) that generates the sequence of input counter blocks.

AES_IVRx registers keeping the **counter block** of data are used for processing each data block. The AES peripheral automatically increments the Counter[31:0] bitfield. The first counter block (CB1) is derived from the initial counter block ICB by the application software (see [Table 103](#)).

Table 103. GCM mode IVI bitfield initialization

Register	AES_IVR3[31:0]	AES_IVR2[31:0]	AES_IVR1[31:0]	AES_IVR0[31:0]
Input data	ICB[31:0]	ICB[63:32]	ICB[95:64]	Counter[31:0] = 0x2

Note: In GCM mode, the settings 01 and 11 of the MODE[1:0] bitfield are forbidden.

The authentication mechanism in GCM mode is based on a hash function called **GF2mul** that performs multiplication by a fixed parameter, called hash subkey (H), within a binary Galois field.

A GCM message is processed through the following phases, further described in next subsections:

- **Init phase:** AES prepares the GCM hash subkey (H).
- **Header phase:** AES processes the additional authenticated data (AAD), with hash computation only.
- **Payload phase:** AES processes the plaintext (P) with hash computation, counter block encryption and data XOR-ing. It operates in a similar way for ciphertext (C).
- **Final phase:** AES generates the authenticated tag (T) using the last block of the message.

GCM init phase

During this first step, the GCM hash subkey (H) is calculated and saved internally, to be used for processing all the blocks. The recommended sequence is:

1. Ensure that AES is disabled (the EN bit of the AES_CR must be 0).
2. Select GCM chaining mode, by setting to 011 the CHMOD[2:0] bitfield of the AES_CR register, and set to 00 (no data swapping) the DATATYPE[1:0] bitfield.
3. Indicate the Init phase, by setting to 00 the GCMPH[1:0] bitfield of the AES_CR register.
4. Set the MODE[1:0] bitfield of the AES_CR register to 00 or 10.
5. Initialize the AES_KEYRx registers with a key, and initialize AES_IVRx registers with the information as defined in [Table 103](#).
6. Start the calculation of the hash key, by setting to 1 the EN bit of the AES_CR register (EN is automatically reset when the calculation finishes).
7. Wait until the end of computation, indicated by the CCF flag of the AES_SR transiting to 1. Alternatively, use the corresponding interrupt.
8. Clear the CCF flag of the AES_SR register, by setting to 1 the CCFC bit of the AES_CR register, and optionally set the data type (1-, 8- or 16-bit) using the DATATYPE[1:0] bitfield.

GCM header phase

This phase coming after the GCM Init phase must be completed before the payload phase. The sequence to execute, identical for encryption and decryption, is:

1. Indicate the header phase, by setting to 01 the GCMPH[1:0] bitfield of the AES_CR register. Do not modify the MODE[1:0] bitfield as set in the Init phase.
2. Enable the AES peripheral by setting the EN bit of the AES_CR register.
3. If it is the last block and the AAD size in the block is inferior to 128 bits, pad the remainder of the block with zeros. Then append the data block into AES in one of ways described in [Section 17.4.4: AES procedure to perform a cipher operation on page 464](#).
4. Repeat the step 3 until the last additional authenticated data block is processed.

Note:

The header phase can be skipped if there is no AAD, that is, Len(A) = 0.

GCM payload phase

This phase, identical for encryption and decryption, is executed after the GCM header phase. During this phase, the encrypted/decrypted payload is stored in the AES_DOUTR register. The sequence to execute is:

1. If the header phase was skipped, enable the AES peripheral by setting the EN bit of the AES_CR register.
2. Indicate the payload phase, by setting to 10 the GCMPH[1:0] bitfield of the AES_CR register. Do not modify the MODE[1:0] bitfield as set in the Init phase.
3. If it is the last block and the plaintext (encryption) or ciphertext (decryption) size in the block is inferior to 128 bits, pad the remainder of the block with zeros.
4. Append the data block into AES in one of ways described in [Section 17.4.4: AES procedure to perform a cipher operation on page 464](#), and read the result.
5. Repeat the previous step till the second-last plaintext block is encrypted or till the last block of ciphertext is decrypted. For the last block of plaintext (encryption only), execute the two previous steps. For the last block, discard the bits that are not part of the payload when the last block size is less than 16 bytes.

Note: *The payload phase can be skipped if there is no payload data, that is, Len(C) = 0 (see GMAC mode).*

GCM final phase

In this last phase, the AES peripheral generates the GCM authentication tag and stores it in the AES_DOUTR register. The sequence to execute is:

1. Indicate the final phase, by setting to 11 the GCMPH[1:0] bitfield of the AES_CR register. Select encrypt mode by setting to 00 the MODE[1:0] bitfield of the AES_CR register.
2. Compose the data of the block, by concatenating the AAD bit length and the payload bit length, as shown in [Table 102](#). Write the block into the AES_DINR register.
3. Wait until the end of computation, indicated by the CCF flag of the AES_SR transiting to 1.
4. Get the GCM authentication tag, by reading the AES_DOUTR register four times.
5. Clear the CCF flag in the AES_SR register, by setting to 1 the CCFC bit of the AES_CR register.
6. Disable the AES peripheral, by clearing the bit EN of the AES_CR register. If it is an authenticated decryption, compare the generated tag with the expected tag passed with the message.

Note: *In the final phase, data must be swapped according to the data type set in the DATATYPE[1:0] bitfield of the AES_CR register.*

When transiting from the header or the payload phase to the final phase, the AES peripheral must not be disabled, otherwise the result is wrong.

Suspend/resume operations in GCM mode

To suspend the processing of a message, proceed as follows:

1. If DMA is used, stop the AES DMA transfers to the IN FIFO by clearing the DMAINEN bit of the AES_CR register. If DMA is not used, make sure that the current computation is completed, which is indicated by the CCF flag of the AES_SR register set to 1.
2. In the payload phase, if DMA is not used, read four times the AES_DOUTR register to save the last-processed block. If DMA is used, wait until the CCF flag is set in the AES_SR register then stop the DMA transfers from the OUT FIFO by clearing the DMAOUTEN bit of the AES_CR register.
3. Clear the CCF flag of the AES_SR register, by setting to 1 the CCFC bit of the AES_CR register.
4. Save the AES_SUSPxR registers in the memory, where x is from 0 to 7.
5. In the payload phase, save the AES_IVRx registers as, during the data processing, they changed from their initial values. In the header phase, this step is not required.
6. Disable the AES peripheral, by clearing the EN bit of the AES_CR register.
7. Save the current AES configuration in the memory, excluding the initialization vector registers AES_IVRx. Key registers do not need to be saved as the original key value is known by the application.
8. If DMA is used, save the DMA controller status (pointers for IN data transfers, number of remaining bytes, and so on). In the payload phase, pointers for OUT data transfers must also be saved.

To resume the processing of a message, proceed as follows:

1. If DMA is used, configure the DMA controller in order to complete the rest of the FIFO IN transfers. In the payload phase, the rest of the FIFO OUT transfers must also be configured in the DMA controller.
2. Ensure that the AES peripheral is disabled (the EN bit of the AES_CR register must be 0).
3. Write the suspend register values, previously saved in the memory, back into their corresponding AES_SUSPxR registers, where x is from 0 to 7.
4. In the payload phase, write the initialization vector register values, previously saved in the memory, back into their corresponding AES_IVRx registers. In the header phase, write initial setting values back into the AES_IVRx registers.
5. Restore the initial setting values in the AES_CR and AES_KEYRx registers.
6. Enable the AES peripheral by setting the EN bit of the AES_CR register.
7. If DMA is used, enable AES DMA requests by setting the DMAINEN bit (and DMAOUTEN bit if in payload phase) of the AES_CR register.

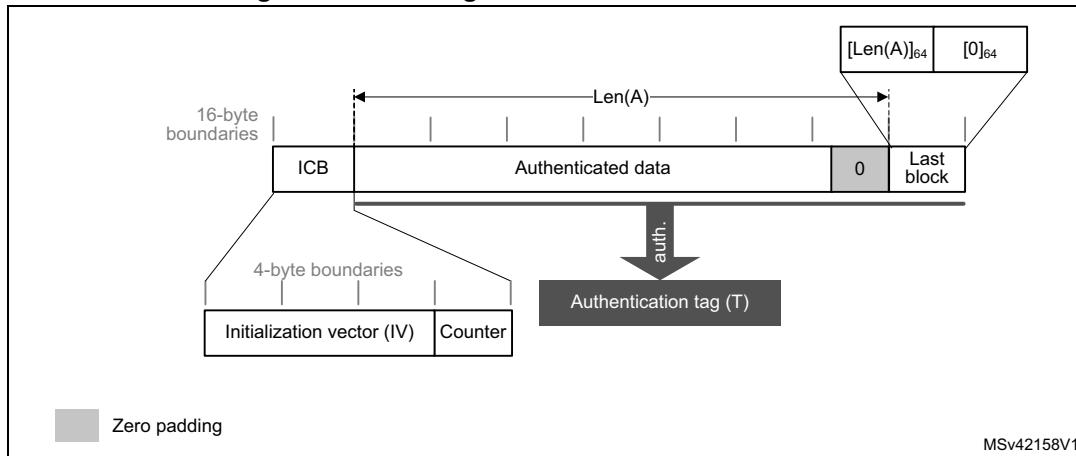
17.4.11 AES Galois message authentication code (GMAC)

Overview

The Galois message authentication code (GMAC) allows the authentication of a plaintext, generating the corresponding tag information (also known as message authentication code). It is based on GCM algorithm, as defined in NIST Special Publication 800-38D, *Recommendation for Block Cipher Modes of Operation - Galois/Counter Mode (GCM) and GMAC*.

A typical message construction for GMAC is given in [Figure 126](#).

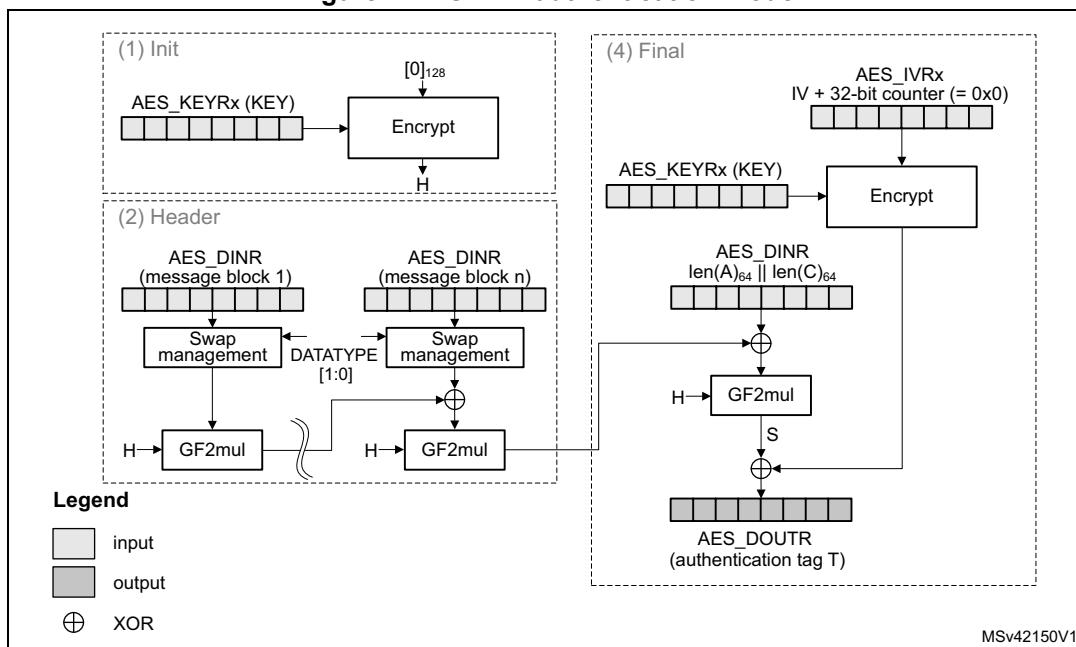
Figure 126. Message construction in GMAC mode



AES GMAC processing

[Figure 127](#) describes the GMAC mode implementation in the AES peripheral. This mode is selected by writing 011 to the CHMOD[2:0] bitfield of the AES_CR register.

Figure 127. GMAC authentication mode



The GMAC algorithm corresponds to the GCM algorithm applied on a message only containing a header. As a consequence, all steps and settings are the same as with the GCM, except that the payload phase is omitted.

Suspend/resume operations in GMAC

In GMAC mode, the sequence described for the GCM applies except that only the header phase can be interrupted.

17.4.12 AES counter with CBC-MAC (CCM)

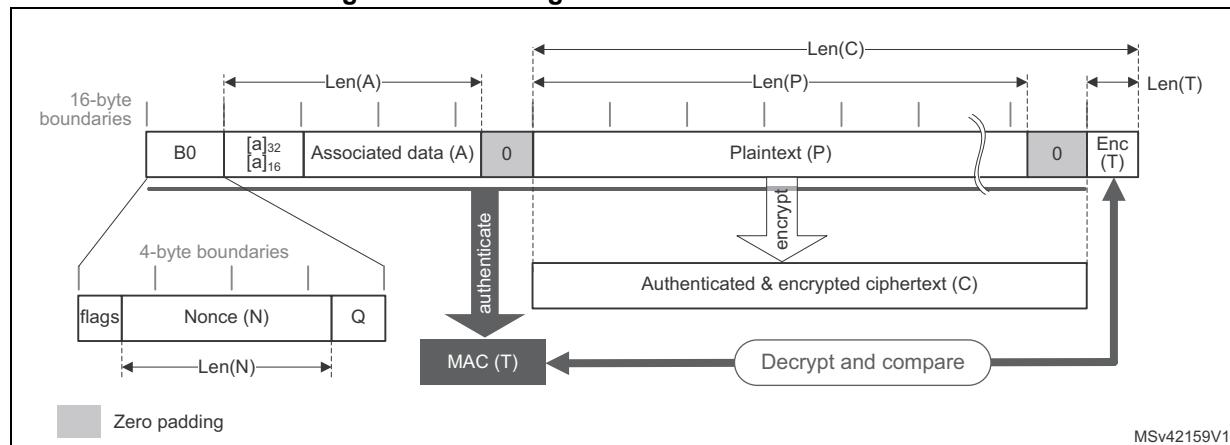
Overview

The AES **counter with cipher block chaining-message authentication code** (CCM) algorithm allows encryption and authentication of plaintext, generating the corresponding ciphertext and tag (also known as message authentication code). To ensure confidentiality, the CCM algorithm is based on AES in counter mode. It uses cipher block chaining technique to generate the message authentication code. This is commonly called CBC-MAC.

Note: *NIST does not approve this CBC-MAC as an authentication mode outside the context of the CCM specification.*

CCM chaining is specified in NIST Special Publication 800-38C, *Recommendation for Block Cipher Modes of Operation - The CCM Mode for Authentication and Confidentiality*. A typical message construction for CCM is given in [Figure 128](#).

Figure 128. Message construction in CCM mode



The structure of the message is:

- **16-byte first authentication block (B0)**, composed of three distinct fields:
 - **Q**: a bit string representation of the octet length of P (Len(P))
 - **Nonce (N)**: a single-use value (that is, a new nonce should be assigned to each new communication) of Len(N) size. The sum Len(N) + Len(P) must be equal to 15 bytes.
 - **Flags**: most significant octet containing four flags for control information, as specified by the standard. It contains two 3-bit strings to encode the values **t** (MAC length expressed in bytes) and **Q** (plaintext length such that Len(P) < 2^{8q} bytes). The counter blocks range associated to **Q** is equal to 2^{8Q-4} , that is, if the maximum value of **Q** is 8, the counter blocks used in cipher shall be on 60 bits.
- **16-byte blocks (B)** associated to the Associated Data (A). This part of the message is only authenticated, not encrypted. This section has a known length Len(A) that can be a non-multiple of 16 bytes (see [Figure 128](#)). The

standard also states that, on MSB bits of the first message block (B1), the associated data length expressed in bytes (a) must be encoded as follows:

- If $0 < a < 2^{16} - 2^8$, then it is encoded as $[a]_{16}$, that is, on two bytes.
- If $2^{16} - 2^8 < a < 2^{32}$, then it is encoded as $0xff \parallel 0xfe \parallel [a]_{32}$, that is, on six bytes.
- If $2^{32} < a < 2^{64}$, then it is encoded as $0xff \parallel 0xff \parallel [a]_{64}$, that is, on ten bytes.

- **16-byte blocks (B)** associated to the plaintext message P, which is both authenticated and encrypted as ciphertext C, with a known length Len(P). This length can be a non-multiple of 16 bytes (see [Figure 128](#)).
- **Encrypted MAC (T)** of length Len(T) appended to the ciphertext C of overall length Len(C).

When a part of the message (A or P) has a length that is a non-multiple of 16-bytes, a special padding scheme is required.

Note: *CCM chaining mode can also be used with associated data only (that is, no payload).*

As an example, the C.1 section in NIST Special Publication 800-38C gives the following values (hexadecimal numbers):

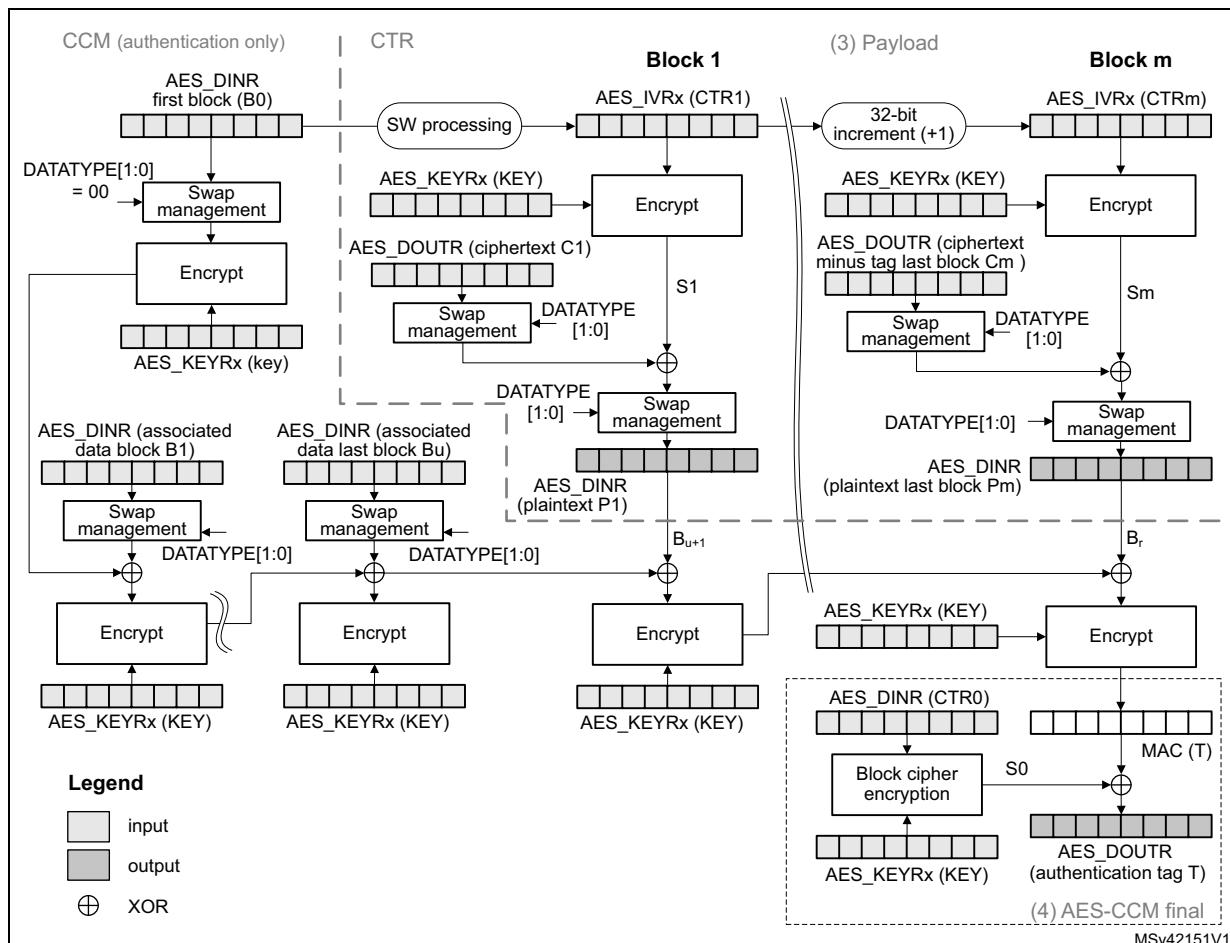
N: 10111213 141516 (Len(N)= 56 bits or 7 bytes)
A: 00010203 04050607 (Len(A)= 64 bits or 8 bytes)
P: **20212223** (Len(P)= 32 bits or 4 bytes)
T: 6084341B (Len(T)= 32 bits or t = 4)
B0: 4F101112 13141516 00000000 0000000**4**
B1: 0008**0001** **02030405** **06070000** 00000000
B2: **20212223** 00000000 00000000 00000000
CTR0: 0710111213 141516 00000000 00000000
CTR1: 0710111213 141516 00000000 00000001

Generation of formatted input data blocks Bx (especially B0 and B1) must be managed by the application.

CCM processing

[Figure 129](#) describes the CCM implementation within the AES peripheral (decryption example).

Figure 129. CCM mode authenticated decryption



The data input to the generation-encryption process are a valid nonce, a valid payload string, and a valid associated data string, all properly formatted. The CBC chaining mechanism is applied to the formatted plaintext data to generate a MAC, with a known length. Counter mode encryption that requires a sufficiently long sequence of counter blocks as input, is applied to the payload string and separately to the MAC. The resulting ciphertext C is the output of the generation-encryption process on plaintext P.

AES_IVRx registers are used for processing each data block, AES automatically incrementing the CTR counter with a bit length defined by the first block B0. [Table 104](#) shows how the application must load the B0 data.

Table 104. Initialization of AES_IVRx registers in CCM mode

Register	AES_IVR3[31:0]	AES_IVR2[31:0]	AES_IVR1[31:0]	AES_IVR0[31:0]
Input data	B0[31:0]	B0[63:32]	B0[95:64]	B0[127:96]

A CCM message is processed through two distinct processes - first, **payload encryption or decryption**, in which the AES peripheral is configured in CTR mode, then **associated data and payload authentication**, in which the AES peripheral first executes the CCM header phase, then the CCM final phase.

Payload encryption/decryption

This step is performed independently of the tag computation. It uses standard CTR chaining mode. Refer to [Section 17.4.9: AES counter \(CTR\) mode](#) for details. The construction of the CTR1 initialization vector (see [Figure 129](#)) to load into AES_IVRx registers is defined in NIST Special Publication 800-38C.

Note: *This phase can be skipped if there is no payload data, that is, when Len(P) = 0 or Len(C) = Len(T).*

Remove LSB_{Len(T)}(C) encrypted tag information when decrypting ciphertext C.

Associated data and payload authentication

In order to compute the CCM authentication tag associated with the plaintext message, it is recommended to execute the following header phase sequence:

1. Ensure that the AES peripheral is disabled (the EN bit of the AES_CR must be 0).
2. Select CCM chaining mode, by setting to 100 the CHMOD[2:0] bitfield of the AES_CR register, and optionally, set the DATATYPE[1:0] bitfield.
3. Indicate the header phase, by setting to 01 the GCMPH[1:0] bitfield of the AES_CR register. Select encrypt mode by setting to 00 the MODE[1:0] bitfield of the AES_CR register.
4. Initialize the AES_KEYRx registers with a key, and initialize AES_IVRx registers with zero values.
5. Enable the AES peripheral by setting the EN bit of the AES_CR register.
6. Write the AES_DINR register with B0, as shown in [Table 104](#). B0 data must be swapped according to the DATATYPE[1:0] bitfield of the AES_CR register.
7. Wait until the end-of-computation flag CCF of the AES_SR register is set to 1.
8. Clear the CCF flag of the AES_SR register by setting the CCFC bit of the AES_CR register.
9. Process data block. If it is the last block of associated data or plaintext and data size in the block is inferior to 128 bits, pad the remainder of the block with zeros. Then append the data block into AES in one of ways described in [Section 17.4.4: AES procedure to perform a cipher operation on page 464](#).
10. Repeat the previous step to process all data blocks, starting from the first block of associated data and ending with the last block of plaintext payload data.

In final phase, the AES peripheral generates the CCM authentication tag and stores it in the AES_DOUTR register:

11. Indicate the final phase, by setting to 11 the GCMPH[1:0] bitfield of the AES_CR register. Keep as-is the encryption mode in the MODE[1:0] bitfield.
12. Write four times the last data input into the AES_DIN register. This input must be the 128-bit value CTR0, formatted from the original B0 packet (that is, 5 flag bits set to 0, and Q length bits set to 0).
13. Wait until the end-of-computation flag CCF of the AES_SR register is set.
14. Read four times the AES_DOUTR register: the output corresponds to the encrypted CCM authentication tag.
15. Clear the CCF flag of the AES_SR register by setting the CCFC bit of the AES_CR register.
16. Disable the AES peripheral, by clearing the EN bit of the AES_CR register.
17. For authenticated decryption, compare the generated encrypted tag with the encrypted tag padded in the ciphertext.

Note:

In this final phase, data must be swapped according to the DATATYPE[1:0] bitfield of the AES_CR register.

When transiting from the header phase to the final phase, the AES peripheral must not be disabled, otherwise the result is wrong.

Application must mask the authentication tag output with tag length to obtain a valid tag.

Suspend/resume operations in CCM mode

To suspend the authentication of the associated data and payload (GCMPH[1:0]= 01), proceed as follows. Suspending the message during the encryption/decryption phase is described in [Section 17.4.9: AES counter \(CTR\) mode on page 476](#).

1. If DMA is used, stop the AES DMA transfers to the IN FIFO by clearing the DMAINEN bit of the AES_CR register. If DMA is not used, make sure that the current computation is completed, which is indicated by the CCF flag of the AES_SR register set to 1.
2. Clear the CCF flag of the AES_SR register, by setting to 1 the CCFC bit of the AES_CR register.
3. Save the AES_SUSPxR registers (where x is from 0 to 7) in the memory.
4. Save the AES_IVRx registers, as during the data processing they changed from their initial values.
5. Disable the AES peripheral, by clearing the bit EN of the AES_CR register.
6. Save the current AES configuration in the memory, excluding the initialization vector registers AES_IVRx. Key registers do not need to be saved as the original key value is known by the application.
7. If DMA is used, save the DMA controller status (pointers for IN data transfers, number of remaining bytes, and so on).

To resume the authentication of the associated data and payload (GCMFH[1:0]= 01 or 11), proceed as follows:

1. If DMA is used, configure the DMA controller in order to complete the rest of the FIFO IN transfers.
2. Ensure that AES processor is disabled (the EN bit of the AES_CR register must be 0).
3. Write the suspend register values, previously saved in the memory, back into their corresponding AES_SUSPxR registers (where x is from 0 to 7).
4. Write the initialization vector register values, previously saved in the memory, back into their corresponding AES_IVRx registers.
5. Restore the initial setting values in the AES_CR and AES_KEYRx registers.
6. Enable the AES peripheral by setting the EN bit of the AES_CR register.
7. If DMA is used, enable AES DMA requests by setting the DMAINEN bit of the AES_CR register.

Note: In CCM mode the MODE[1:0] bitfield settings 01 and 11 (key derivation) are forbidden.

17.4.13 AES data registers and data swapping

Data input and output

A 128-bit data block is entered into the AES peripheral with four successive 32-bit word writes into the AES_DINR register (bitfield DIN[127:0]), the most significant word (bits [127:96]) first, the least significant word (bits [31:0]) last.

A 128-bit data block is retrieved from the AES peripheral with four successive 32-bit word reads from the AES_DOUTR register (bitfield DOUT[127:0]), the most significant word (bits [127:96]) first, the least significant word (bits [31:0]) last.

The 32-bit data word for AES_DINR register or from AES_DOUTR register is organized in big endian order, that is:

- the most significant byte of a word to write into AES_DINR must be put on the lowest address out of the four adjacent memory locations keeping the word to write, or
- the most significant byte of a word read from AES_DOUTR goes to the lowest address out of the four adjacent memory locations receiving the word

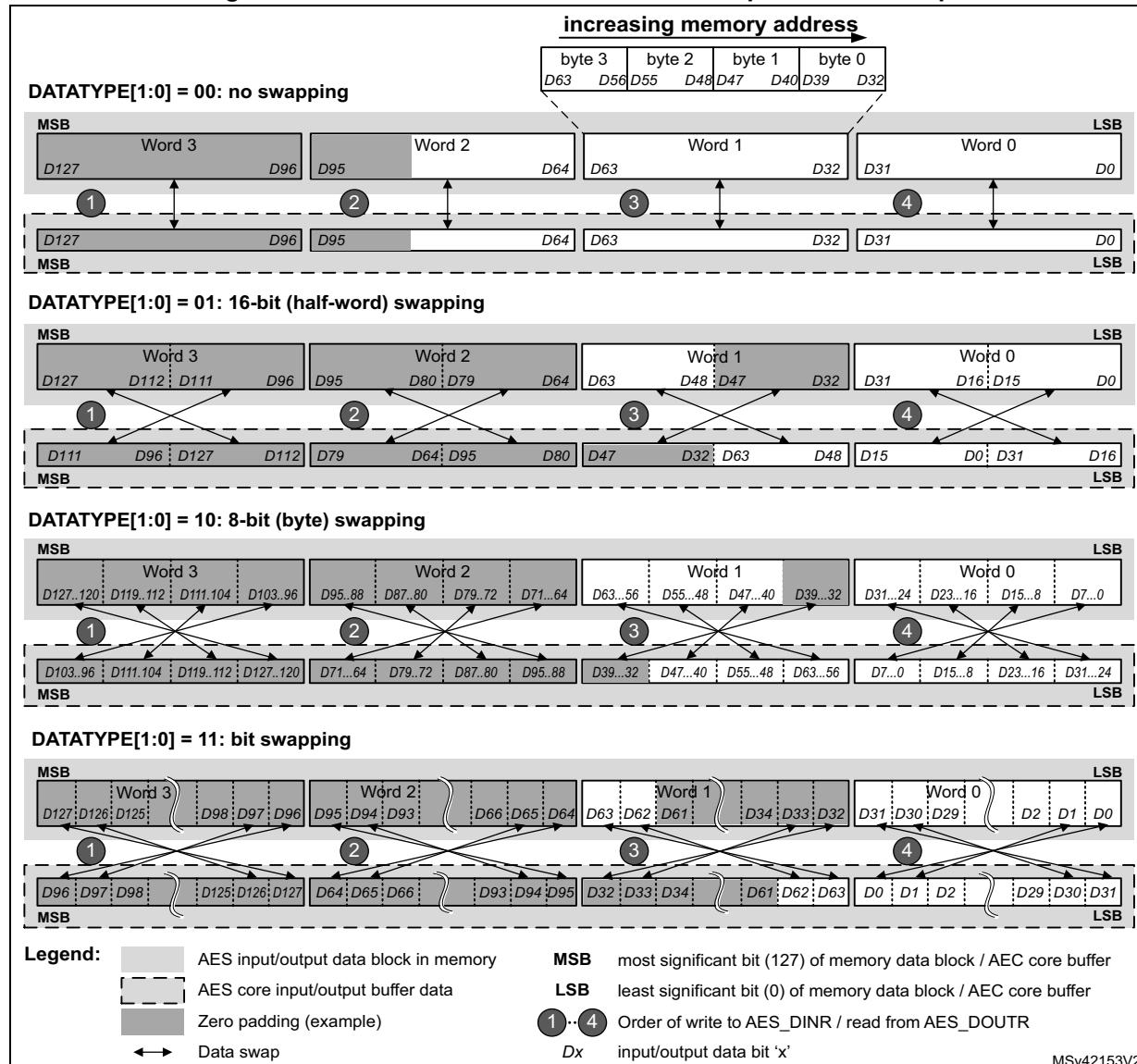
For using DMA for input data block write into AES, the four words of the input block must be stored in the memory consecutively and in big-endian order, that is, the most significant word on the lowest address. See [Section 17.4.16: AES DMA interface](#).

Data swapping

The AES peripheral can be configured to perform a bit-, a byte-, a half-word-, or no swapping on the input data word in the AES_DINR register, before loading it to the AES processing core, and on the data output from the AES processing core, before sending it to the AES_DOUTR register. The choice depends on the type of data. For example, a byte swapping is used for an ASCII text stream.

The data swap type is selected through the DATATYPE[1:0] bitfield of the AES_CR register. The selection applies both to the input and the output of the AES core.

For different data swap types, [Figure 130](#) shows the construction of AES processing core input buffer data P127..0, from the input data entered through the AES_DINR register, or the construction of the output data available through the AES_DOUTR register, from the AES processing core output buffer data P127..0.

Figure 130. 128-bit block construction with respect to data swap

Note: The data in AES key registers (AES_KEYRx) and initialization registers (AES_IVRx) are not sensitive to the swap mode selection.

Data padding

Figure 130 also gives an example of memory data block padding with zeros such that the zeroed bits after the data swap form a contiguous zone at the MSB end of the AES core input buffer. The example shows the padding of an input data block containing:

- 48 message bits, with DATATYPE[1:0] = 01
- 56 message bits, with DATATYPE[1:0] = 10
- 34 message bits, with DATATYPE[1:0] = 11

17.4.14 AES key registers

The AES_KEYRx registers store the encryption or decryption key bitfield KEY[127:0] or KEY[255:0]. The data to write to or to read from each register is organized in the memory in little-endian order, that is, with most significant byte on the highest address.

The key is spread over the eight registers as shown in [Table 105](#).

Table 105. Key endianness in AES_KEYRx registers (128- or 256-bit key length)

AES_KEYR7 [31:0]	AES_KEYR6 [31:0]	AES_KEYR5 [31:0]	AES_KEYR4 [31:0]	AES_KEYR3 [31:0]	AES_KEYR2 [31:0]	AES_KEYR1 [31:0]	AES_KEYR0 [31:0]
-	-	-	-	KEY[127:96]	KEY[95:64]	KEY[63:32]	KEY[31:0]
KEY[255:224]	KEY[223:192]	KEY[191:160]	KEY[159:128]	KEY[127:96]	KEY[95:64]	KEY[63:32]	KEY[31:0]

The key for encryption or decryption may be written into these registers when the AES peripheral is disabled.

The key registers are not affected by the data swapping controlled by DATATYPE[1:0] bitfield of the AES_CR register.

17.4.15 AES initialization vector registers

The four AES_IVRx registers keep the initialization vector input bitfield IVI[127:0]. The data to write to or to read from each register is organized in the memory in little-endian order, that is, with most significant byte on the highest address. The registers are also ordered from lowest address (AES_IVR0) to highest address (AES_IVR3).

The signification of data in the bitfield depends on the chaining mode selected. When used, the bitfield is updated upon each computation cycle of the AES core.

Write operations to the AES_IVRx registers when the AES peripheral is enabled have no effect to the register contents. For modifying the contents of the AES_IVRx registers, the EN bit of the AES_CR register must first be cleared.

Reading the AES_IVRx registers returns the latest counter value (useful for managing suspend mode).

The AES_IVRx registers are not affected by the data swapping feature controlled by the DATATYPE[1:0] bitfield of the CRYP_CR register.

17.4.16 AES DMA interface

The AES peripheral provides an interface to connect to the DMA (direct memory access) controller. The DMA operation is controlled through the AES_CR register.

Data input using DMA

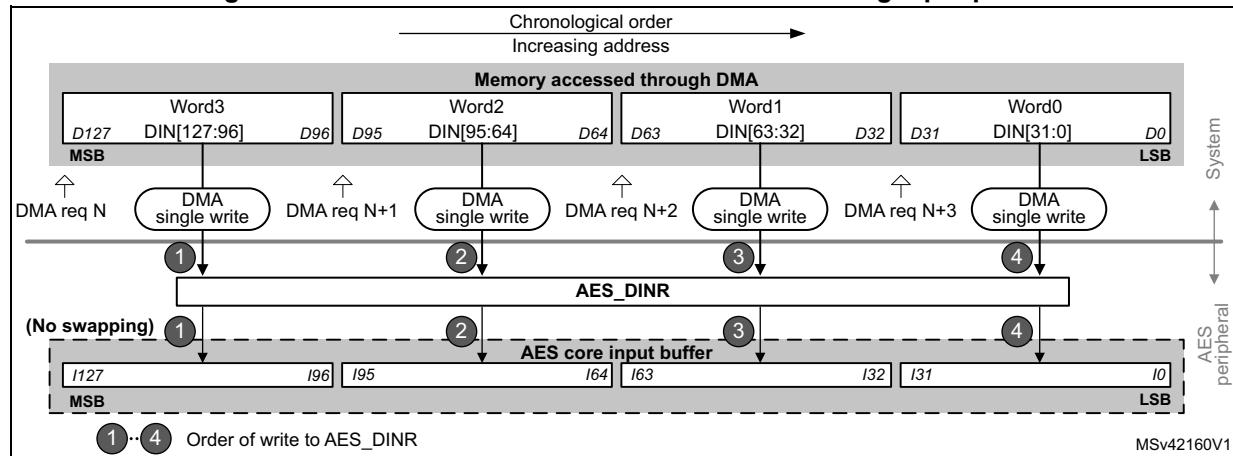
Setting the DMAINEN bit of the AES_CR register enables DMA writing into AES. The AES peripheral then initiates a DMA request during the input phase each time it requires a word to be written to the AES_DINR register. It asserts four DMA requests to transfer one 128-bit (four-word) input data block from memory, as shown in [Figure 131](#).

See [Table 106](#) for recommended DMA configuration.

Table 106. DMA channel configuration for memory-to-AES data transfer

DMA channel control register field	Recommended configuration
Transfer size	Message length: a multiple of 128 bits. According to the algorithm and the mode selected, special padding/ciphertext stealing might be required. Refer to Section 17.4.6: AES ciphertext stealing and data padding on page 469 for details.
Source burst size (memory)	Single
Destination burst size (peripheral)	Single
DMA FIFO size	AES FIFO_size = 4 bytes.
Source transfer width (memory)	32-bit words
Destination transfer width (peripheral)	32-bit words
Source address increment (memory)	Yes, after each 32-bit transfer
Destination address increment (peripheral)	Fixed address of AES_DINR (no increment)

Figure 131. DMA transfer of a 128-bit data block during input phase



Data output using DMA

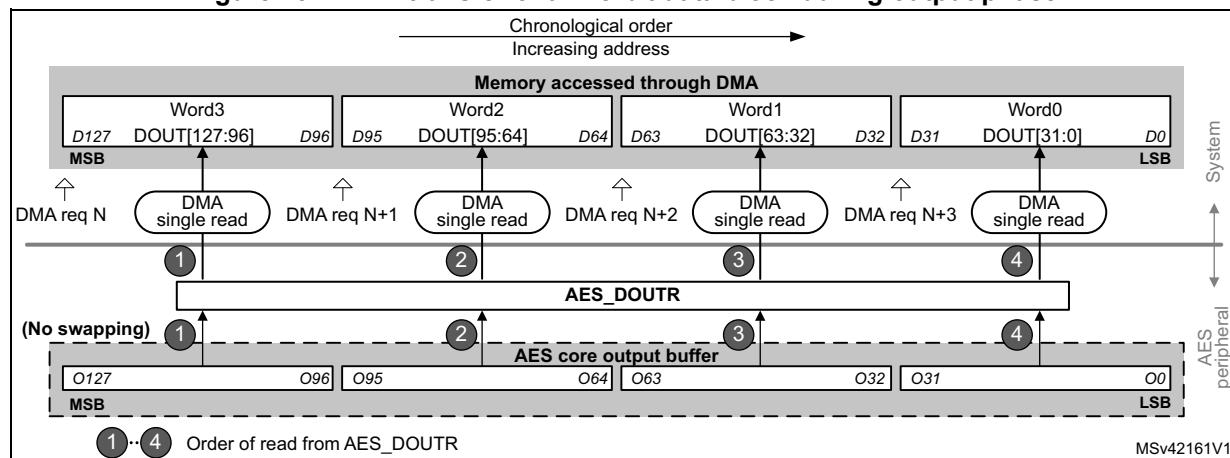
Setting the DMAOUTEN bit of the AES_CR register enables DMA reading from AES. The AES peripheral then initiates a DMA request during the Output phase each time it requires a word to be read from the AES_DOUTR register. It asserts four DMA requests to transfer one 128-bit (four-word) output data block to memory, as shown in [Figure 132](#).

See [Table 107](#) for recommended DMA configuration.

Table 107. DMA channel configuration for AES-to-memory data transfer

DMA channel control register field	Recommended configuration
Transfer size	It is the message length multiple of AES block size (4 words). According to the case extra bytes will have to be discarded.
Source burst size (peripheral)	Single
Destination burst size (memory)	Single
DMA FIFO size	AES FIFO_size = 4 bytes
Source transfer width (peripheral)	32-bit words
Destination transfer width (memory)	32-bit words
Source address increment (peripheral)	Fixed address of AES_DINR (no increment)
Destination address increment (memory)	Yes, after each 32-bit transfer

Figure 132. DMA transfer of a 128-bit data block during output phase



DMA operation in different operating modes

DMA operations are usable when Mode 1 (encryption) or Mode 3 (decryption) are selected via the MODE[1:0] bitfield of the register AES_CR. As in Mode 2 (key derivation) the AES_KEYRx registers must be written by software, enabling the DMA transfer through the DMAINEN and DMAOUTEN bits of the AES_CR register have no effect in that mode.

DMA single requests are generated by AES until it is disabled. So, after the data output phase at the end of processing of a 128-bit data block, AES switches automatically to a new data input phase for the next data block, if any.

When the data transferring between AES and memory is managed by DMA, the CCF flag is not relevant and can be ignored (left set) by software. It must only be cleared when transitioning back to data transferring managed by software. See [Suspend/resume operations](#)

in ECB/CBC modes in Section 17.4.8: AES basic chaining modes (ECB, CBC) as example.

17.4.17 AES error management

The read error flag (RDERR) and write error flag (WRERR) of the AES_SR register are set when an unexpected read or write operation, respectively, is detected. An interrupt can be generated if the error interrupt enable (ERRIE) bit of the AES_CR register is set. For more details, refer to [Section 17.5: AES interrupts](#).

Note: *AES is not disabled after an error detection and continues processing.*

AES can be re-initialized at any moment by clearing then setting the EN bit of the AES_CR register.

Read error flag (RDERR)

When an unexpected read operation is detected during the computation phase or during the input phase, the AES read error flag (RDERR) is set in the AES_SR register. An interrupt is generated if the ERRIE bit of the AES_CR register is set.

The RDERR flag is cleared by setting the corresponding ERRC bit of the AES_CR register.

Write error flag (WDERR)

When an unexpected write operation is detected during the computation phase or during the output phase, the AES write error flag (WRERR) is set in the AES_SR register. An interrupt is generated if the ERRIE bit of the AES_CR register is set.

The WDERR flag is cleared by setting the corresponding ERRC bit of the AES_CR register.

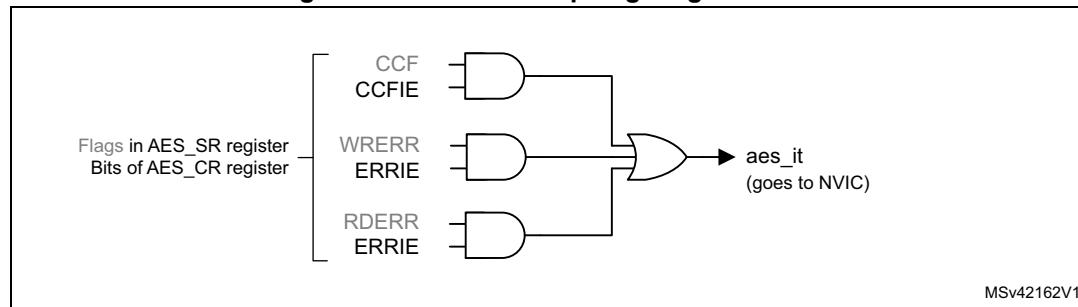
17.5 AES interrupts

There are three individual maskable interrupt sources generated by the AES peripheral, to signal the following events:

- computation completed
- read error, see [Section 17.4.17](#)
- write error, see [Section 17.4.17](#)

These three sources are combined into a common interrupt signal aes_it that connects to NVIC (nested vectored interrupt controller).

Figure 133. AES interrupt signal generation



Each AES interrupt source can individually be enabled/disabled, by setting/clearing the corresponding enable bit of the AES_CR register. See [Figure 133](#).

The status of the individual maskable interrupt sources can be read from the AES_SR register.

Table 108 gives a summary of the interrupt sources, their event flags and enable bits.

Table 108. AES interrupt requests

AES interrupt event	Event flag	Enable bit
computation completed flag	CCF	CCFIE
read error flag	RDERR	ERRIE
write error flag	WRERR	ERRIE

17.6 AES processing latency

The tables below summarize the latency to process a 128-bit block for each mode of operation.

Table 109. Processing latency (in clock cycle) for ECB, CBC and CTR

Key size	Mode of operation	Algorithm	Input phase	Computation phase	Output phase	Total
128-bit	Mode 1: Encryption	ECB, CBC, CTR	8	202	4	214
	Mode 2: Key derivation	-	-	80	-	80
	Mode 3: Decryption	ECB, CBC, CTR	8	202	4	214
	Mode 4: Key derivation then decryption	ECB, CBC	8	276	4	288
256-bit	Mode 1: Encryption	ECB, CBC, CTR	8	286	4	298
	Mode 2: Key derivation	-	-	109	-	109
	Mode 3: Decryption	ECB, CBC, CTR	8	286	4	298
	Mode 4: Key derivation then decryption	ECB, CBC	8	380	4	392

Table 110. Processing latency for GCM and CCM (in clock cycle)

Key size	Mode of operation	Algorithm	Init Phase	Header phase	Payload phase	Tag phase
128-bit	Mode 1: Encryption/ Mode 3: Decryption	GCM	215	67	202	202
	-	CCM authentication	-	206	-	202
256-bit	Mode 1: Encryption/ Mode 3: Decryption	GCM	299	67	286	286
	-	CCM authentication	-	290	-	286

Note: *Data insertion can include wait states forced by AES on the AHB bus (maximum 3 cycles, typical 1 cycle). This applies to all header/payload/tag phases.*

17.7 AES registers

17.7.1 AES control register (AES_CR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	KEYSIZE	Res.	CHMOD[2]
													rw		rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	GCMPH[1:0]	DMAOUTEN	DMAINEN	ERRIE	CCFIE	ERRC	CCFC	CHMOD[1:0]	MODE[1:0]	DATATYPE[1:0]	EN				
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:19 Reserved, must be kept at zero

Bit 18 **KEYSIZE**: Key size selection

This bitfield defines the length of the key used in the AES cryptographic core, in bits:

0: 128

1: 256

The bit value change is allowed only when AES is disabled, so as to avoid an unpredictable behavior.

Bit 17 Reserved, must be kept at zero

Bit 16 **CHMOD[2]**: Chaining mode selection, bit [2]

Refer to the bits [5:6] of the register for the description of the CHMOD[2:0] bitfield

Bit 15 Reserved, must be kept at zero

Bits 14:13 **GCMPH[1:0]**: GCM or CCM phase selection

This bitfield selects the phase of GCM, GMAC or CCM algorithm:

00: Init phase

01: Header phase

10: Payload phase

11: Final phase

The bitfield has no effect if other than GCM, GMAC or CCM algorithms are selected (through the ALGOMODE bitfield).

Bit 12 DMAOUTEN: DMA output enable

This bit enables/disables data transferring with DMA, in the output phase:

- 0: Disable
- 1: Enable

When the bit is set, DMA requests are automatically generated by AES during the output data phase. This feature is only effective when Mode 1 or Mode 3 is selected through the MODE[1:0] bitfield. It is not effective for Mode 2 (key derivation).

Usage of DMA with Mode 4 (single decryption) is not recommended.

Bit 11 DMAINEN: DMA input enable

This bit enables/disables data transferring with DMA, in the input phase:

- 0: Disable
- 1: Enable

When the bit is set, DMA requests are automatically generated by AES during the input data phase. This feature is only effective when Mode 1 or Mode 3 is selected through the MODE[1:0] bitfield. It is not effective for Mode 2 (key derivation).

Usage of DMA with Mode 4 (single decryption) is not recommended.

Bit 10 ERRIE: Error interrupt enable

This bit enables or disables (masks) the AES interrupt generation when RDERR and/or WRERR is set:

- 0: Disable (mask)
- 1: Enable

Bit 9 CCFIE: CCF interrupt enable

This bit enables or disables (masks) the AES interrupt generation when CCF (computation complete flag) is set:

- 0: Disable (mask)
- 1: Enable

Bit 8 ERRC: Error flag clear

Upon written to 1, this bit clears the RDERR and WRERR error flags in the AES_SR register:

- 0: No effect
- 1: Clear RDERR and WRERR flags

Reading the flag always returns zero.

Bit 7 CCFC: Computation complete flag clear

Upon written to 1, this bit clears the computation complete flag (CCF) in the AES_SR register:

- 0: No effect
- 1: Clear CCF

Reading the flag always returns zero.

Bits 6:5 **CHMOD[1:0]**: Chaining mode selection, bits [1:0]

These bits, together with the bit CHMOD[2] (see bit 16 of this register), form CHMOD[2:0] bitfield that selects the AES chaining mode:

- 000: Electronic codebook (ECB)
- 001: Cipher-Block Chaining (CBC)
- 010: Counter Mode (CTR)
- 011: Galois Counter Mode (GCM) and Galois Message Authentication Code (GMAC)
- 100: Counter with CBC-MAC (CCM)
- >100: Reserved

The bitfield value change is allowed only when AES is disabled, so as to avoid an unpredictable behavior.

Bits 4:3 **MODE[1:0]**: AES operating mode

This bitfield selects the AES operating mode:

- 00: Mode 1: encryption
- 01: Mode 2: key derivation (or key preparation for ECB/CBC decryption)
- 10: Mode 3: decryption
- 11: Mode 4: key derivation then single decryption

The bitfield value change is allowed only when AES is disabled, so as to avoid an unpredictable behavior. Any attempt to selecting Mode 4 while either ECB or CBC chaining mode is not selected, defaults to effective selection of Mode 3. It is not possible to select a Mode 3 following a Mode 4.

Bits 2:1 **DATATYPE[1:0]**: Data type selection

This bitfield defines the format of data written in the AES_DINR register or read from the AES_DOUTR register, through selecting the mode of data swapping:

- 00: None
- 01: Half-word (16-bit)
- 10: Byte (8-bit)
- 11: Bit

For more details, refer to [Section 17.4.13: .AES data registers and data swapping](#).

The bitfield value change is allowed only when AES is disabled, so as to avoid an unpredictable behavior.

Bit 0 **EN**: AES enable

This bit enables/disables the AES peripheral:

- 0: Disable
- 1: Enable

At any moment, clearing then setting the bit re-initializes the AES peripheral.

This bit is automatically cleared by hardware when the key preparation process ends (Mode 2).

17.7.2 AES status register (AES_SR)

Address offset: 0x04

Reset value: 0x0000 0000

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

Bits 31:4 Reserved, must be kept at zero

Bit 3 **BUSY**: Busy

This flag indicates whether AES is idle or busy during GCM payload **encryption** phase:

0: Idle

1: Busy

The flag is controlled by hardware. When the flag indicates “idle”, the current message processing may be suspended to process a higher-priority message.

This flag is effective only in GCM payload encryption phase. In other chaining modes, or in GCM phases other than payload encryption, the flag must be ignored.

Bit 2 WRERR: Write error

This flag indicates the detection of an unexpected write operation to the AES_DINR register (during computation or data output phase):

- 0: Not detected
- 1: Detected

The flag is set by hardware. It is cleared by software upon setting the ERRC bit of the AES_CR register.

Upon the flag setting, an interrupt is generated if enabled through the ERRIE bit of the AES_CR register.

The flag setting has no impact on the AES operation.

The flag is not effective when key derivation mode, or GCM/CCM Init phase is selected.

Bit 1 RDERR: Read error flag

This flag indicates the detection of an unexpected read operation from the AES_DOUTR register (during computation or data input phase):

- 0: Not detected
- 1: Detected

The flag is set by hardware. It is cleared by software upon setting the ERRC bit of the AES_CR register.

Upon the flag setting, an interrupt is generated if enabled through the ERRIE bit of the AES_CR register.

The flag setting has no impact on the AES operation.

The flag is not effective when key derivation mode, nor GCM/CCM init/header phase is selected.

Bit 0 CCF: Computation completed flag

This flag indicates whether the computation is completed:

- 0: Not completed
- 1: Completed

The flag is set by hardware upon the completion of the computation. It is cleared by software, upon setting the CCFC bit of the AES_CR register.

Upon the flag setting, an interrupt is generated if enabled through the CCFIE bit of the AES_CR register.

The flag is significant only when the DMAOUTEN bit is 0. It may stay high when DMA_EN is 1.

17.7.3 AES data input register (AES_DINR)

Address offset: 0x08

Reset value: 0x0000 0000

Only 32-bit access type is supported.

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

Bits 31:0 **DIN[x+31:x]**: One of four 32-bit words of a 128-bit input data block being written into the peripheral
 This bitfield feeds a 32-bit input buffer. A 4-fold sequential write to this bitfield during the input phase virtually writes a complete 128-bit block of input data to the AES peripheral. Upon each write, the data from the input buffer are handled by the data swap block according to the DATATYPE[1:0] bitfield, then written into the AES core 128-bit input buffer.

The substitution for “x”, from the first to the fourth write operation, is: 96, 64, 32, and 0. In other words, data from the first to the fourth write operation are: DIN[127:96], DIN[95:64], DIN[63:32], and DIN[31:0].

The data signification of the input data block depends on the AES operating mode:

- **Mode 1** (encryption): plaintext
- **Mode 2** (key derivation): the bitfield is not used (AES_KEYRx registers used for input)
- **Mode 3** (decryption) and **Mode 4** (key derivation then single decryption): ciphertext

The data swap operation is described in [Section 17.4.13: .AES data registers and data swapping on page 490](#).

17.7.4 AES data output register (AES_DOUTR)

Address offset: 0x0C

Reset value: 0x0000 0000

Only 32-bit access type is supported.

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

Bits 31:0 **DOUT[x+31:x]**: One of four 32-bit words of a 128-bit output data block being read from the peripheral

This bitfield fetches a 32-bit output buffer. A 4-fold sequential read of this bitfield, upon the computation completion (CCF set), virtually reads a complete 128-bit block of output data from the AES peripheral. Before reaching the output buffer, the data produced by the AES core are handled by the data swap block according to the DATATYPE[1:0] bitfield.

The substitution for DOUT[x+31:x], from the first to the fourth read operation, is: 96, 64, 32, and 0. In other words, data from the first to the fourth read operation are: DOUT[127:96], DOUT[95:64], DOUT[63:32], and DOUT[31:0].

The data signification of the output data block depends on the AES operating mode:

- **Mode 1** (encryption): ciphertext
- **Mode 2** (key derivation): the bitfield is not used (AES_KEYRx registers used for output).
- **Mode 3** (decryption) and **Mode 4** (key derivation then single decryption): plaintext

The data swap operation is described in [Section 17.4.13: .AES data registers and data swapping on page 490](#).

17.7.5 AES key register 0 (AES_KEYR0)

Address offset: 0x10

Reset value: 0x0000 0000

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

Bits 31:0 **KEY[31:0]**: Cryptographic key, bits [31:0]

This bitfield contains the bits [31:0] of the AES encryption or decryption key, depending on the operating mode:

- In **Mode 1** (encryption), **Mode 2** (key derivation) and **Mode 4** (key derivation then single decryption): the value to write into the bitfield is the encryption key.
- In **Mode 3** (decryption): the value to write into the bitfield is the encryption key to be derived before being used for decryption. After writing the encryption key into the bitfield, its reading before enabling AES returns the same value. Its reading after enabling AES and after the CCF flag is set returns the decryption key derived from the encryption key.

Note: In mode 4 (key derivation then decryption) the bitfield always contains the encryption key.

The AES_KEYRx registers may be written only when the AES peripheral is disabled.

Refer to [Section 17.4.14: AES key registers on page 492](#) for more details.

17.7.6 AES key register 1 (AES_KEYR1)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
KEY[63:48]															
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
KEY[47:32]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **KEY[63:32]**: Cryptographic key, bits [63:32]

Refer to the AES_KEYR0 register for description of the KEY[255:0] bitfield.

17.7.7 AES key register 2 (AES_KEYR2)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
KEY[95:80]															
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
KEY[79:64]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **KEY[95:64]**: Cryptographic key, bits [95:64]

Refer to the AES_KEYR0 register for description of the KEY[255:0] bitfield.

17.7.8 AES key register 3 (AES_KEYR3)

Address offset: 0x1C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
KEY[127:112]															
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
KEY[111:96]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **KEY[127:96]**: Cryptographic key, bits [127:96]

Refer to the AES_KEYR0 register for description of the KEY[255:0] bitfield.

17.7.9 AES initialization vector register 0 (AES_IVR0)

Address offset: 0x20

Reset value: 0x0000 0000

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

Bits 31:0 **IVI[31:0]**: Initialization vector input, bits [31:0]

Refer to [Section 17.4.15: AES initialization vector registers on page 492](#) for description of the IVI[127:0] bitfield.

The initialization vector is only used in chaining modes other than ECB.

The initialization vector may be written only when the AES peripheral is disabled.

17.7.10 AES initialization vector register 1 (AES_IVR1)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IVI[63:48]															
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
IVI[47:32]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **IVI[63:32]**: Initialization vector input, bits [63:32]

Refer to [Section 17.4.15: AES initialization vector registers on page 492](#) for description of the IVI[127:0] bitfield.

The initialization vector is only used in chaining modes other than ECB.

The initialization vector may be written only when the AES peripheral is disabled.

17.7.11 AES initialization vector register 2 (AES_IVR2)

Address offset: 0x28

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IVI[95:80]															
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
IVI[79:64]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **IVI[95:64]**: Initialization vector input, bits [95:64]

Refer to [Section 17.4.15: AES initialization vector registers on page 492](#) for description of the IVI[127:0] bitfield.

The initialization vector is only used in chaining modes other than ECB.

The initialization vector may be written only when the AES peripheral is disabled.

17.7.12 AES initialization vector register 3 (AES_IVR3)

Address offset: 0x2C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IVI[127:112]															
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
IVI[111:96]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **IVI[127:96]**: Initialization vector input, bits [127:96]

Refer to [Section 17.4.15: AES initialization vector registers on page 492](#) for description of the IVI[127:0] bitfield.

The initialization vector is only used in chaining modes other than ECB.

The initialization vector may be written only when the AES peripheral is disabled.

17.7.13 AES key register 4 (AES_KEYR4)

Address offset: 0x30

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
KEY[159:144]															
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
KEY[143:128]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **KEY[159:128]**: Cryptographic key, bits [159:128]

Refer to the AES_KEYR0 register for description of the KEY[255:0] bitfield.

17.7.14 AES key register 5 (AES_KEYR5)

Address offset: 0x34

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
KEY[191:176]															
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
KEY[175:160]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **KEY[191:160]**: Cryptographic key, bits [191:160]

Refer to the AES_KEYR0 register for description of the KEY[255:0] bitfield.

17.7.15 AES key register 6 (AES_KEYR6)

Address offset: 0x38

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
KEY[223:208]															
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
KEY[207:192]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **KEY[223:192]**: Cryptographic key, bits [223:192]

Refer to the AES_KEYR0 register for description of the KEY[255:0] bitfield.

17.7.16 AES key register 7 (AES_KEYR7)

Address offset: 0x3C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
KEY[255:240]															
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
KEY[239:224]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **KEY[255:224]**: Cryptographic key, bits [255:224]

Refer to the AES_KEYR0 register for description of the KEY[255:0] bitfield.

Note: *The key registers from 4 to 7 are used only when the key length of 256 bits is selected. They have no effect when the key length of 128 bits is selected (only key registers 0 to 3 are used in that case).*

17.7.17 AES suspend registers (AES_SUSPxR)

Address offset: 0x040 + x * 0x4, (x = 0 to 7)

Reset value: 0x0000 0000

These registers contain the complete internal register states of the AES processor when the AES processing of the current task is suspended to process a higher-priority task.

Upon suspend, the software reads and saves the AES_SUSPxR register contents (where x is from 0 to 7) into memory, before using the AES processor for the higher-priority task.

Upon completion, the software restores the saved contents back into the corresponding suspend registers, before resuming the original task.

Note: *These registers are used only when GCM, GMAC, or CCM chaining mode is selected.*

These registers can be read only when AES is enabled. Reading these registers while AES is disabled returns 0x0000 0000.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SUSPx															
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
SUSPx															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **SUSPx**: AES suspend

Upon suspend operation, this bitfield of every AES_SUSPxR register takes the value of one of internal AES registers.

17.7.18 AES register map

Table 111. AES 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		
0x0000	AES_CR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	KEYSIZE	0	CHMOD[2]	Res.	GCMPH[1:0]	Res.	DMAOUTEN	Res.	ERRIE	Res.	CCFIE	Res.	MODE[1:0]	Res.	WIBERR	0	DATA[TYPE][1:0]				
	Reset value														0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x0004	AES_SR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
	Reset value																																		
0x0008	AES_DINR x=96,64,32,0	DIN[x+31:x]																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x000C	AES_DOUTR x=96,64,32,0	DOUT[x+31:x]																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x0010	AES_KEYR0	KEY[31:0]																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x0014	AES_KEYR1	KEY[63:32]																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x0018	AES_KEYR2	KEY[95:64]																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x001C	AES_KEYR3	KEY[127:96]																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x0020	AES_IVR0	IVI[31:0]																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x0024	AES_IVR1	IVI[63:32]																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x0028	AES_IVR2	IVI[95:64]																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x002C	AES_IVR3	IVI[127:96]																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x0030	AES_KEYR4	KEY[159:128]																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x0034	AES_KEYR5	KEY[191:160]																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x0038	AES_KEYR6	KEY[223:192]																																	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

Table 111. AES 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
0x003 C	AES_KEYR7	KEY[255:224]																															
	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		
0x0040	AES_SUSP0R	SUSP0[31:0]																															
	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		
0x0044	AES_SUSP1R	SUSP1[31:0]																															
	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		
0x0048	AES_SUSP2R	SUSP2[31:0]																															
	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		
0x004 C	AES_SUSP3R	SUSP3[31:0]																															
	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		
0x0050	AES_SUSP4R	SUSP4[31:0]																															
	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		
0x0054	AES_SUSP5R	SUSP5[31:0]																															
	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		
0x0058	AES_SUSP6R	SUSP6[31:0]																															
	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		
0x005 C	AES_SUSP7R	SUSP7[31:0]																															
	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		

18 Advanced-control timers (TIM1/TIM8)

18.1 TIM1/TIM8 introduction

The advanced-control timers (TIM1/TIM8) consist of a 16-bit auto-reload counter driven by a programmable prescaler.

It may be used for a variety of purposes, including measuring the pulse lengths of input signals (input capture) or generating output waveforms (output compare, PWM, complementary PWM with dead-time insertion).

Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the RCC clock controller prescalers.

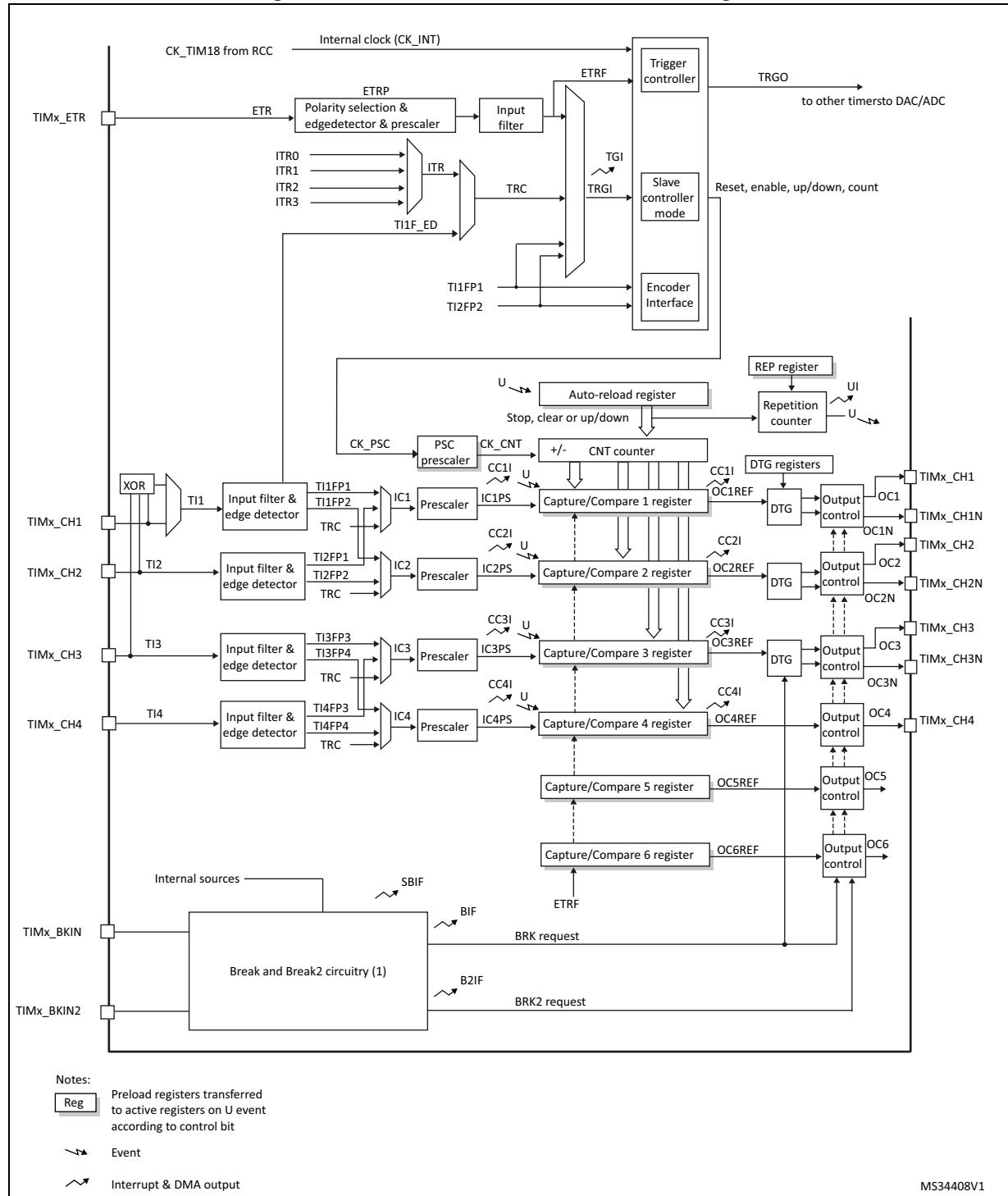
The advanced-control (TIM1/TIM8) and general-purpose (TIMy) timers are completely independent, and do not share any resources. They can be synchronized together as described in [Section 18.3.25: Timer synchronization](#).

18.2 TIM1/TIM8 main features

TIM1/TIM8 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 65536.
- Up to 6 independent channels for:
 - Input Capture (but channels 5 and 6)
 - 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.
- 2 break inputs to put the timer’s output signals in a safe user selectable configuration.
- Interrupt/DMA generation on the following events:
 - Update: counter overflow/underflow, counter initialization (by software or internal/external trigger)
 - Trigger event (counter start, stop, initialization or count by internal/external trigger)
 - Input capture
 - Output compare
- Supports incremental (quadrature) encoder and Hall-sensor circuitry for positioning purposes
- Trigger input for external clock or cycle-by-cycle current management

Figure 134. Advanced-control timer block diagram



1. See [Figure 176: Break and Break2 circuitry overview](#) for details

18.3 TIM1/TIM8 functional description

18.3.1 Time-base unit

The main block of the programmable advanced-control timer is a 16-bit counter with its related auto-reload register. The counter can count up, down or both up and down. The counter clock can be divided by a prescaler.

The counter, the auto-reload register and the prescaler register can be written or read by software. This is true even when the counter is running.

The time-base unit includes:

- Counter register (TIMx_CNT)
- Prescaler register (TIMx_PSC)
- Auto-reload register (TIMx_ARR)
- Repetition counter register (TIMx_RCR)

The auto-reload register is preloaded. Writing to or reading from the auto-reload register accesses the preload register. The content of the preload register are transferred into the shadow register permanently or at each update event (UEV), depending on the auto-reload preload enable bit (ARPE) in TIMx_CR1 register. The update event is sent when the counter reaches the overflow (or underflow when downcounting) and if the UDIS bit equals 0 in the TIMx_CR1 register. It can also be generated by software. The generation of the update event is described in detailed for each configuration.

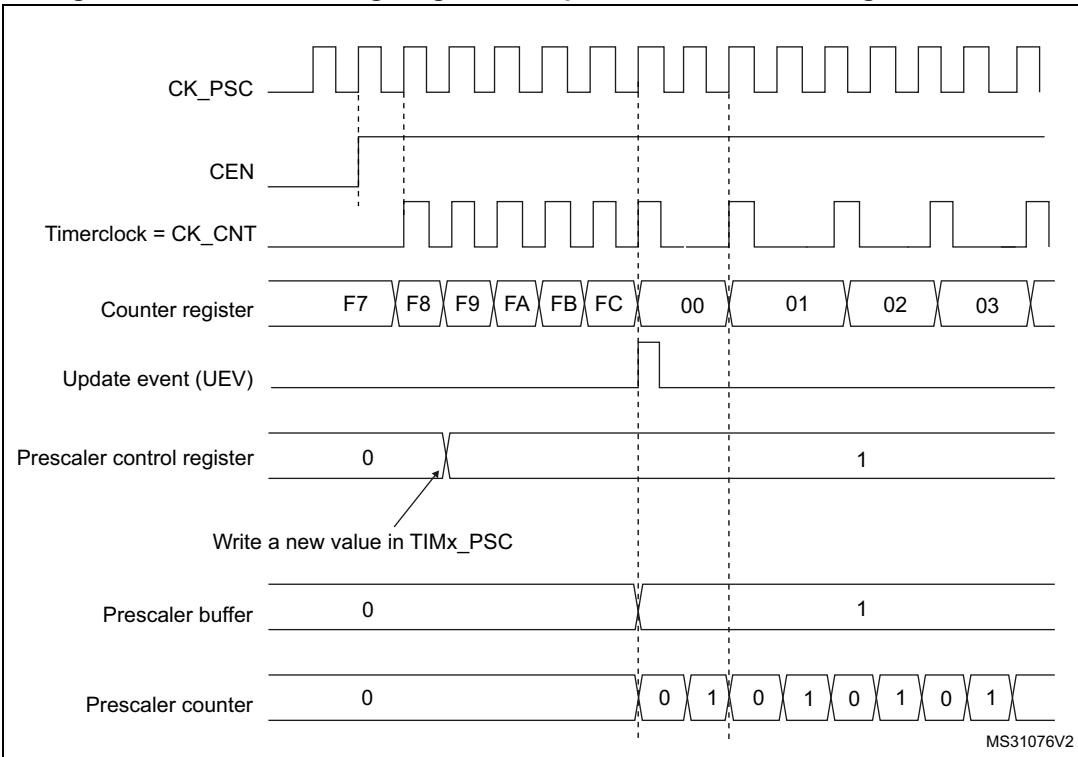
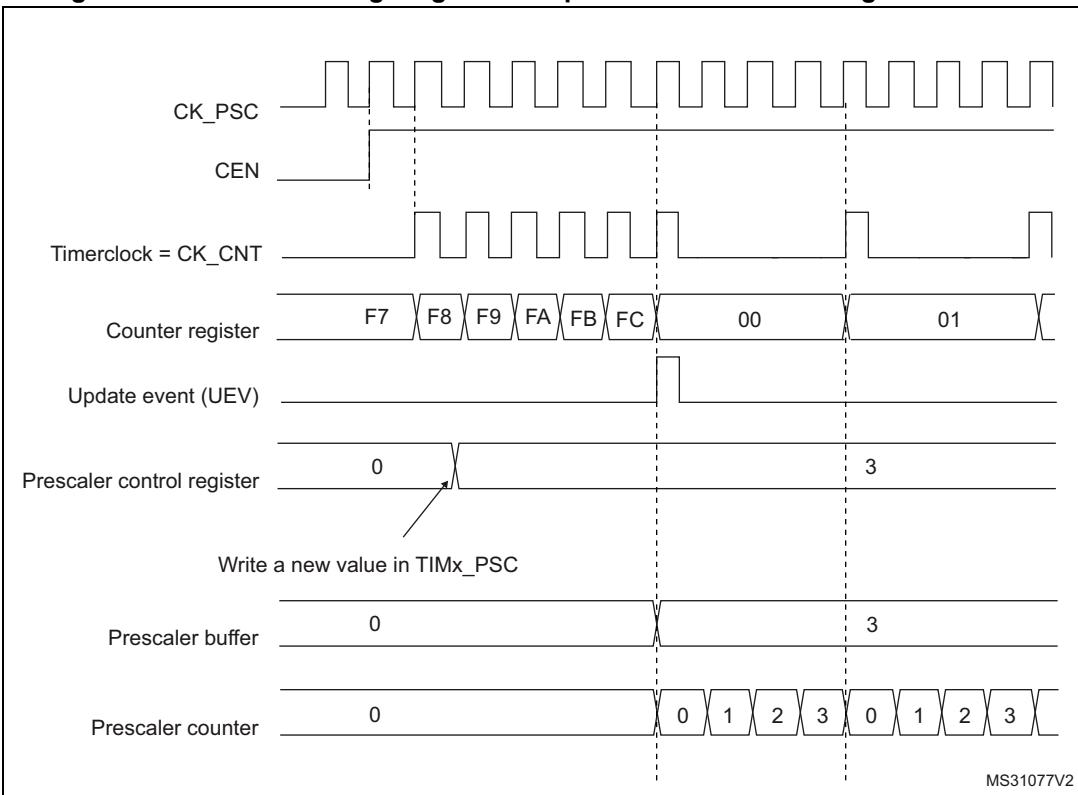
The counter is clocked by the prescaler output CK_CNT, which is enabled only when the counter enable bit (CEN) in TIMx_CR1 register is set (refer also to the slave mode controller description to get more details on counter enabling).

Note that the counter starts counting 1 clock cycle after setting the CEN bit in the TIMx_CR1 register.

Prescaler description

The prescaler can divide the counter clock frequency by any factor between 1 and 65536. It is based on a 16-bit counter controlled through a 16-bit register (in the TIMx_PSC register). It can be changed on the fly as this control register is buffered. The new prescaler ratio is taken into account at the next update event.

Figure 135 and *Figure 136* give some examples of the counter behavior when the prescaler ratio is changed on the fly:

Figure 135. Counter timing diagram with prescaler division change from 1 to 2**Figure 136. Counter timing diagram with prescaler division change from 1 to 4**

18.3.2 Counter modes

Upcounting mode

In upcounting mode, the counter counts from 0 to the auto-reload value (content of the TIMx_ARR register), then restarts from 0 and generates a counter overflow event.

If the repetition counter is used, the update event (UEV) is generated after upcounting is repeated for the number of times programmed in the repetition counter register (TIMx_RCR) + 1. Else the update event is generated at each counter overflow.

Setting the UG bit in the TIMx_EGR register (by software or by using the slave mode controller) also generates an update event.

The UEV event can be disabled by software by setting the UDIS bit in the TIMx_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until the UDIS bit has been written to 0. However, the counter restarts from 0, as well as the counter of the prescaler (but the prescale rate does not change). In addition, if the URS bit (update request selection) in TIMx_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

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

- The repetition counter is reloaded with the content of TIMx_RCR register,
- The auto-reload shadow register is updated with the preload value (TIMx_ARR),
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSC register).

The following figures show some examples of the counter behavior for different clock frequencies when TIMx_ARR=0x36.

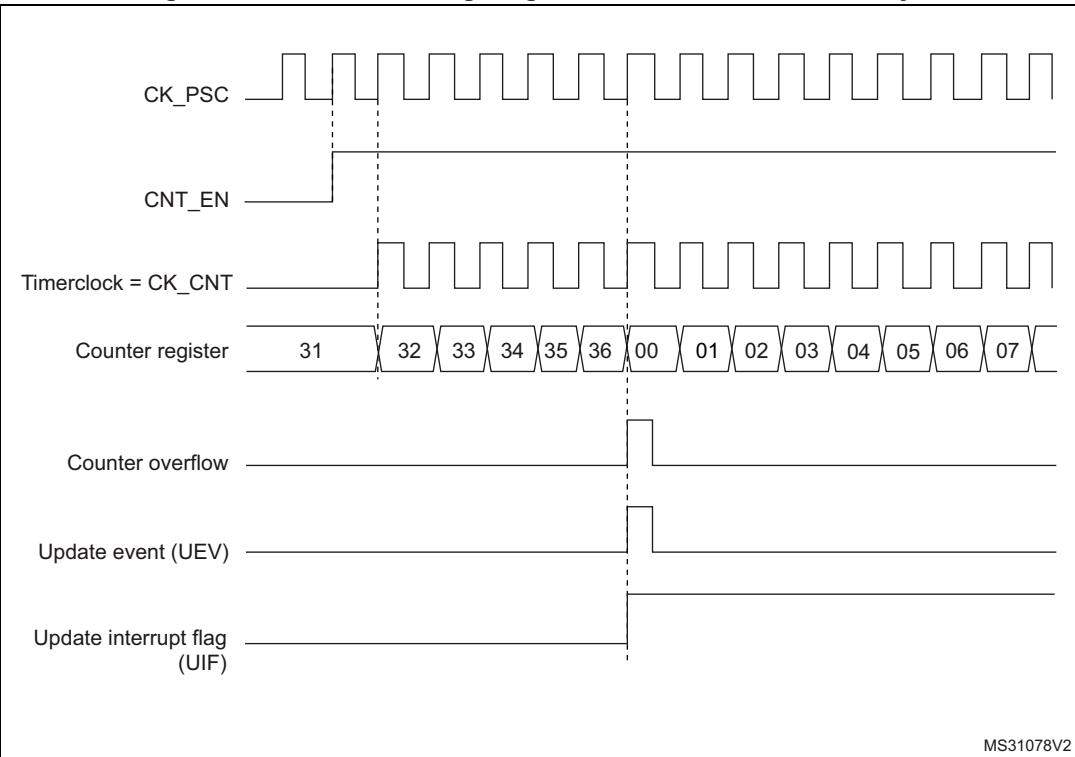
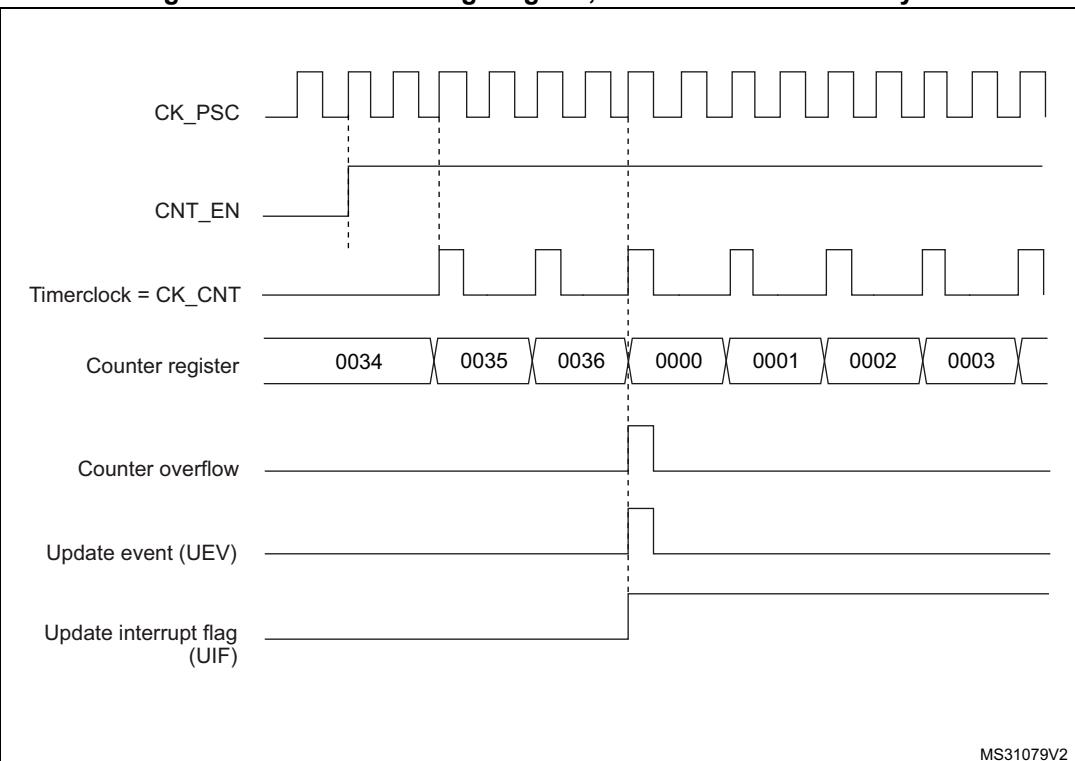
Figure 137. Counter timing diagram, internal clock divided by 1**Figure 138. Counter timing diagram, internal clock divided by 2**

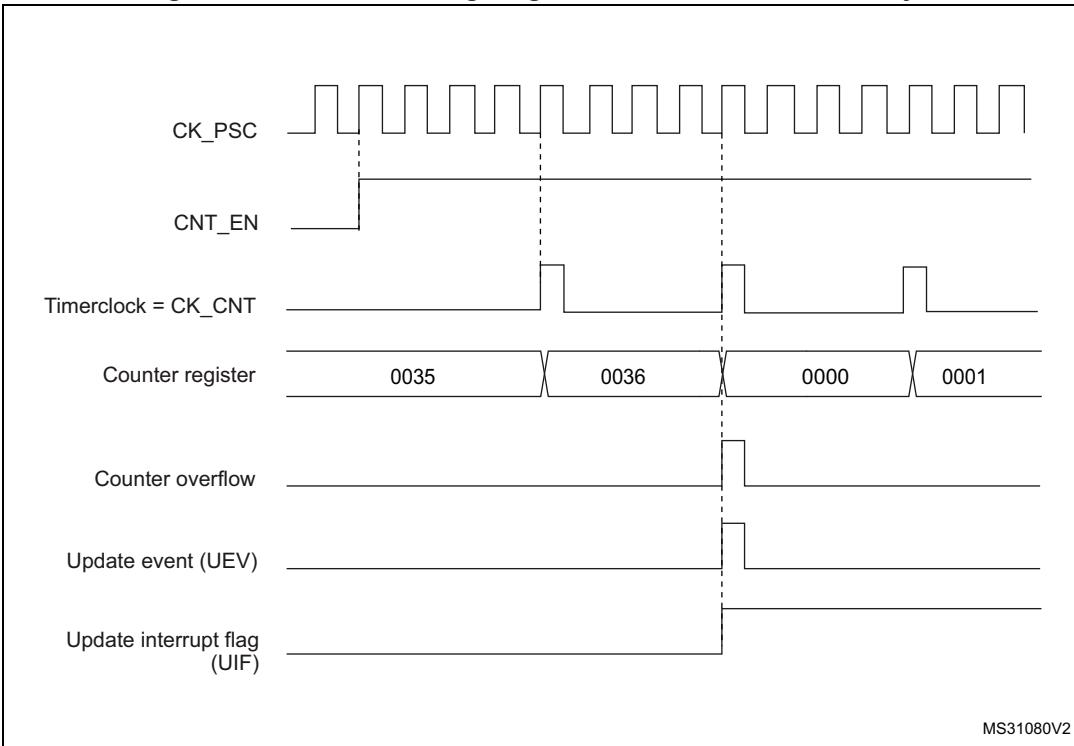
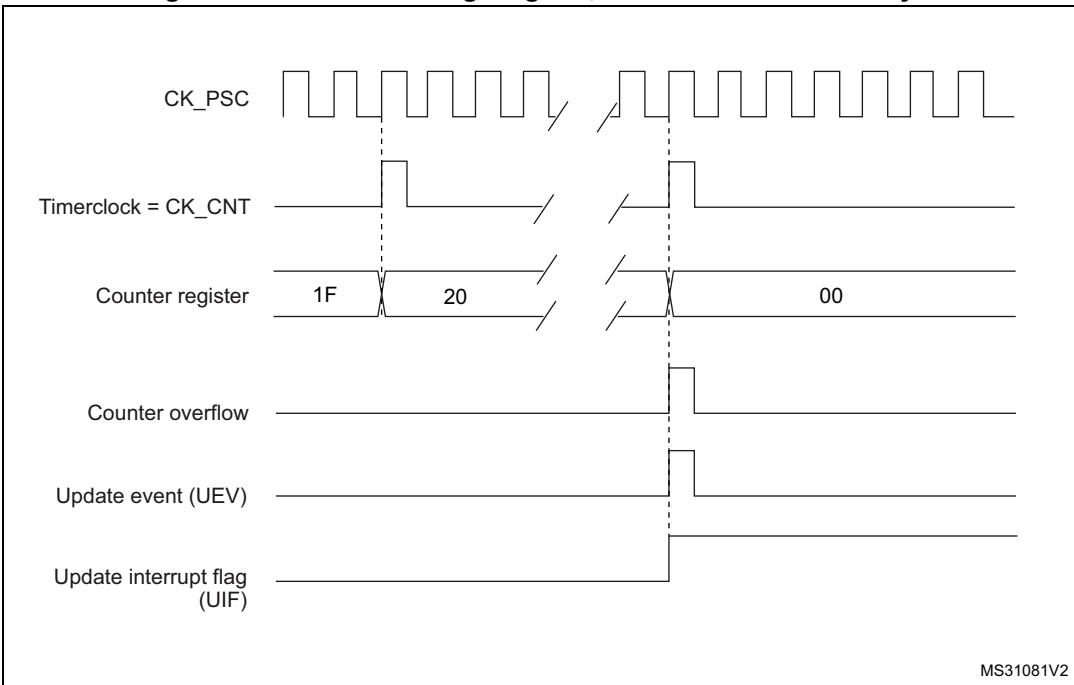
Figure 139. Counter timing diagram, internal clock divided by 4**Figure 140. Counter timing diagram, internal clock divided by N**

Figure 141. Counter timing diagram, update event when ARPE=0 (TIMx_ARR not preloaded)

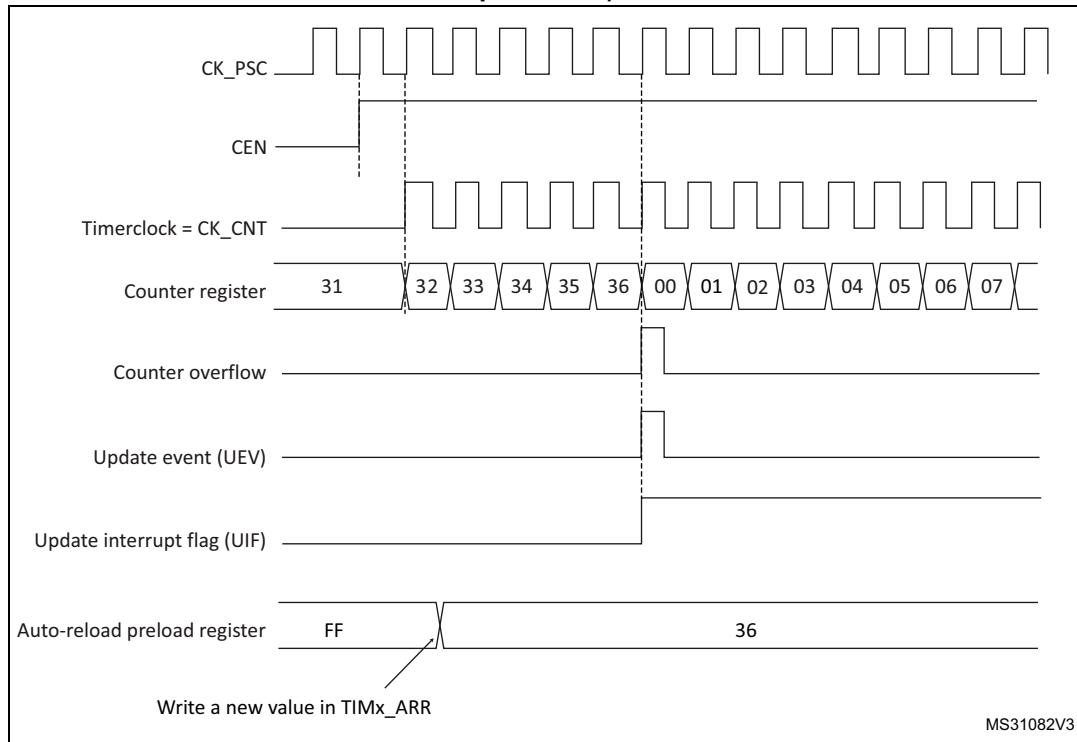
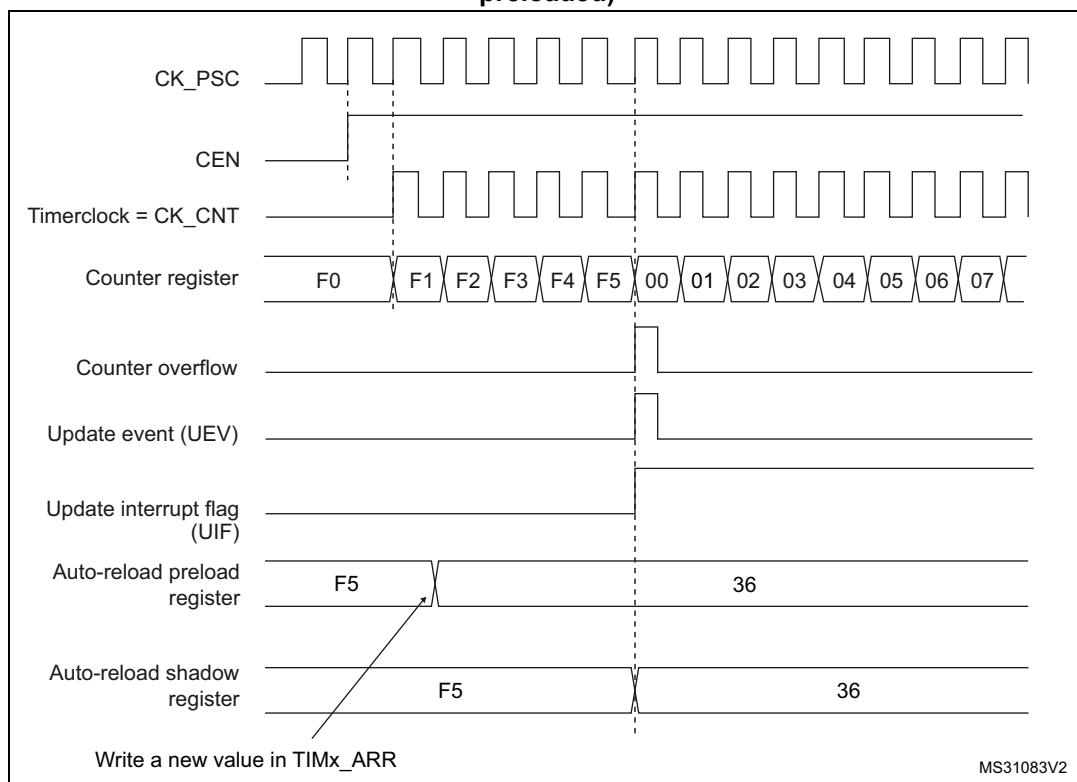


Figure 142. Counter timing diagram, update event when ARPE=1 (TIMx_ARR preloaded)



Downcounting mode

In downcounting mode, the counter counts from the auto-reload value (content of the TIMx_ARR register) down to 0, then restarts from the auto-reload value and generates a counter underflow event.

If the repetition counter is used, the update event (UEV) is generated after downcounting is repeated for the number of times programmed in the repetition counter register (TIMx_RCR) + 1. Else the update event is generated at each counter underflow.

Setting the UG bit in the TIMx_EGR register (by software or by using the slave mode controller) also generates an update event.

The UEV update event can be disabled by software by setting the UDIS bit in TIMx_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until UDIS bit has been written to 0. However, the counter restarts from the current auto-reload value, whereas the counter of the prescaler restarts from 0 (but the prescale rate doesn't change).

In addition, if the URS bit (update request selection) in TIMx_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

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

- The repetition counter is reloaded with the content of TIMx_RCR register.
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSC register).
- The auto-reload active register is updated with the preload value (content of the TIMx_ARR register). Note that the auto-reload is updated before the counter is reloaded, so that the next period is the expected one.

The following figures show some examples of the counter behavior for different clock frequencies when TIMx_ARR=0x36.

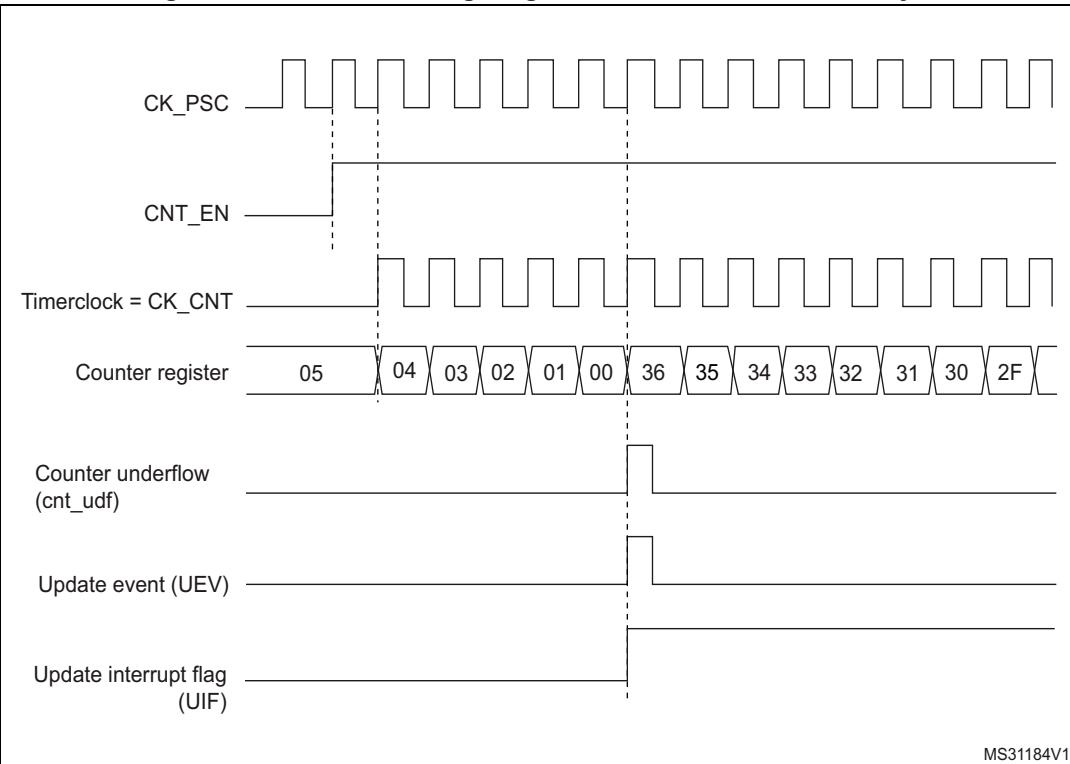
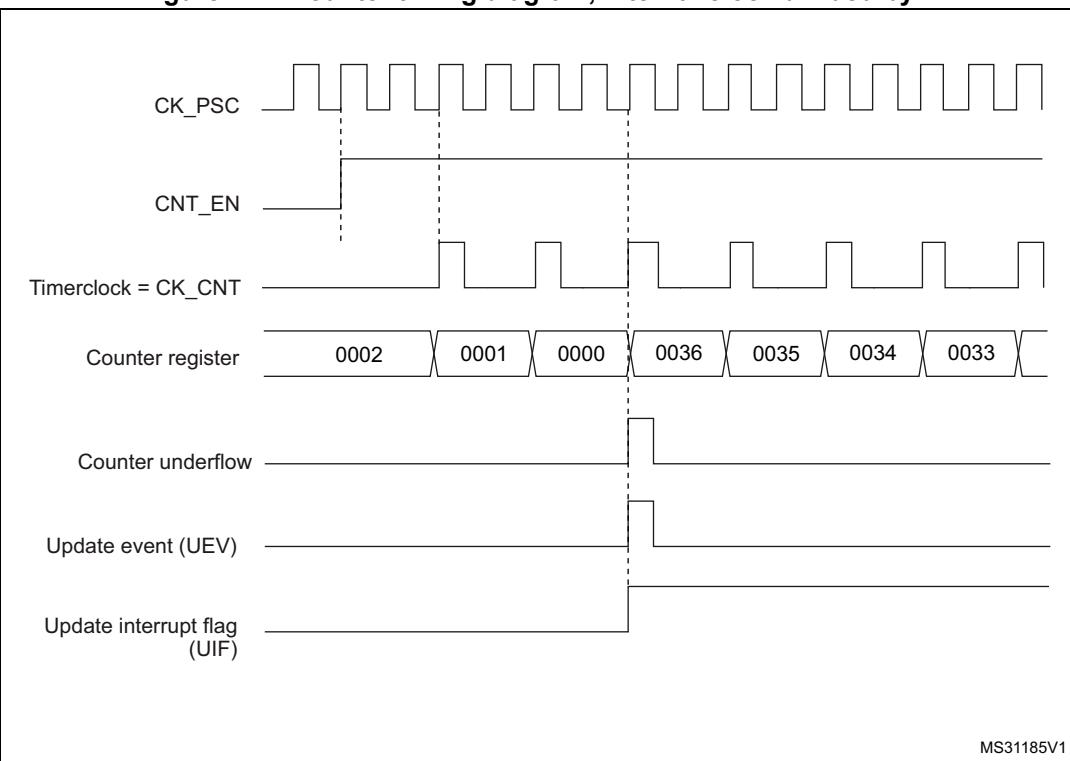
Figure 143. Counter timing diagram, internal clock divided by 1**Figure 144. Counter timing diagram, internal clock divided by 2**

Figure 145. Counter timing diagram, internal clock divided by 4

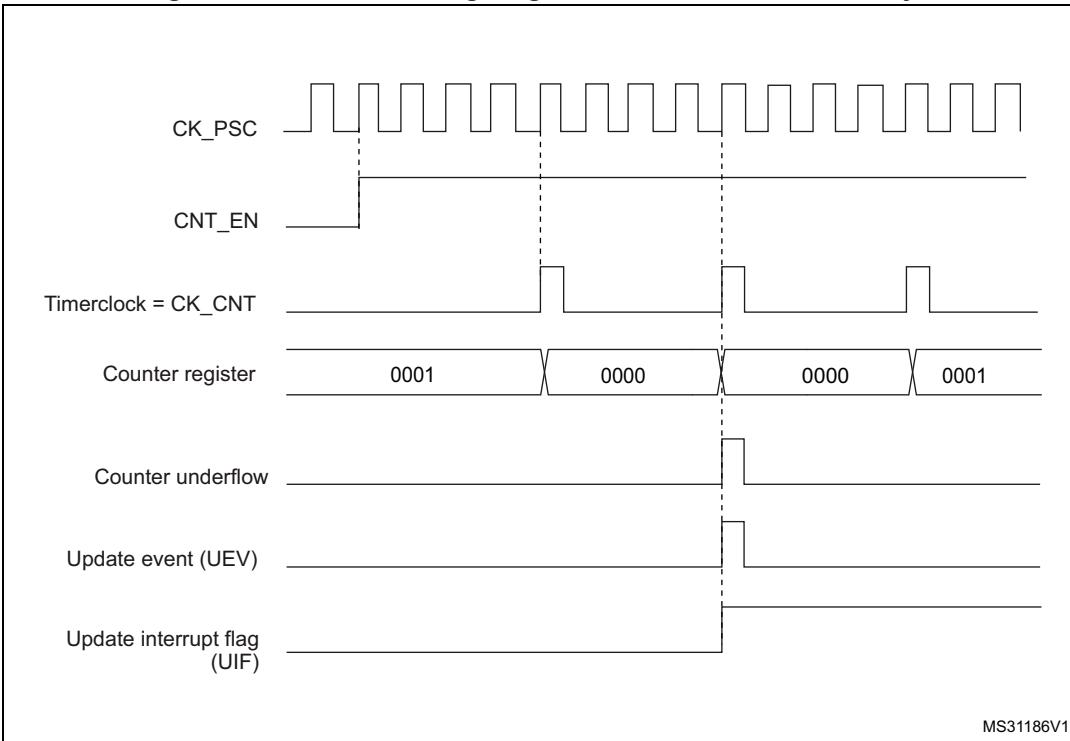


Figure 146. Counter timing diagram, internal clock divided by N

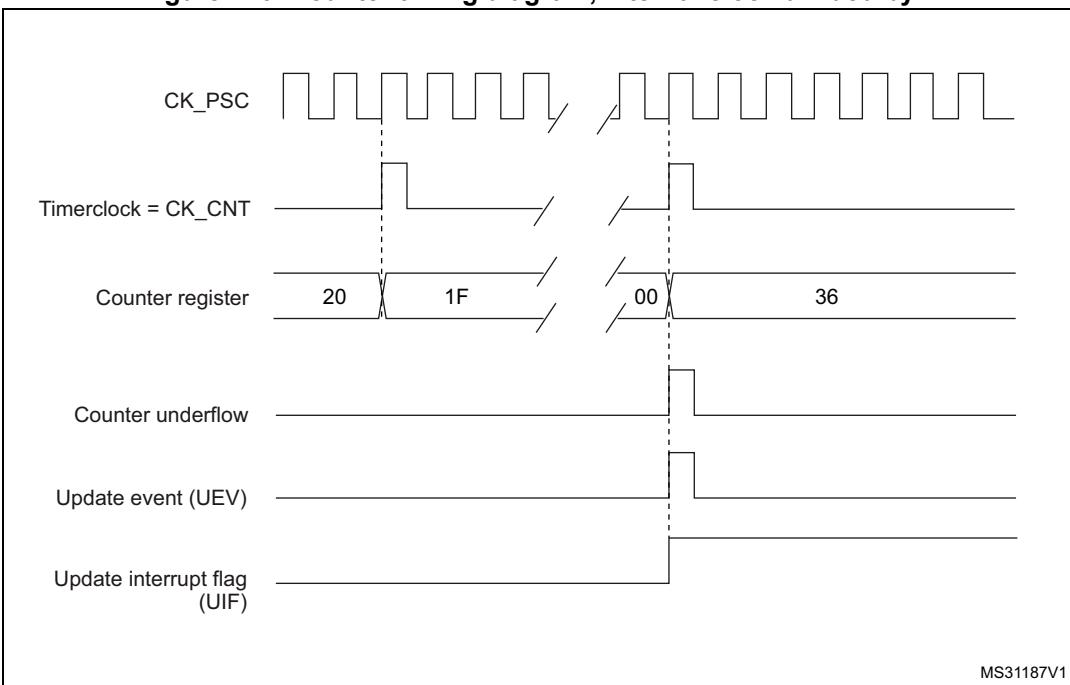
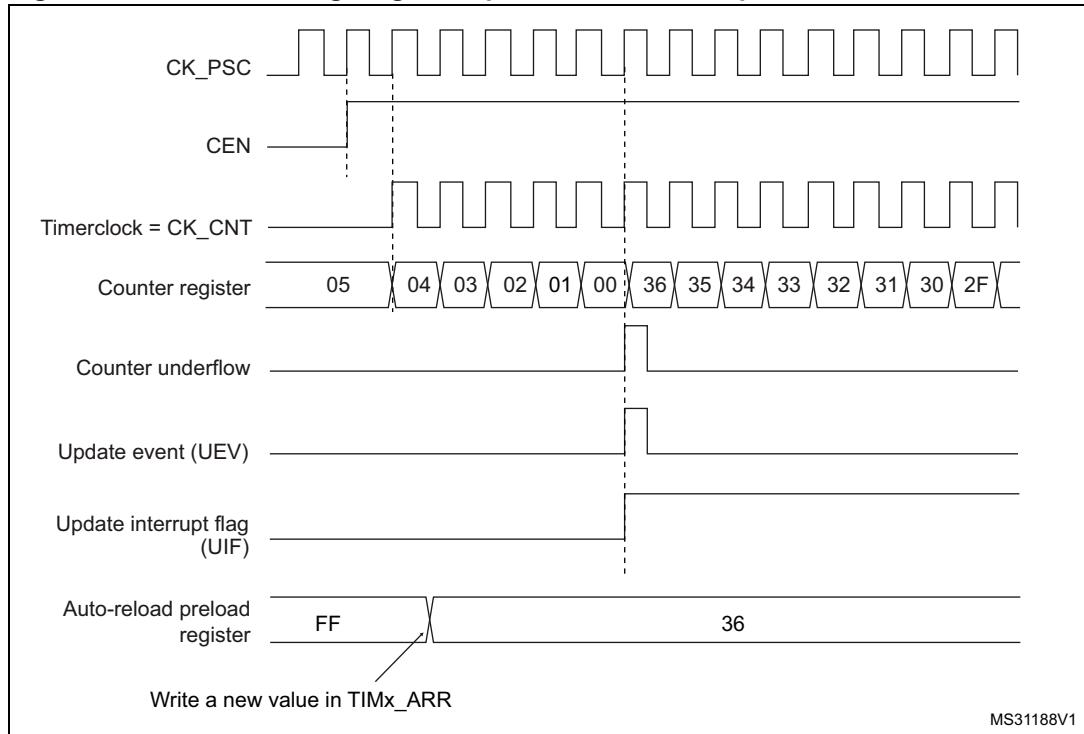


Figure 147. Counter timing diagram, update event when repetition counter is not used

Center-aligned mode (up/down counting)

In center-aligned mode, the counter counts from 0 to the auto-reload value (content of the TIMx_ARR register) – 1, generates a counter overflow event, then counts from the auto-reload value down to 1 and generates a counter underflow event. Then it restarts counting from 0.

Center-aligned mode is active when the CMS bits in TIMx_CR1 register are not equal to '00'. The Output compare interrupt flag of channels configured in output is set when: the counter counts down (Center aligned mode 1, CMS = "01"), the counter counts up (Center aligned mode 2, CMS = "10") the counter counts up and down (Center aligned mode 3, CMS = "11").

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

The update event can be generated at each counter overflow and at each counter underflow or by setting the UG bit in the TIMx_EGR register (by software or by using the slave mode controller) also generates an update event. In this case, the counter restarts counting from 0, as well as the counter of the prescaler.

The UEV update event can be disabled by software by setting the UDIS bit in the TIMx_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until UDIS bit has been written to 0. However, the counter continues counting up and down, based on the current auto-reload value.

In addition, if the URS bit (update request selection) in TIMx_CR1 register is set, setting the UG bit generates an UEV update event but without setting the UIF flag (thus no interrupt or

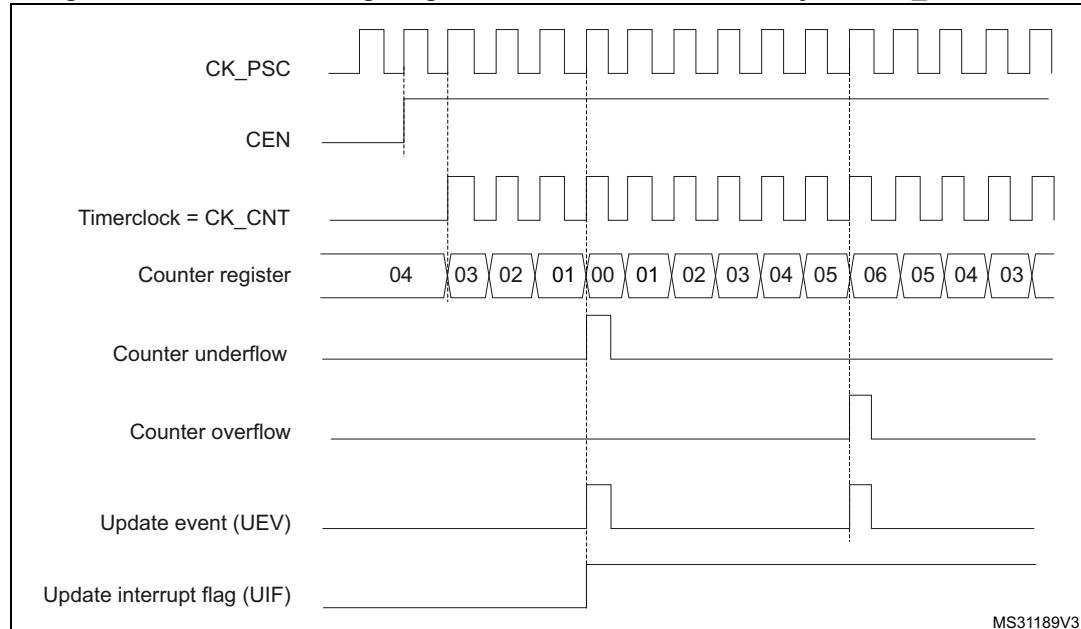
DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

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

- The repetition counter is reloaded with the content of TIMx_RCR register
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSC register)
- The auto-reload active register is updated with the preload value (content of the TIMx_ARR register). Note that if the update source is a counter overflow, the auto-reload is updated before the counter is reloaded, so that the next period is the expected one (the counter is loaded with the new value).

The following figures show some examples of the counter behavior for different clock frequencies.

Figure 148. Counter timing diagram, internal clock divided by 1, TIMx_ARR = 0x6



1. Here, center-aligned mode 1 is used (for more details refer to [Section 18.4: TIM1/TIM8 registers](#)).

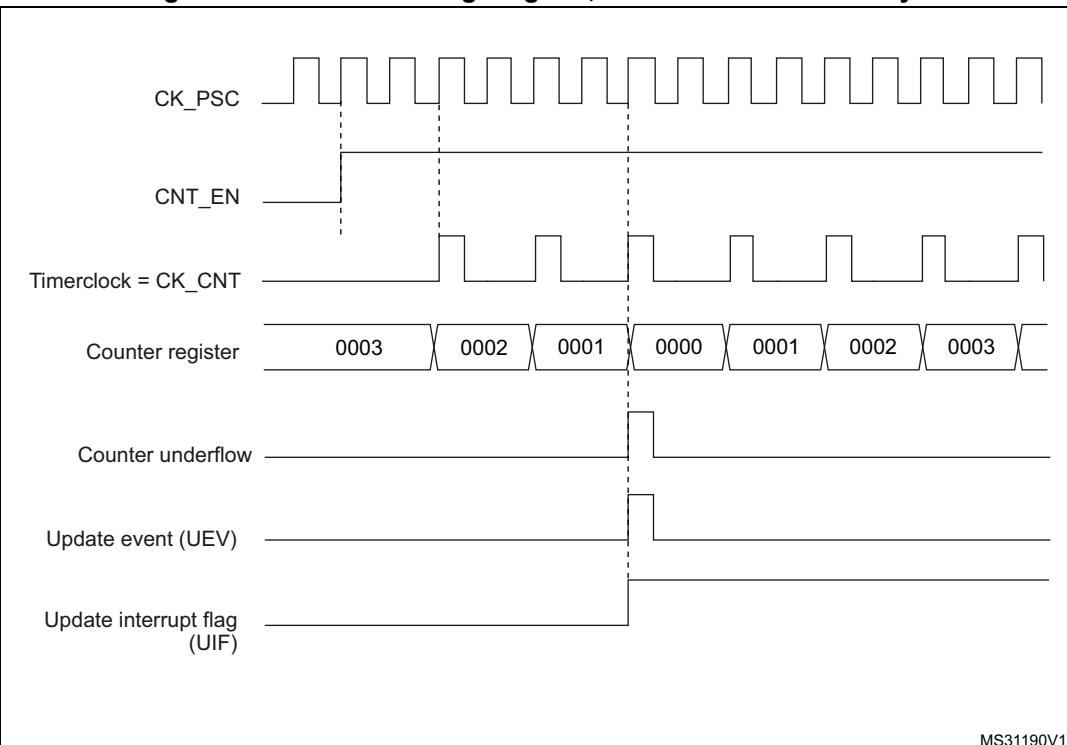
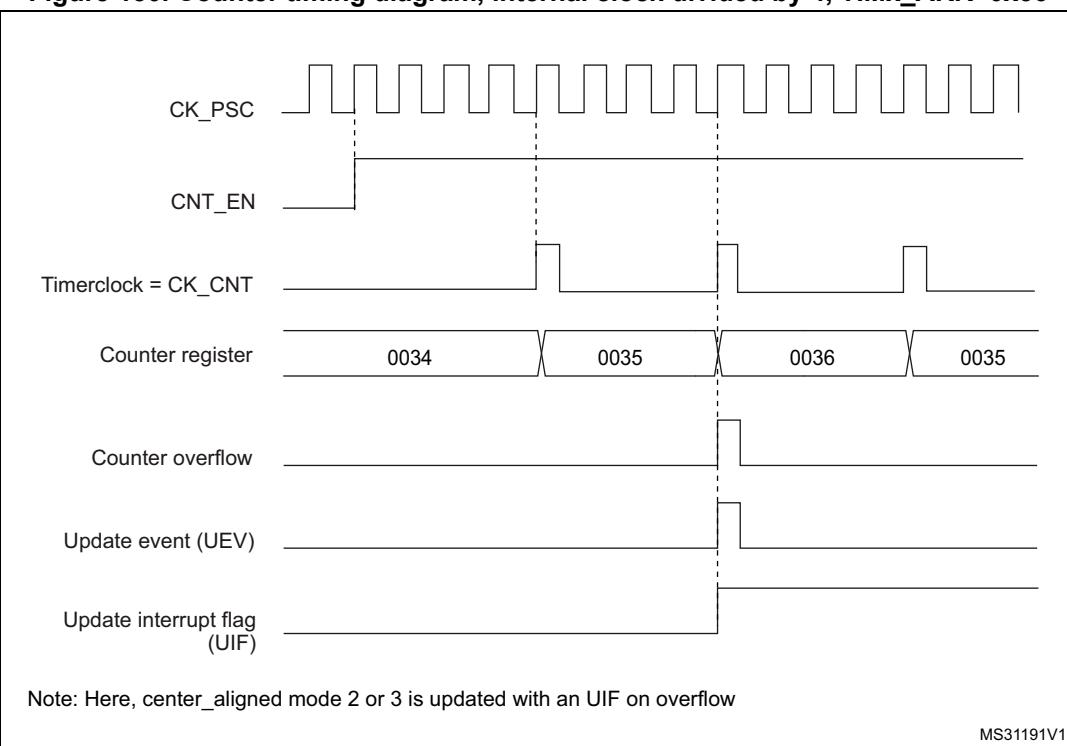
Figure 149. Counter timing diagram, internal clock divided by 2**Figure 150. Counter timing diagram, internal clock divided by 4, TIMx_ARR=0x36**

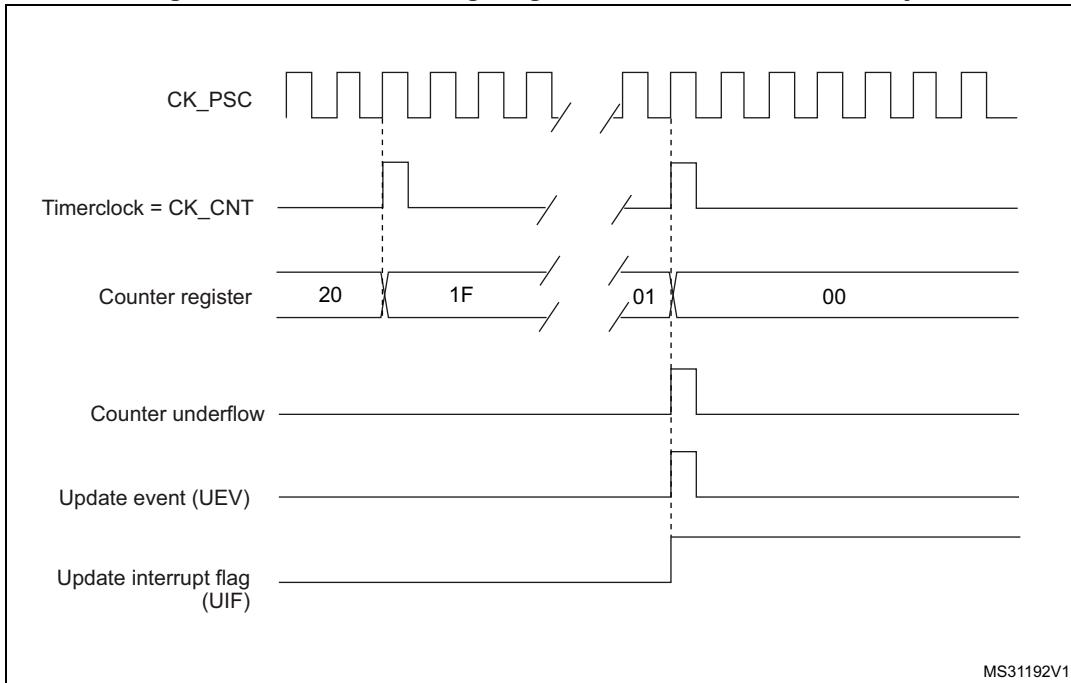
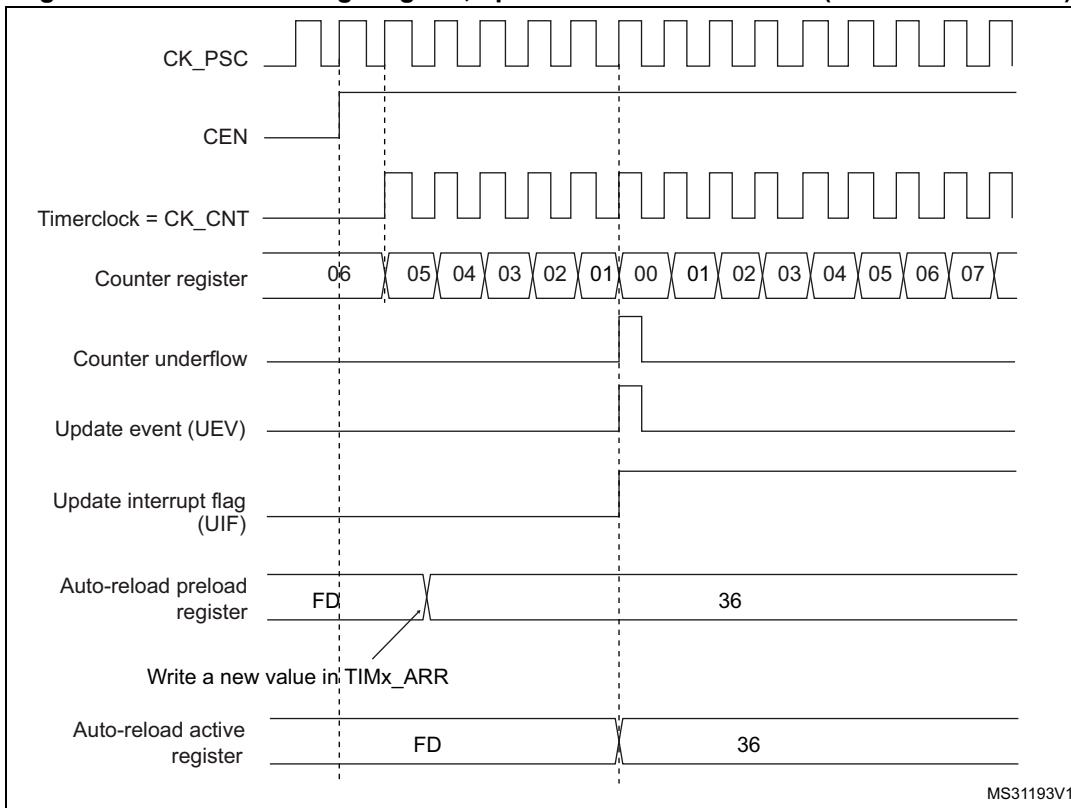
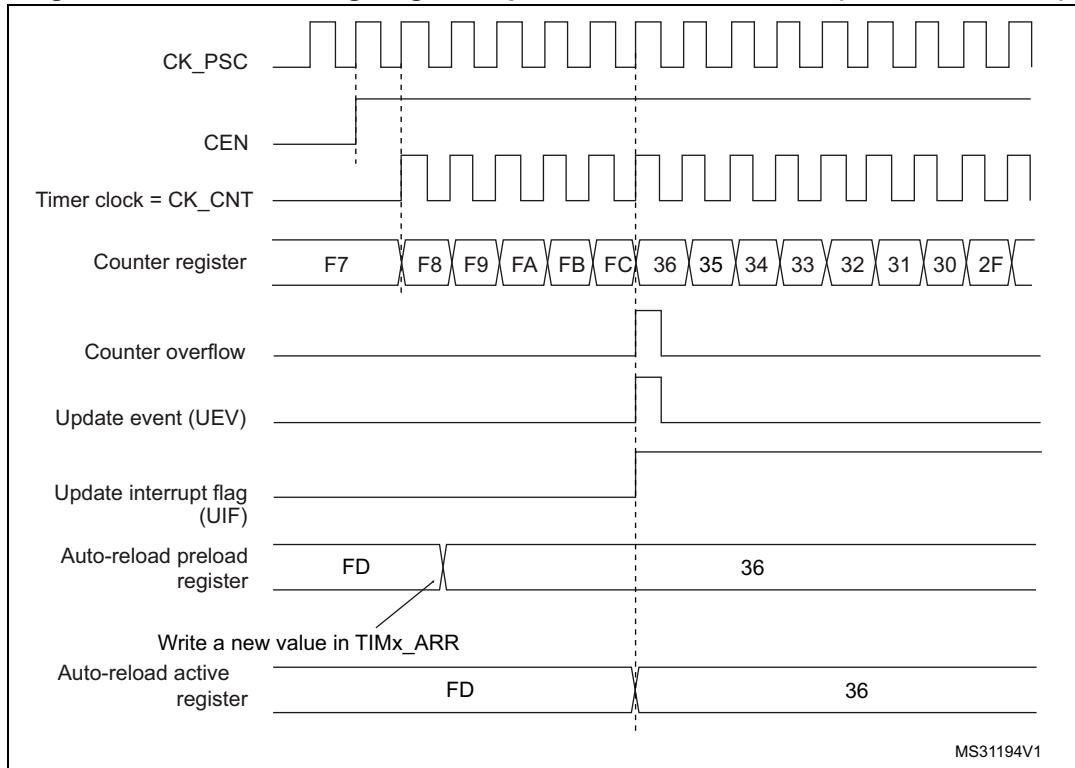
Figure 151. Counter timing diagram, internal clock divided by N**Figure 152. Counter timing diagram, update event with ARPE=1 (counter underflow)**

Figure 153. Counter timing diagram, Update event with ARPE=1 (counter overflow)

18.3.3 Repetition counter

[Section 18.3.1: Time-base unit](#) describes how the update event (UEV) is generated with respect to the counter overflows/underflows. It is actually generated only when the repetition counter has reached zero. This can be useful when generating PWM signals.

This means that data are transferred from the preload registers to the shadow registers (TIMx_ARR auto-reload register, TIMx_PSC prescaler register, but also TIMx_CCRx capture/compare registers in compare mode) every N+1 counter overflows or underflows, where N is the value in the TIMx_RCR repetition counter register.

The repetition counter is decremented:

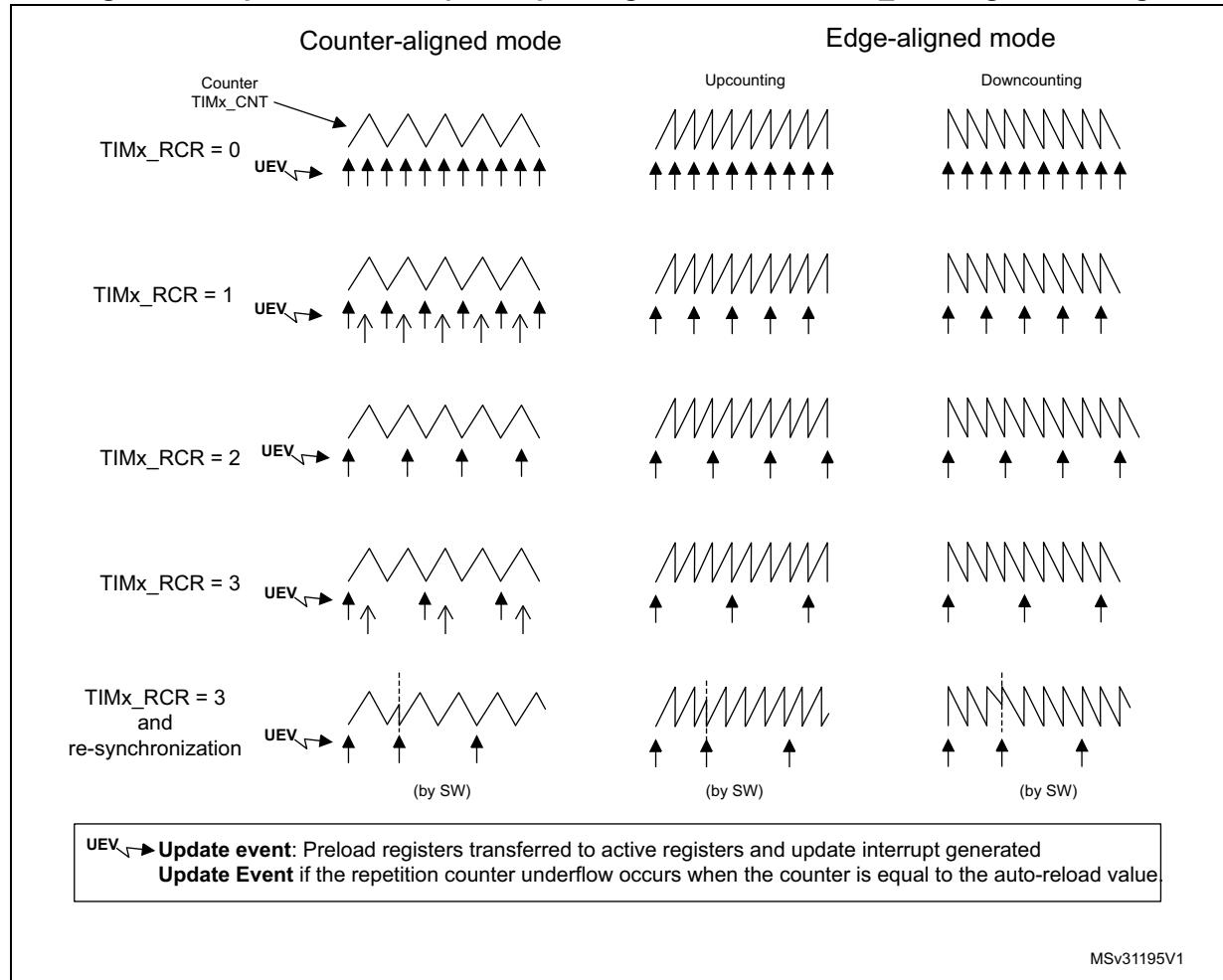
- At each counter overflow in upcounting mode,
 - At each counter underflow in downcounting mode,
 - At each counter overflow and at each counter underflow in center-aligned mode.
- Although this limits the maximum number of repetition to 32768 PWM cycles, it makes it possible to update the duty cycle twice per PWM period. When refreshing compare registers only once per PWM period in center-aligned mode, maximum resolution is $2 \times T_{ck}$, due to the symmetry of the pattern.

The repetition counter is an auto-reload type; the repetition rate is maintained as defined by the TIMx_RCR register value (refer to [Figure 154](#)). When the update event is generated by software (by setting the UG bit in TIMx_EGR register) or by hardware through the slave mode controller, it occurs immediately whatever the value of the repetition counter is and the repetition counter is reloaded with the content of the TIMx_RCR register.

In Center aligned mode, for odd values of RCR, the update event occurs either on the overflow or on the underflow depending on when the RCR register was written and when the counter was launched: if the RCR was written before launching the counter, the UEV occurs on the overflow. If the RCR was written after launching the counter, the UEV occurs on the underflow.

For example, for RCR = 3, the UEV is generated each 4th overflow or underflow event depending on when the RCR was written.

Figure 154. Update rate examples depending on mode and TIMx_RCR register settings



MSv31195V1

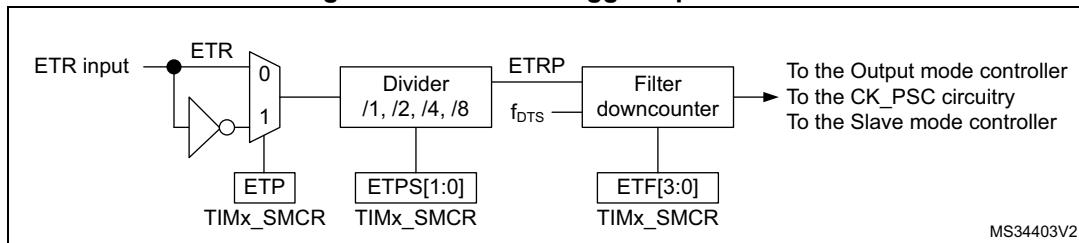
18.3.4 External trigger input

The timer features an external trigger input ETR. It can be used as:

- external clock (external clock mode 2, see [Section 18.3.5](#))
- trigger for the slave mode (see [Section 18.3.25](#))
- PWM reset input for cycle-by-cycle current regulation (see [Section 18.3.7](#))

[Figure 155](#) below describes the ETR input conditioning. The input polarity is defined with the ETP bit in TIMxSMCR register. The trigger can be prescaled with the divider programmed by the ETPS[1:0] bitfield and digitally filtered with the ETF[3:0] bitfield.

Figure 155. External trigger input block



MS34403V2

18.3.5 Clock selection

The counter clock can be provided by the following clock sources:

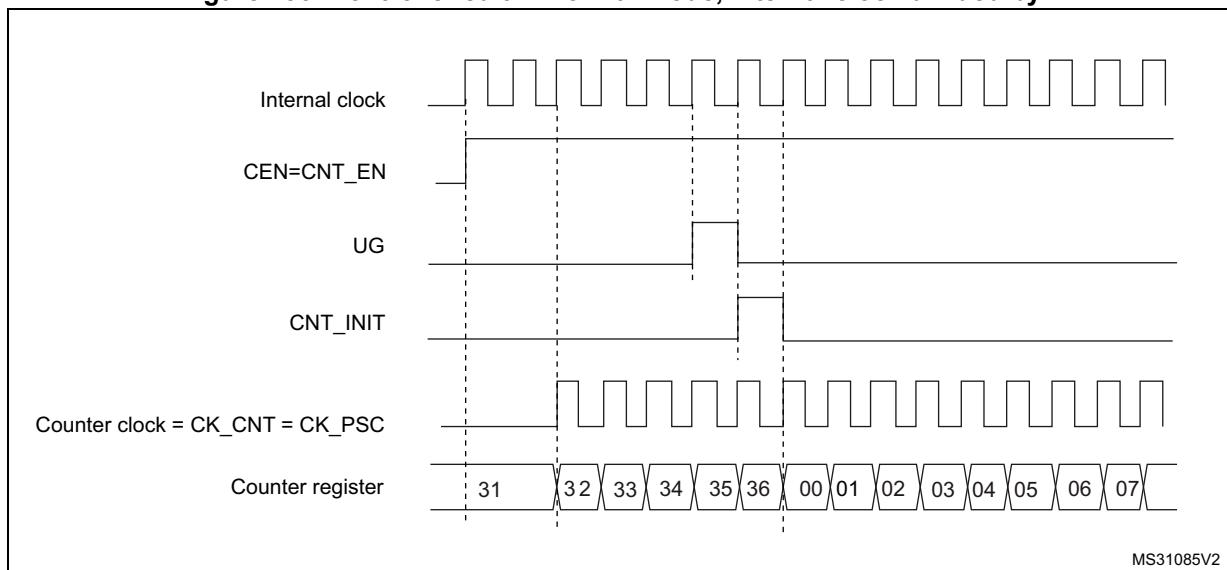
- Internal clock (CK_INT)
- External clock mode1: external input pin
- External clock mode2: external trigger input ETR
- Encoder mode

Internal clock source (CK_INT)

If the slave mode controller is disabled (SMS=000), then the CEN, DIR (in the TIMx_CR1 register) and UG bits (in the TIMx_EGR register) are actual control bits and can be changed only by software (except UG which remains cleared automatically). As soon as the CEN bit is written to 1, the prescaler is clocked by the internal clock CK_INT.

Figure 156 shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

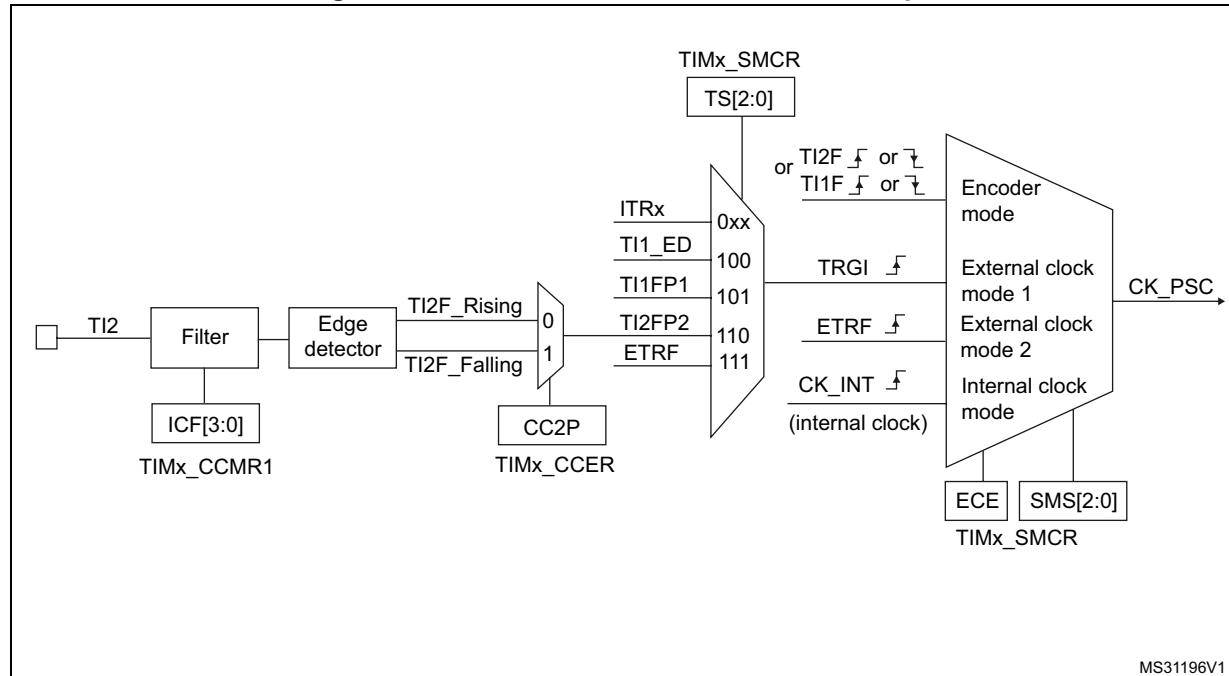
Figure 156. Control circuit in normal mode, internal clock divided by 1



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 157. TI2 external clock connection example



MS31196V1

For example, to configure the upcounter to count in response to a rising edge on the TI2 input, use the following procedure:

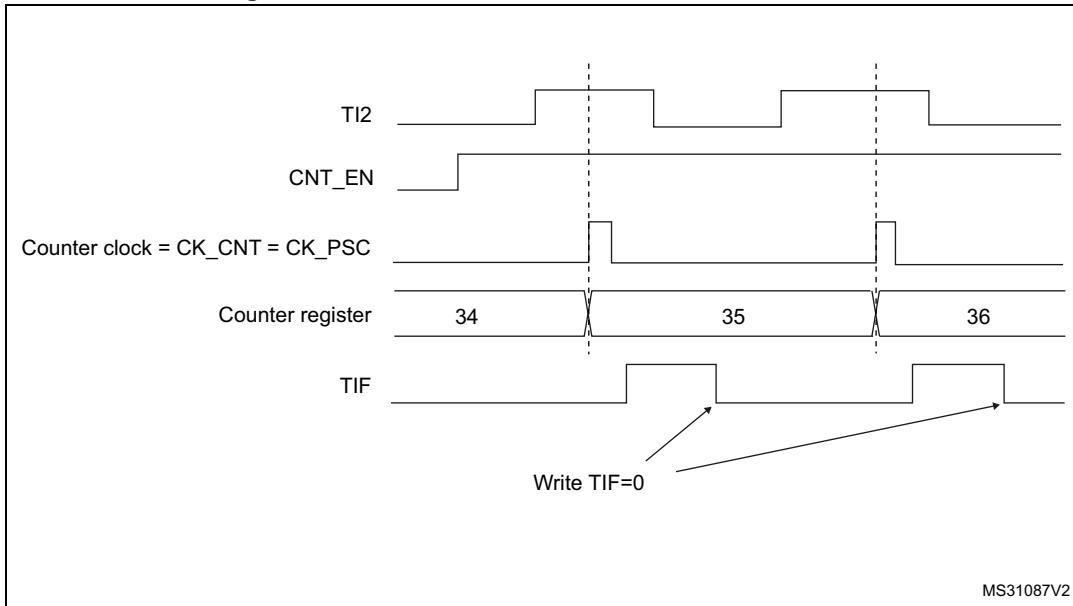
1. Configure channel 2 to detect rising edges on the TI2 input by writing CC2S = '01' in the TIMx_CCMR1 register.
2. Configure the input filter duration by writing the IC2F[3:0] bits in the TIMx_CCMR1 register (if no filter is needed, keep IC2F=0000).
3. Select rising edge polarity by writing CC2P=0 and CC2NP=0 in the TIMx_CCER register.
4. Configure the timer in external clock mode 1 by writing SMS=111 in the TIMx_SMCR register.
5. Select TI2 as the trigger input source by writing TS=110 in the TIMx_SMCR register.
6. Enable the counter by writing CEN=1 in the TIMx_CR1 register.

Note: The capture prescaler is not used for triggering, so the user does not 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 158. Control circuit in external clock mode 1



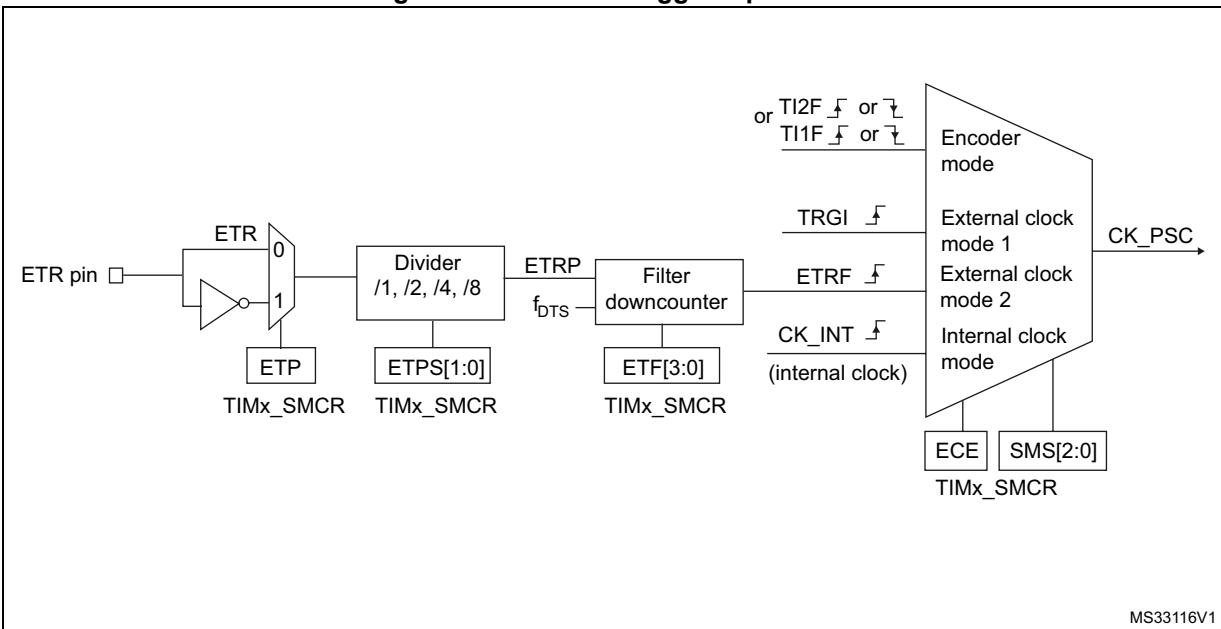
External clock source mode 2

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 159](#) gives an overview of the external trigger input block.

Figure 159. External trigger input block



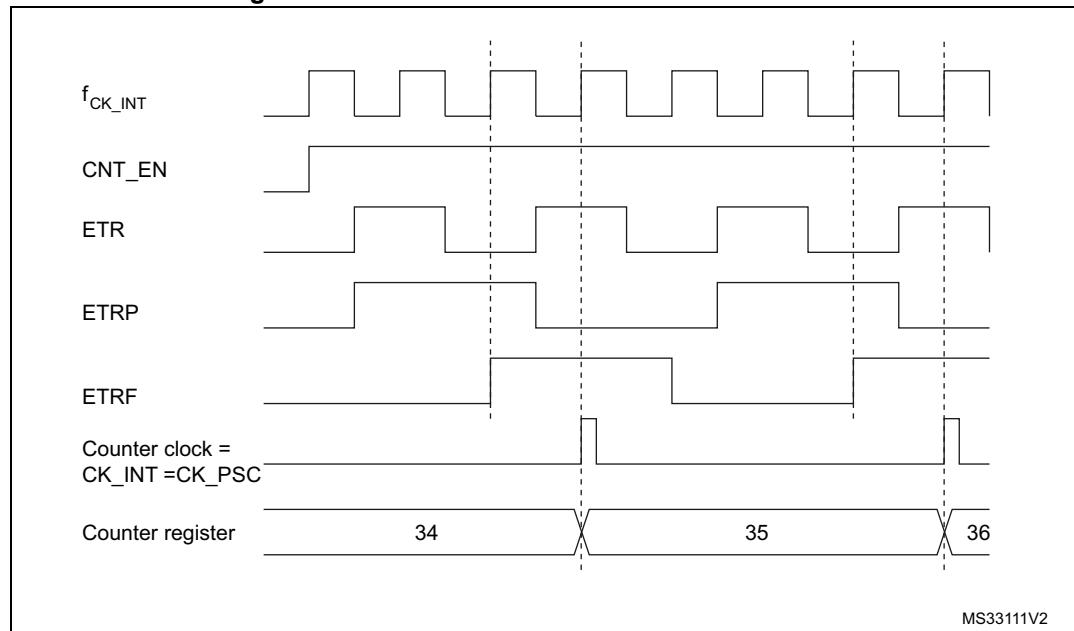
For example, to configure the upcounter to count each 2 rising edges on ETR, use the following procedure:

1. As no filter is needed in this example, write ETF[3:0]=0000 in the TIMx_SMCR register.
2. Set the prescaler by writing ETPS[1:0]=01 in the TIMx_SMCR register
3. Select rising edge detection on the ETR pin by writing ETP=0 in the TIMx_SMCR register
4. Enable external clock mode 2 by writing ECE=1 in the TIMx_SMCR register.
5. Enable the counter by writing CEN=1 in the TIMx_CR1 register.

The counter counts once each 2 ETR rising edges.

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

Figure 160. Control circuit in external clock mode 2



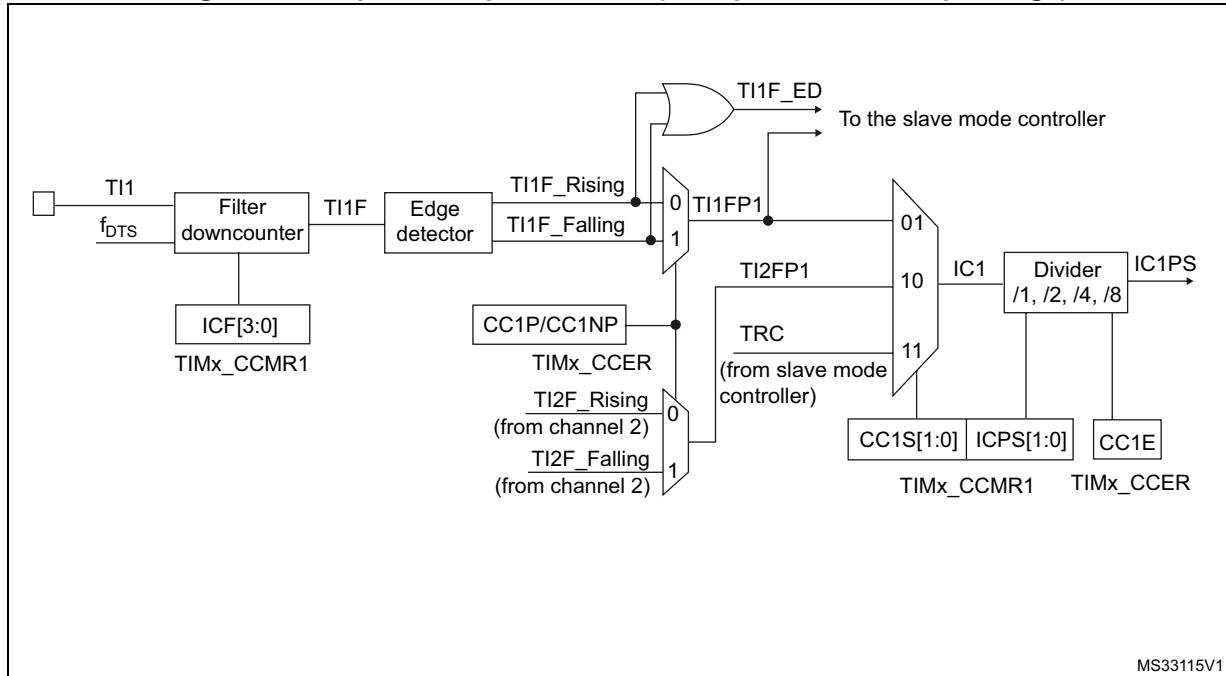
18.3.6 Capture/compare channels

Each Capture/Compare channel is built around a capture/compare register (including a shadow register), an input stage for capture (with digital filter, multiplexing, and prescaler, except for channels 5 and 6) and an output stage (with comparator and output control).

Figure 161 to *Figure 164* give an overview of one Capture/Compare channel.

The input stage samples the corresponding TI_x input to generate a filtered signal TI_xF. Then, an edge detector with polarity selection generates a signal (TI_xFP_x) which can be used as trigger input by the slave mode controller or as the capture command. It is prescaled before the capture register (IC_xPS).

Figure 161. Capture/compare channel (example: channel 1 input stage)



The output stage generates an intermediate waveform which is then used for reference: OC_xRef (active high). The polarity acts at the end of the chain.

Figure 162. Capture/compare channel 1 main circuit

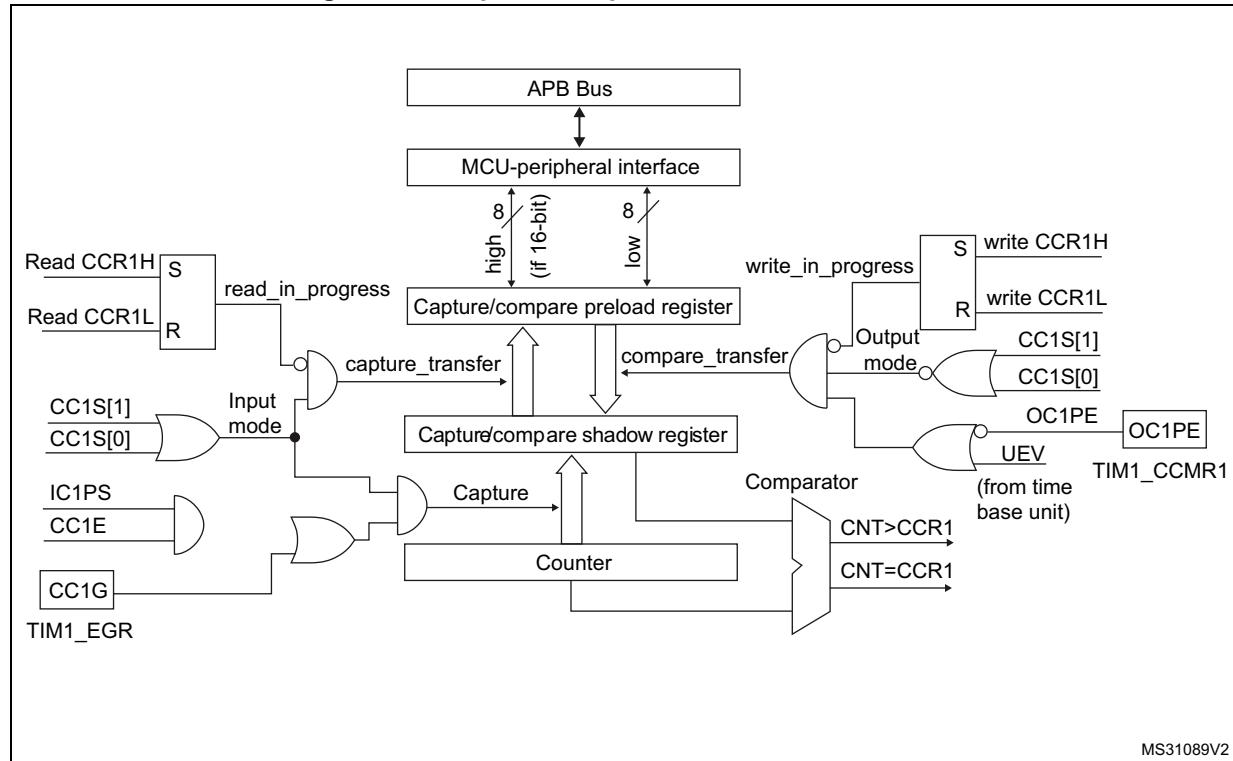
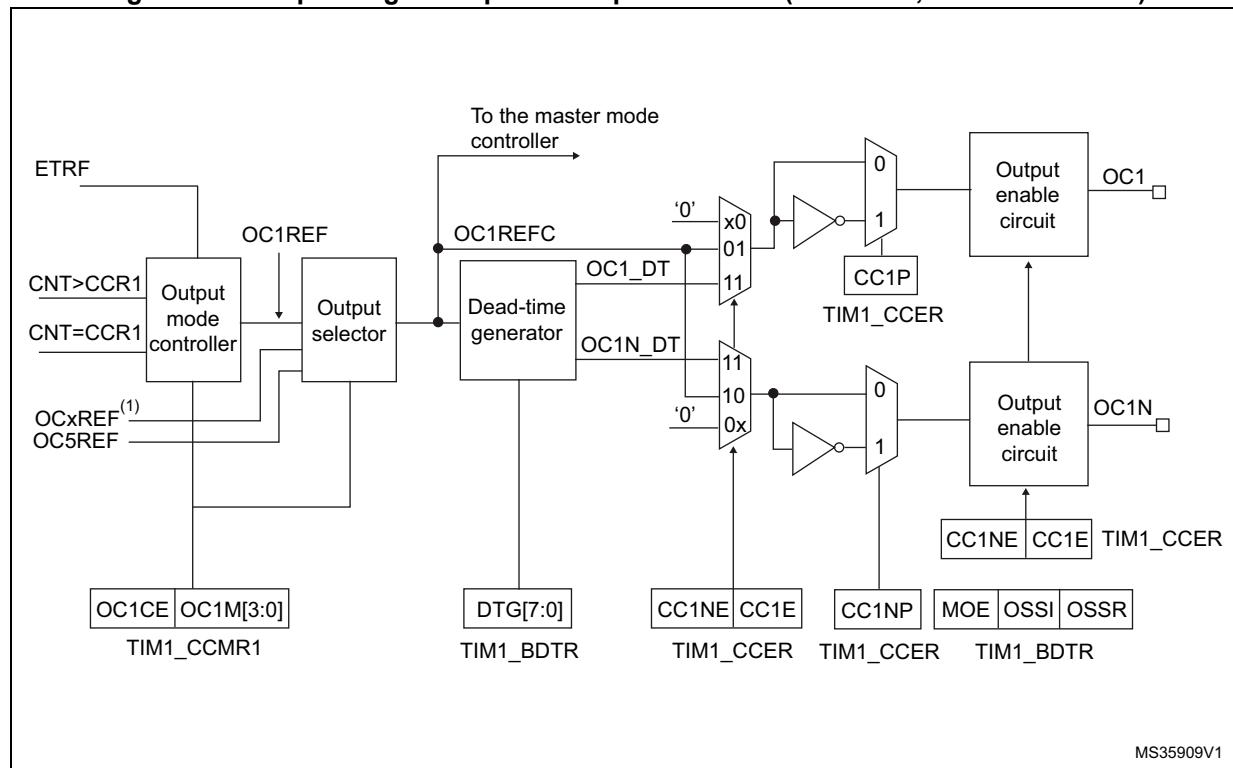


Figure 163. Output stage of capture/compare channel (channel 1, idem ch. 2 and 3)



1. OCxREF, where x is the rank of the complementary channel

Figure 164. Output stage of capture/compare channel (channel 4)

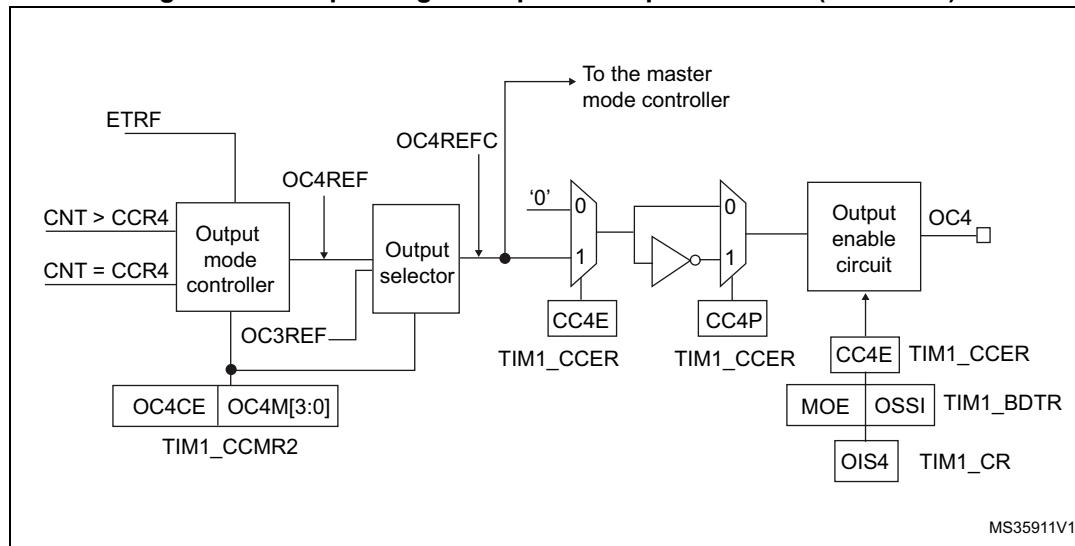
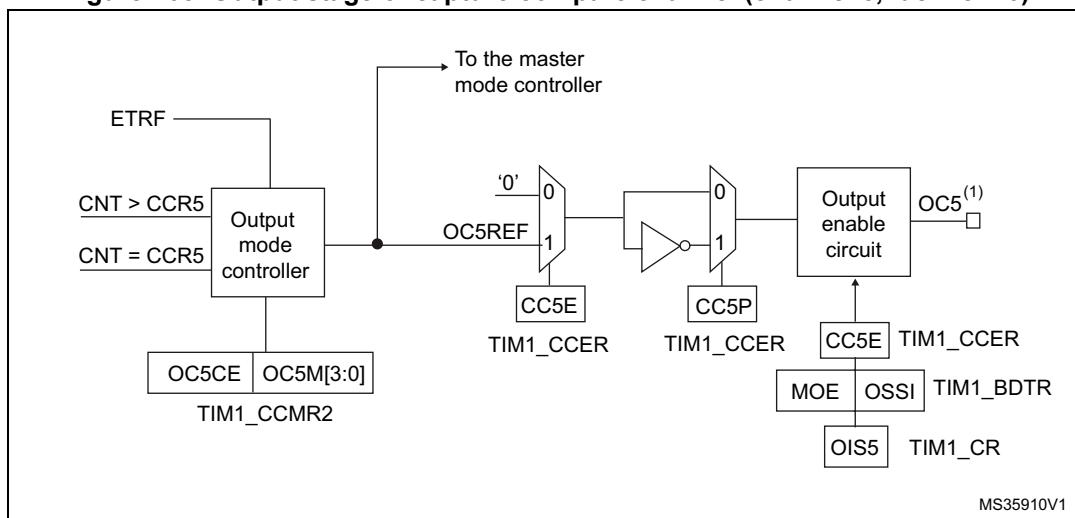


Figure 165. Output stage of capture/compare channel (channel 5, idem ch. 6)



1. Not available externally.

The capture/compare block is made of one preload register and one shadow register. Write and read always access the preload register.

In capture mode, captures are actually done in the shadow register, which is copied into the preload register.

In compare mode, the content of the preload register is copied into the shadow register which is compared to the counter.

18.3.7 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:

1. Select the active input: TIMx_CCR1 must be linked to the TI1 input, so write the CC1S bits to 01 in the TIMx_CCMR1 register. As soon as CC1S becomes different from 00, the channel is configured in input and the TIMx_CCR1 register becomes read-only.
2. 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.
3. Select the edge of the active transition on the TI1 channel by writing CC1P and CC1NP bits to 0 in the TIMx_CCER register (rising edge in this case).
4. Program the input prescaler. In our example, we wish the capture to be performed at each valid transition, so the prescaler is disabled (write IC1PS bits to '00' in the TIMx_CCMR1 register).
5. Enable capture from the counter into the capture register by setting the CC1E bit in the TIMx_CCER register.
6. If needed, enable the related interrupt request by setting the CC1IE bit in the TIMx_DIER register, 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.

18.3.8 PWM input mode

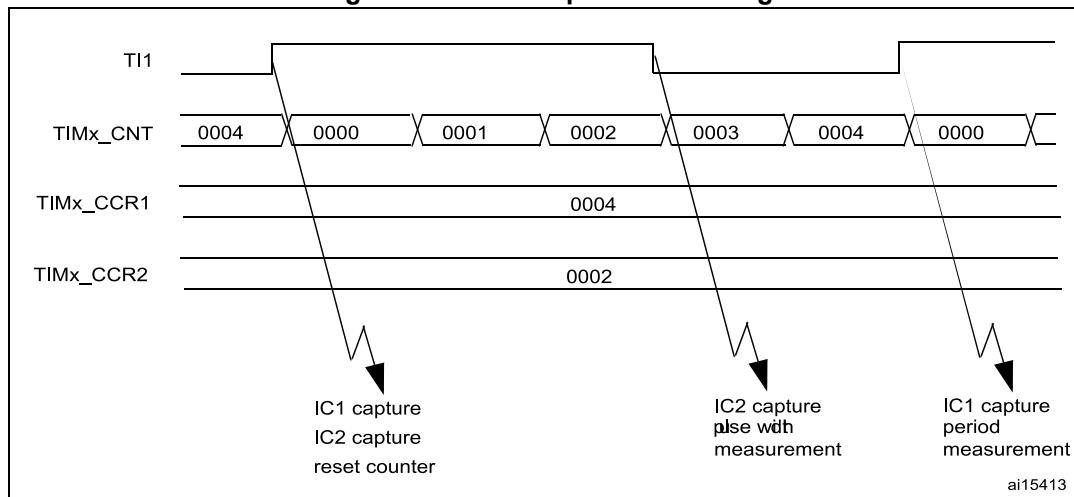
This mode is a particular case of input capture mode. The procedure is the same except:

- 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, the user 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):

1. Select the active input for TIMx_CCR1: write the CC1S bits to 01 in the TIMx_CCMR1 register (TI1 selected).
2. Select the active polarity for TI1FP1 (used both for capture in TIMx_CCR1 and counter clear): write the CC1P and CC1NP bits to '0' (active on rising edge).
3. Select the active input for TIMx_CCR2: write the CC2S bits to 10 in the TIMx_CCMR1 register (TI1 selected).
4. Select the active polarity for TI1FP2 (used for capture in TIMx_CCR2): write the CC2P and CC2NP bits to CC2P/CC2NP='10' (active on falling edge).
5. Select the valid trigger input: write the TS bits to 101 in the TIMx_SMCR register (TI1FP1 selected).
6. Configure the slave mode controller in reset mode: write the SMS bits to 0100 in the TIMx_SMCR register.
7. Enable the captures: write the CC1E and CC2E bits to '1' in the TIMx_CCER register.

Figure 166. PWM input mode timing



18.3.9 Forced output mode

In output mode (CCxS bits = 00 in the TIMx_CCMRx register), each output compare signal (OCxREF and then OCx/OCxN) can be forced to active or inactive level directly by software, independently of any comparison between the output compare register and the counter.

To force an output compare signal (OCXREF/OCx) to its active level, user just needs to write 0101 in the OCxM bits in the corresponding TIMx_CCMRx register. Thus OCXREF is forced high (OCxREF is always active high) and OCx get opposite value to CCxP polarity bit.

For example: CCxP=0 (OCx active high) => OCx is forced to high level.

The OCxREF signal can be forced low by writing the OCxM bits to 0100 in the TIMx_CCMRx register.

Anyway, the comparison between the TIMx_CCRx shadow register and the counter is still performed and allows the flag to be set. Interrupt and DMA requests can be sent accordingly. This is described in the output compare mode section below.

18.3.10 Output compare mode

This function is used to control an output waveform or indicate when a period of time has elapsed. Channels 1 to 4 can be output, while Channel 5 and 6 are only available inside the microcontroller (for instance, for compound waveform generation or for ADC triggering).

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=0000), be set active (OCXM=0001), be set inactive (OCXM=0010) or can toggle (OCXM=0011) on match.
- Sets a flag in the interrupt status register (CCxIF bit in the TIMx_SR register).
- Generates an interrupt if the corresponding interrupt mask is set (CCXIE bit in the TIMx_DIER register).
- Sends a DMA request if the corresponding enable bit is set (CCxDE bit in the TIMx_DIER register, CCDS bit in the TIMx_CR2 register for the DMA request selection).

The TIMx_CCRx registers can be programmed with or without preload registers using the OCxPE bit in the TIMx_CCMRx register.

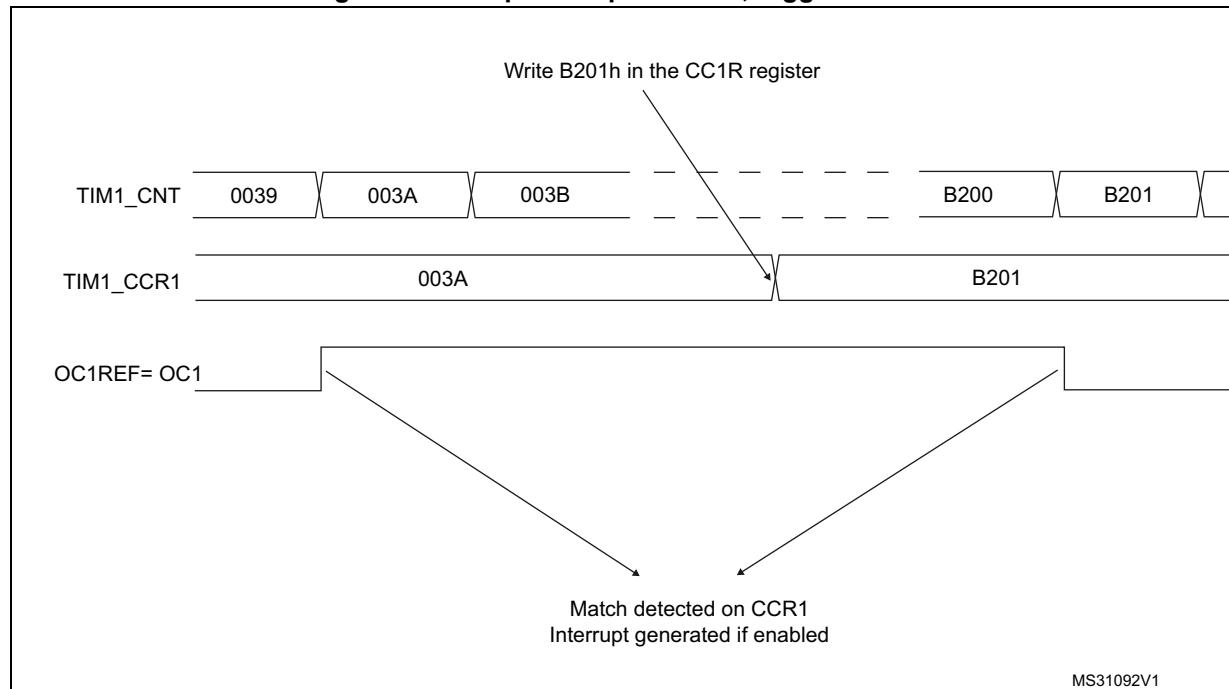
In output compare mode, the update event UEV has no effect on OCxREF and OCx output. The timing resolution is one count of the counter. Output compare mode can also be used to output a single pulse (in One Pulse mode).

Procedure

1. Select the counter clock (internal, external, prescaler).
2. Write the desired data in the TIMx_ARR and TIMx_CCRx registers.
3. Set the CCxIE bit if an interrupt request is to be generated.
4. Select the output mode. For example:
 - Write OCxM = 0011 to toggle OCx output pin when CNT matches CCRx
 - Write OCxPE = 0 to disable preload register
 - Write CCxP = 0 to select active high polarity
 - Write CCxE = 1 to enable the output
5. Enable the counter by setting the CEN bit in the TIMx_CR1 register.

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 167](#).

Figure 167. Output compare mode, toggle on OC1



18.3.11 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 '0110' (PWM mode 1) or '0111' (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 a combination of the CCxE, CCxNE, MOE, OSS1 and OSSR bits (TIMx_CCER and TIMx_BDTR registers). Refer to the TIMx_CCER register description for more details.

In PWM mode (1 or 2), TIMx_CNT and TIMx_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).

The timer is able to generate PWM in edge-aligned mode or center-aligned mode depending on the CMS bits in the TIMx_CR1 register.

PWM edge-aligned mode

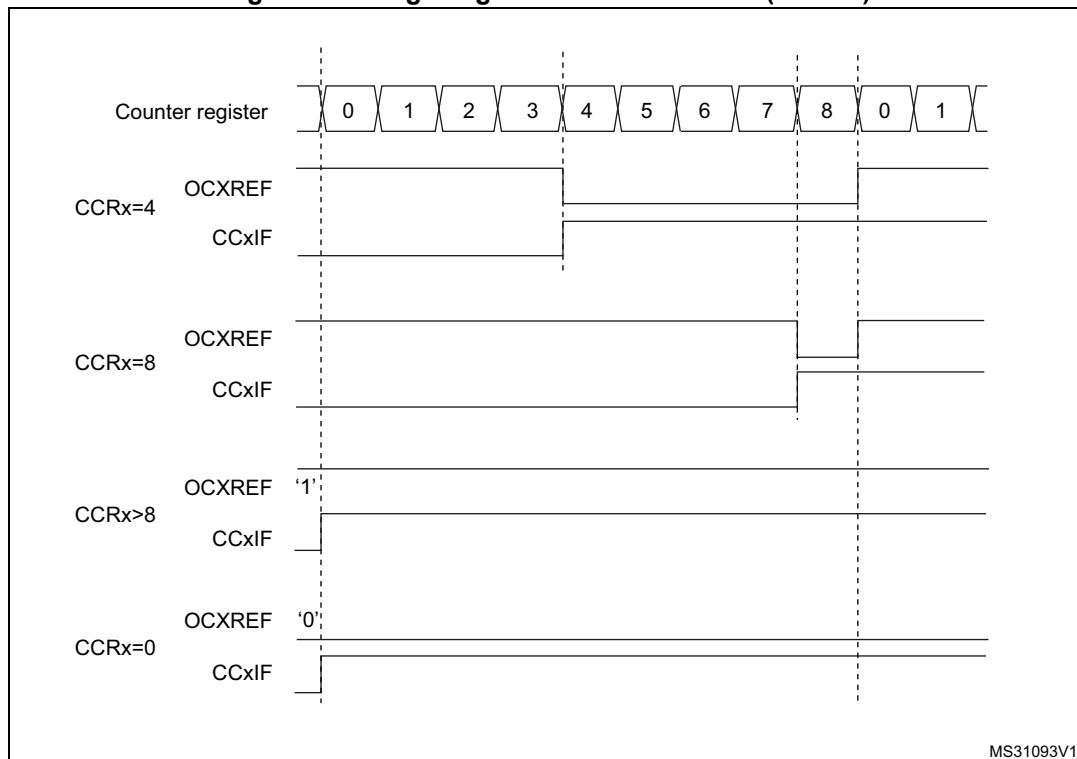
- Upcounting configuration

Upcounting is active when the DIR bit in the TIMx_CR1 register is low. Refer to the [Upcounting mode on page 514](#).

In the following example, we consider PWM mode 1. The reference PWM signal OCxREF is high as long as $\text{TIMx_CNT} < \text{TIMx_CCR}_x$ else it becomes low. If the compare value in TIMx_CCR_x 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 168](#) shows some edge-aligned PWM waveforms in an example where $\text{TIMx_ARR}=8$.

Figure 168. Edge-aligned PWM waveforms (ARR=8)



- Downcounting configuration

Downcounting is active when DIR bit in TIMx_CR1 register is high. Refer to the [Downcounting mode on page 518](#)

In PWM mode 1, the reference signal OCxRef is low as long as $\text{TIMx_CNT} > \text{TIMx_CCR}_x$ else it becomes high. If the compare value in TIMx_CCR_x 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

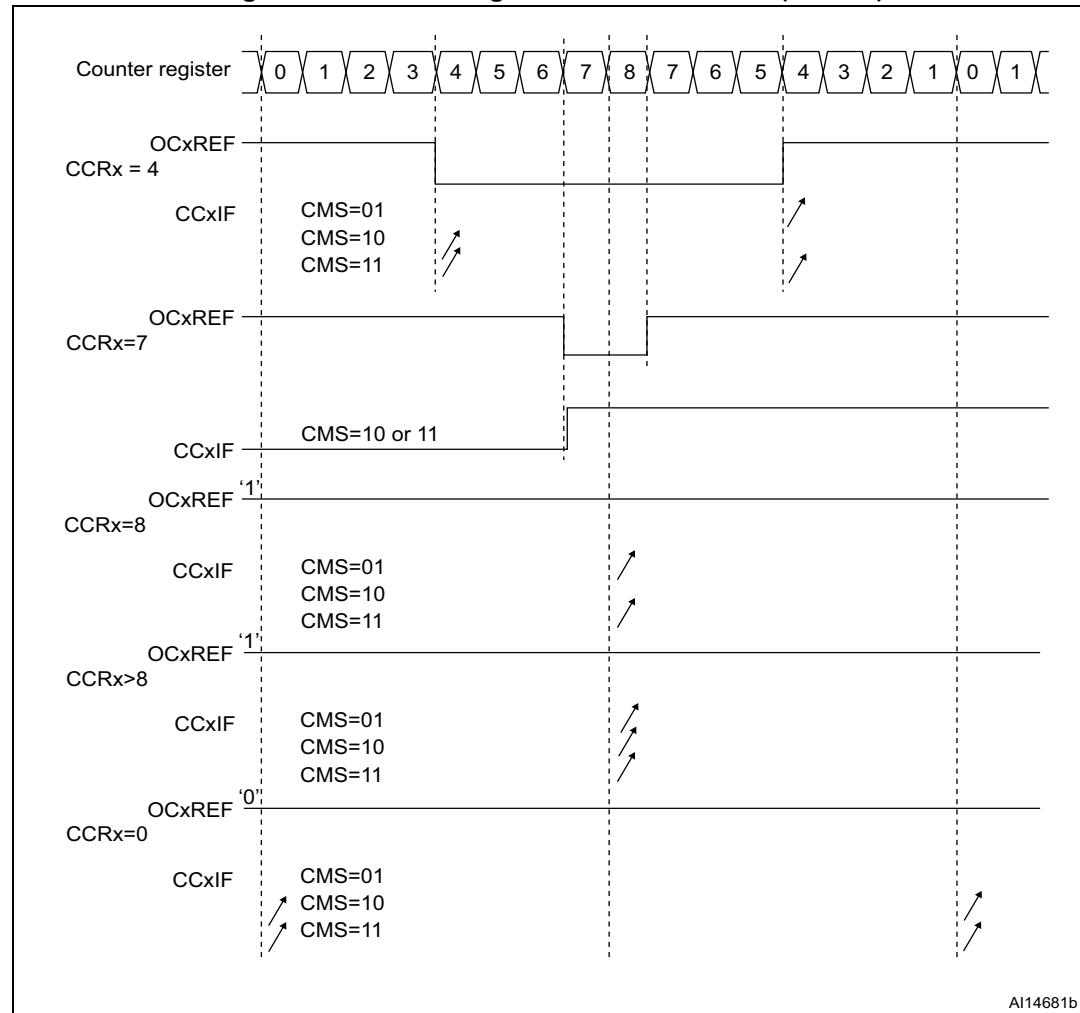
Center-aligned mode is active when the CMS bits in TIMx_CR1 register are different from '00' (all the remaining configurations having the same effect on the OCxRef/OCx signals). The compare flag is set when the counter counts up, when it counts down or both when it counts up and down depending on the 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 521](#).

Figure 169 shows some center-aligned PWM waveforms in an example where:

- $\text{TIMx_ARR}=8$,
- 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 169. Center-aligned PWM waveforms (ARR=8)



AI14681b

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.

18.3.12 Asymmetric PWM mode

Asymmetric mode allows two center-aligned PWM signals to be generated with a programmable phase shift. While the frequency is determined by the value of the TIMx_ARR register, the duty cycle and the phase-shift are determined by a pair of TIMx_CCRx register. One register controls the PWM during up-counting, the second during down counting, so that PWM is adjusted every half PWM cycle:

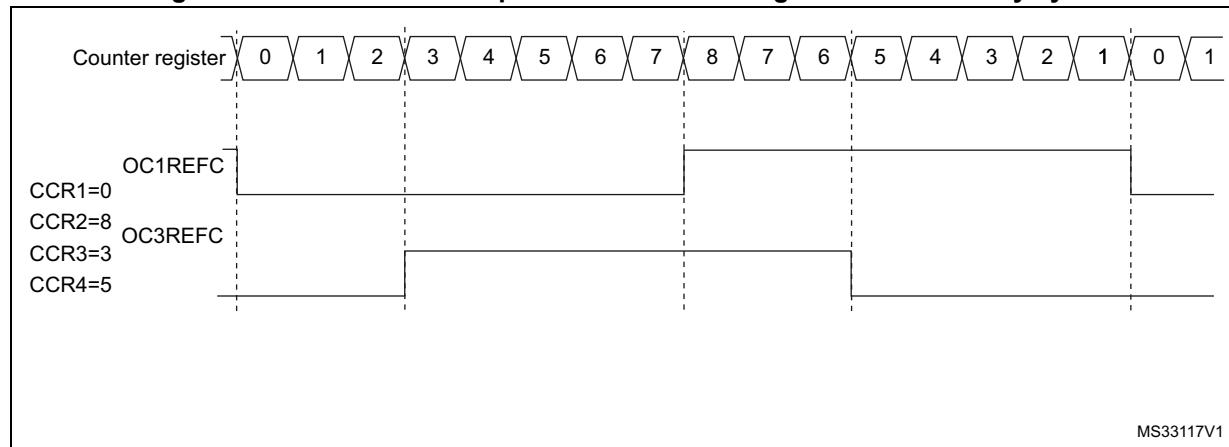
- OC1REFC (or OC2REFC) is controlled by TIMx_CCR1 and TIMx_CCR2
- OC3REFC (or OC4REFC) is controlled by TIMx_CCR3 and TIMx_CCR4

Asymmetric PWM mode can be selected independently on two channel (one OCx output per pair of CCR registers) by writing ‘1110’ (Asymmetric PWM mode 1) or ‘1111’ (Asymmetric PWM mode 2) in the OCxM bits in the TIMx_CCMRx register.

Note: The OCxM[3:0] bit field is split into two parts for compatibility reasons, the most significant bit is not contiguous with the 3 least significant ones.

When a given channel is used as asymmetric PWM channel, its complementary channel can also be used. For instance, if an OC1REFC signal is generated on channel 1 (Asymmetric PWM mode 1), it is possible to output either the OC2REF signal on channel 2, or an OC2REFC signal resulting from asymmetric PWM mode 1.

Figure 170 represents an example of signals that can be generated using Asymmetric PWM mode (channels 1 to 4 are configured in Asymmetric PWM mode 1). Together with the deadtime generator, this allows a full-bridge phase-shifted DC to DC converter to be controlled.

Figure 170. Generation of 2 phase-shifted PWM signals with 50% duty cycle

18.3.13 Combined PWM mode

Combined PWM mode allows two edge or center-aligned PWM signals to be generated with programmable delay and phase shift between respective pulses. While the frequency is determined by the value of the TIMx_ARR register, the duty cycle and delay are determined by the two TIMx_CCRx registers. The resulting signals, OCxREFC, are made of an OR or AND logical combination of two reference PWMs:

- OC1REFC (or OC2REFC) is controlled by TIMx_CCR1 and TIMx_CCR2
- OC3REFC (or OC4REFC) is controlled by TIMx_CCR3 and TIMx_CCR4

Combined PWM mode can be selected independently on two channels (one OCx output per pair of CCR registers) by writing ‘1100’ (Combined PWM mode 1) or ‘1101’ (Combined PWM mode 2) in the OCxM bits in the TIMx_CCMRx register.

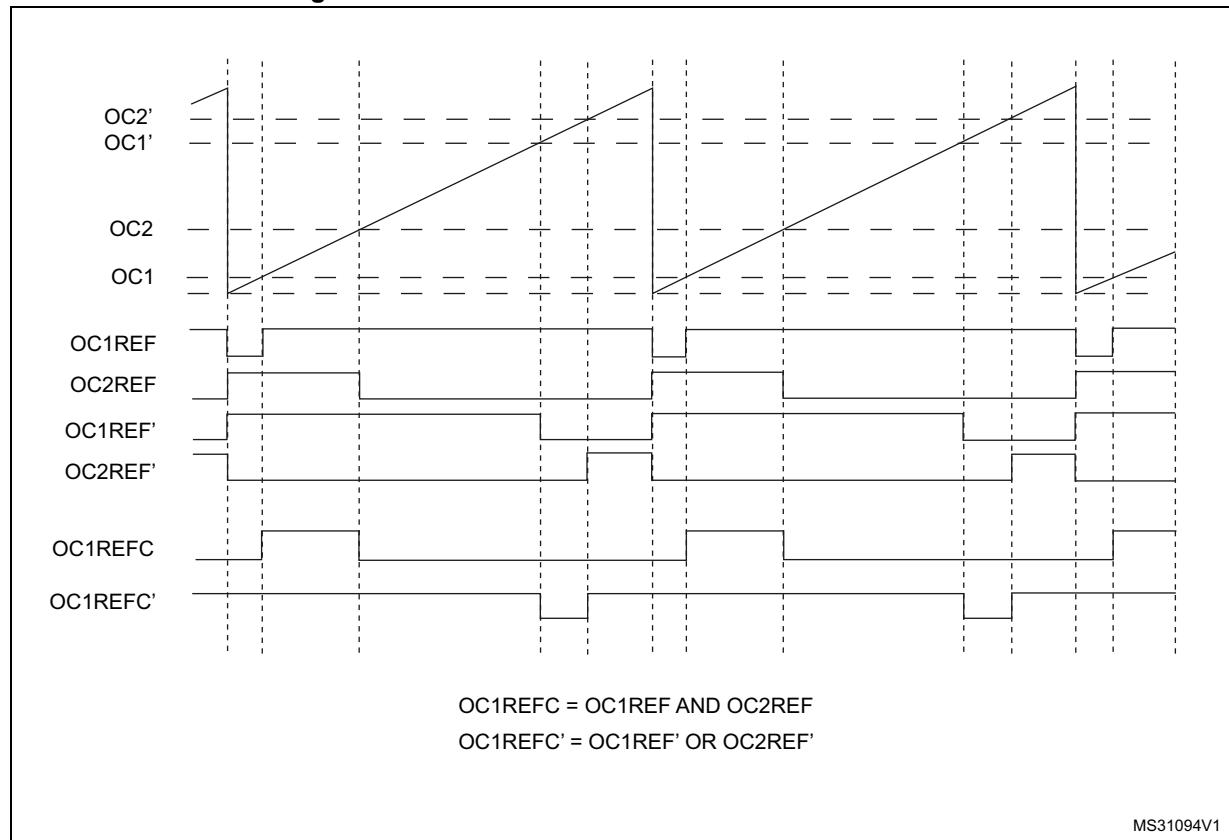
When a given channel is used as combined PWM channel, its complementary channel must be configured in the opposite PWM mode (for instance, one in Combined PWM mode 1 and the other in Combined PWM mode 2).

Note:

The OCxM[3:0] bit field is split into two parts for compatibility reasons, the most significant bit is not contiguous with the 3 least significant ones.

Figure 171 represents an example of signals that can be generated using Asymmetric PWM mode, obtained with the following configuration:

- Channel 1 is configured in Combined PWM mode 2,
- Channel 2 is configured in PWM mode 1,
- Channel 3 is configured in Combined PWM mode 2,
- Channel 4 is configured in PWM mode 1.

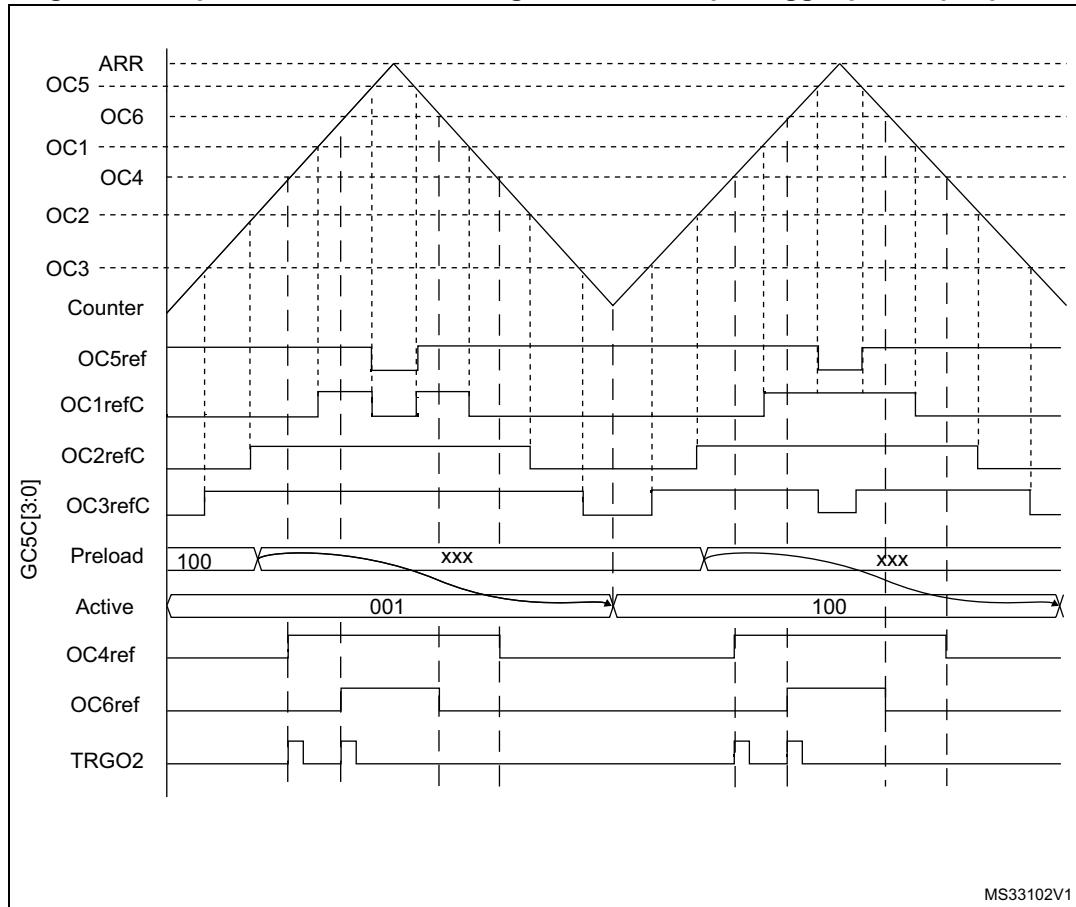
Figure 171. Combined PWM mode on channel 1 and 3

18.3.14 Combined 3-phase PWM mode

Combined 3-phase PWM mode allows one to three center-aligned PWM signals to be generated with a single programmable signal ANDed in the middle of the pulses. The OC5REF signal is used to define the resulting combined signal. The 3-bits GC5C[3:1] in the TIMx_CCR5 allow selection on which reference signal the OC5REF is combined. The resulting signals, OCxREFC, are made of an AND logical combination of two reference PWMs:

- If GC5C1 is set, OC1REFC is controlled by TIMx_CCR1 and TIMx_CCR5
- If GC5C2 is set, OC2REFC is controlled by TIMx_CCR2 and TIMx_CCR5
- If GC5C3 is set, OC3REFC is controlled by TIMx_CCR3 and TIMx_CCR5

Combined 3-phase PWM mode can be selected independently on channels 1 to 3 by setting at least one of the 3-bits GC5C[3:1].

Figure 172. 3-phase combined PWM signals with multiple trigger pulses per period

The TRGO2 waveform shows how the ADC can be synchronized on given 3-phase PWM signals. Refer to [Section 18.3.26: ADC synchronization](#) for more details.

18.3.15 Complementary outputs and dead-time insertion

The advanced-control timers (TIM1/TIM8) 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 TIM_x_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 TIM_x_CCER register and the MOE, OIS_x, OIS_{xN}, OSSI and OSSR bits in the TIM_x_BDTR and TIM_x_CR2 registers. Refer to

[Table 115: Output control bits for complementary OC_x and OC_{xN} channels with break feature on page 589](#) for more details. In particular, the dead-time is activated when switching to the idle state (MOE falling down to 0).

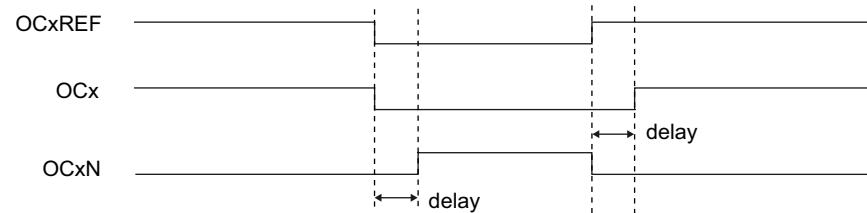
Dead-time insertion is enabled by setting both CCxE and CCxNE bits, and the MOE bit if the break circuit is present. There is one 10-bit dead-time generator for each channel. From a reference waveform OCxREF, it generates 2 outputs OCx and OCxN. If OCx and OCxN are active high:

- The OCx output signal is the same as the reference signal except for the rising edge, which is delayed relative to the reference rising edge.
- The OCxN output signal is the opposite of the reference signal except for the rising edge, which is delayed relative to the reference falling edge.

If the delay is greater than the width of the active output (OCx or OCxN) then the corresponding pulse is not generated.

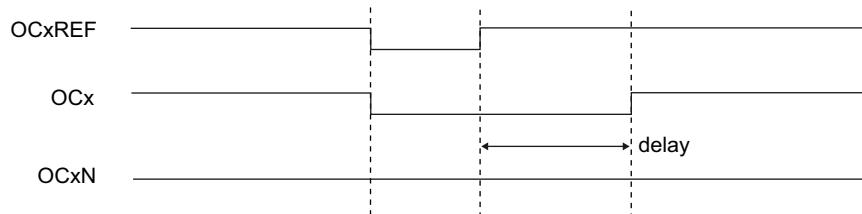
The following figures show the relationships between the output signals of the dead-time generator and the reference signal OCxREF. (we suppose CCxP=0, CCxNP=0, MOE=1, CCxE=1 and CCxNE=1 in these examples)

Figure 173. Complementary output with dead-time insertion

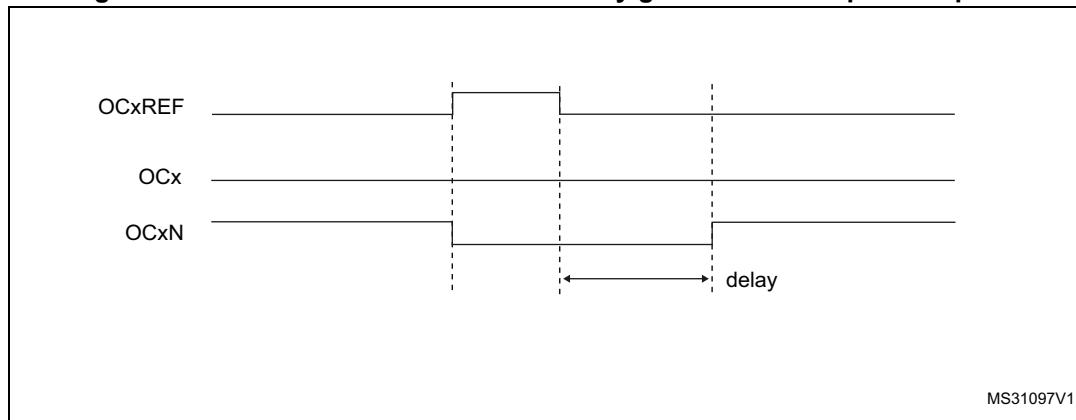


MS31095V1

Figure 174. Dead-time waveforms with delay greater than the negative pulse



MS31096V1

Figure 175. Dead-time waveforms with delay greater than the positive pulse

The dead-time delay is the same for each of the channels and is programmable with the DTG bits in the TIMx_BDTR register. Refer to [Section 18.4.18: TIMx break and dead-time register \(TIMx_BDTR\)\(x = 1, 8\)](#) for delay calculation.

Re-directing OCxREF to OCx or OCxN

In output mode (forced, output compare or PWM), OCxREF can be re-directed to the OCx output or to OCxN output by configuring the CCxE and CCxNE bits in the TIMx_CCER register.

This allows 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.

18.3.16 Using the break function

The purpose of the break function is to protect power switches driven by PWM signals generated with the TIM1 and TIM8 timers. The two break inputs are usually connected to fault outputs of power stages and 3-phase inverters. When activated, the break circuitry shuts down the PWM outputs and forces them to a predefined safe state. A number of internal MCU events can also be selected to trigger an output shut-down.

The break features two channels. A break channel which gathers both system-level fault (clock failure,...) and application fault (from input pins), and can force the outputs to a predefined level (either active or inactive) after a deadtime duration. A break2 channel which only includes application faults and is able to force the outputs to an inactive state.

The output enable signal and output levels during break are depending on several control bits:

- the MOE bit in TIMx_BDTR register allows to enable /disable the outputs by software and is reset in case of break or break2 event.
- the OSS1 bit in the TIMx_BDTR register defines whether the timer controls the output in inactive state or releases the control to the GPIO controller (typically to have it in Hi-Z mode)
- the OISx and OISxN bits in the TIMx_CR2 register which are setting the output shut-down level, either active or inactive. The OCx and OCxN outputs cannot be set both to active level at a given time, whatever the OISx and OISxN values.
Refer to [Table 115: Output control bits for complementary OCx and OCxN channels with break feature on page 589](#) for more details.

When exiting from reset, the break circuit is disabled and the MOE bit is low. You can enable the break functions by setting the BKE and BKE2 bits in the TIMx_BDTR register. The break input polarities can be selected by configuring the BKP and BKP2 bits in the same register. BKE_x and BKP_x can be modified at the same time. When the BKE_x and BKP_x bits are written, a delay of 1 APB clock cycle is applied before the writing is effective. Consequently, it is necessary to wait 1 APB clock period to correctly read back the bit after the write operation.

Because MOE falling edge can be asynchronous, a resynchronization circuit has been inserted between the actual signal (acting on the outputs) and the synchronous control bit (accessed in the TIMx_BDTR register). It results in some delays between the asynchronous and the synchronous signals. In particular, if 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.

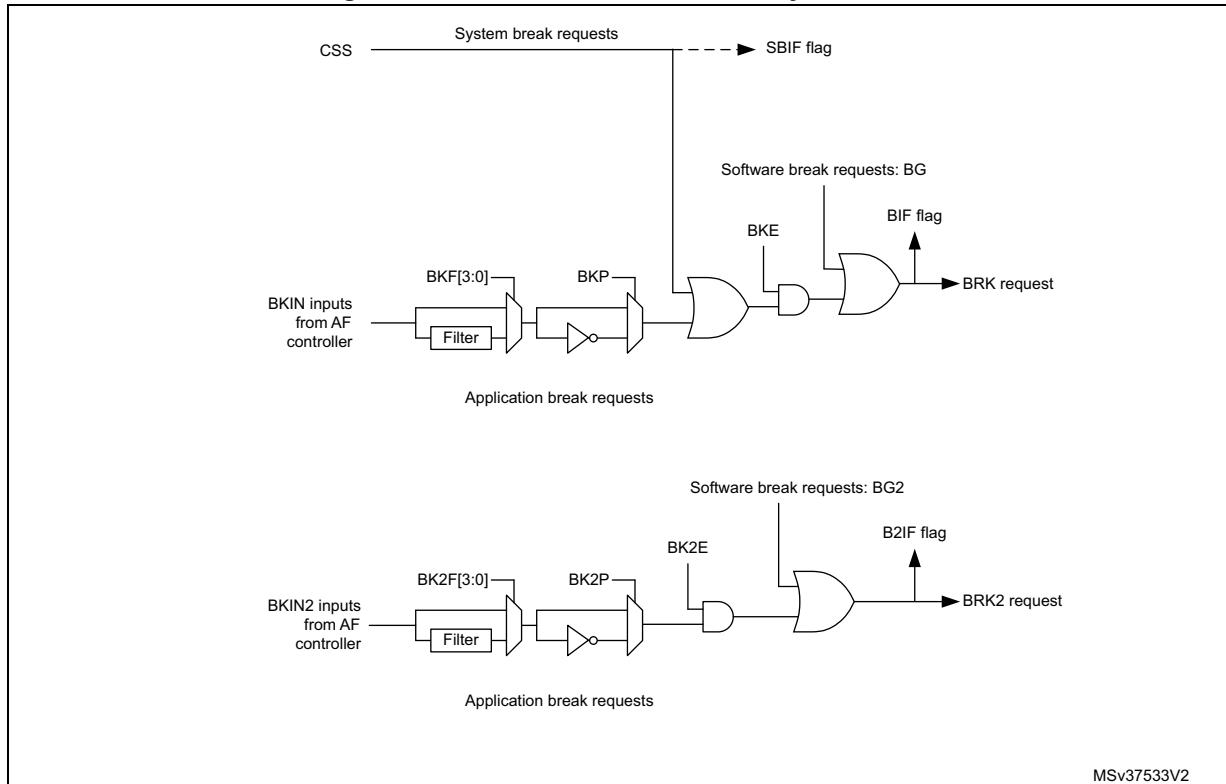
The break (BRK) event can be generated by two sources of events ORed together:

- An external source connected to one of the BKIN pin (as per selection done in the AFIO controller)
- An internal source: clock failure event generated by the CSS detector

The break2 (BRK2) can be generated by an external source connected to one of the BKIN2 pin (as per selection done in the AFIO controller).

Break events can also be generated by software using BG and B2G bits in the TIMx_EGR register. The software break generation using BG and BG2 is active whatever the BKE and BKE2 enable bits values.

Figure 176. Break and Break2 circuitry overview



MSV37533V2

Note: An asynchronous (clockless) operation is only guaranteed when the programmable filter is disabled. If it is enabled, a fail safe clock mode (for example by using the internal PLL and/or the CSS) must be used to guarantee that break events are handled.

When one of the breaks occurs (selected level on one of the break inputs):

- The MOE bit is cleared asynchronously, putting the outputs in inactive state, idle state or even releasing the control to the GPIO controller (selected by the OSS1 bit). This feature is enabled even if the MCU oscillator is off.
- Each output channel is driven with the level programmed in the OISx bit in the TIMx_CR2 register as soon as MOE=0. If OSS1=0, the timer releases the output control (taken over by the GPIO controller), otherwise the enable output remains high.
- When complementary outputs are used:
 - The outputs are first put in inactive state (depending on the polarity). This is done asynchronously so that it works even if no clock is provided to the timer.
 - If the timer clock is still present, 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 slightly longer than usual (around 2 ck_tim clock cycles).

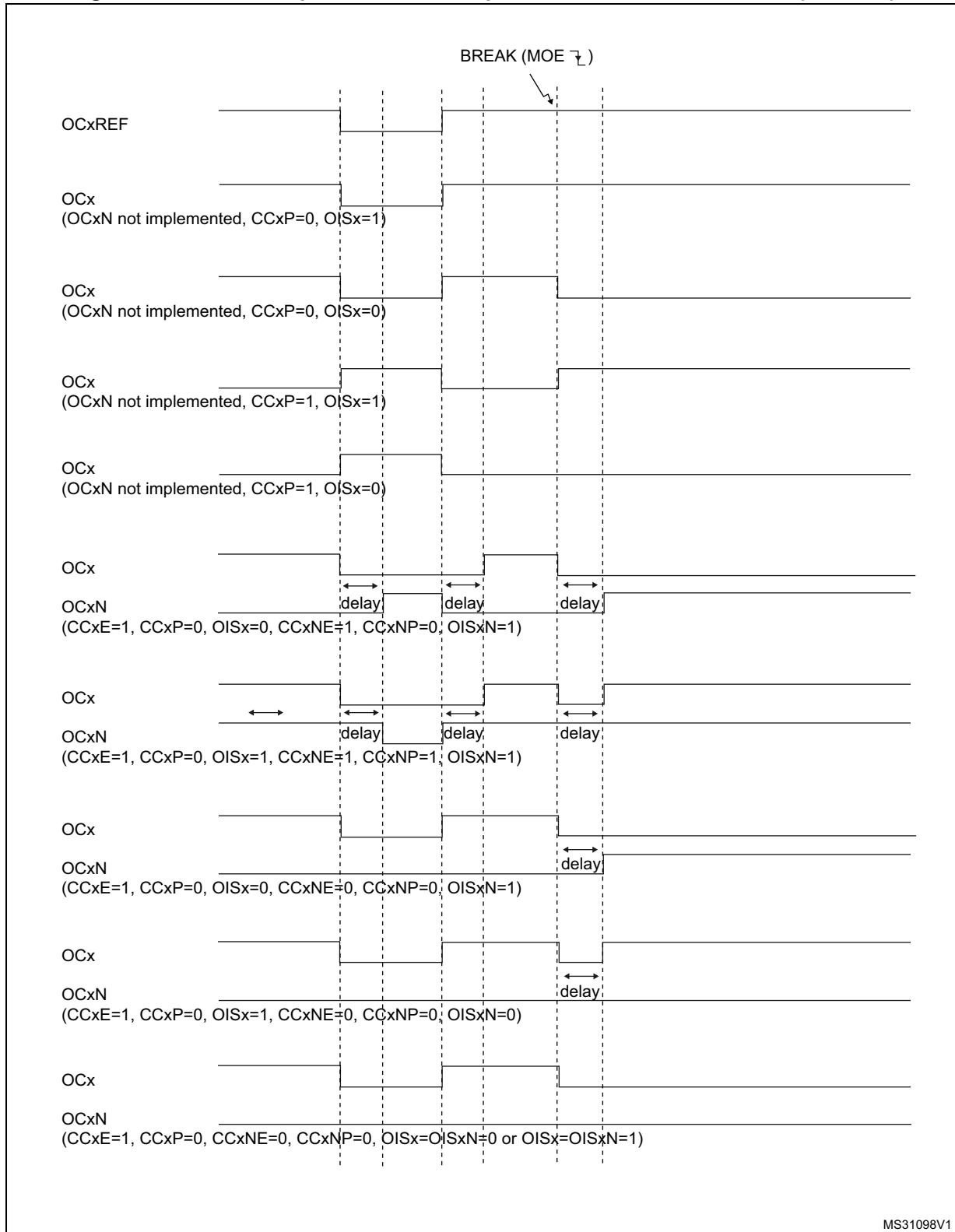
- If OSS1=0, the timer releases the output control (taken over by the GPIO controller which forces a Hi-Z state), otherwise the enable outputs remain or become high as soon as one of the CCxE or CCxNE bits is high.
- The break status flag (SBIF, BIF and B2IF bits in the TIMx_SR register) is set. An interrupt is generated if the BIE bit in the TIMx_DIER register is set. A DMA request can be sent if the BDE bit in the TIMx_DIER register is set.
- If the AOE bit in the TIMx_BDTR register is set, the MOE bit is automatically set again at the next update event (UEV). As an example, this can be used to perform a regulation. Otherwise, MOE remains low until the application sets it to ‘1’ again. In this case, it can be used for security and you can connect the break input to an alarm from power drivers, thermal sensors or any security components.

Note: *The break inputs are active 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 and B2IF cannot be cleared.*

In addition to the break input and the output management, a write protection has been implemented inside the break circuit to safeguard the application. It allows to freeze the configuration of several parameters (dead-time duration, OCx/OCxN polarities and state when disabled, OCxM configurations, break enable and polarity). The application can choose from 3 levels of protection selected by the LOCK bits in the TIMx_BDTR register. Refer to [Section 18.4.18: TIMx break and dead-time register \(TIMx_BDTR\)\(x = 1, 8\)](#). The LOCK bits can be written only once after an MCU reset.

Figure 177 shows an example of behavior of the outputs in response to a break.

Figure 177. Various output behavior in response to a break event on BRK (OSSI = 1)



MS31098V1

The two break inputs have different behaviors on timer outputs:

- The BRK input can either disable (inactive state) or force the PWM outputs to a predefined safe state.
- BRK2 can only disable (inactive state) the PWM outputs.

The BRK has a higher priority than BRK2 input, as described in [Table 112](#).

Note: BRK2 must only be used with OSSR = OSSI = 1.

Table 112. Behavior of timer outputs versus BRK/BRK2 inputs

BRK	BRK2	Timer outputs state	Typical use case	
			OCxN output (low side switches)	OCx output (high side switches)
Active	X	<ul style="list-style-type: none"> – Inactive then forced output state (after a deadtime) – Outputs disabled if OSSI = 0 (control taken over by GPIO logic) 	ON after deadtime insertion	OFF
Inactive	Active	Inactive	OFF	OFF

[Figure 178](#) gives an example of OCx and OCxN output behavior in case of active signals on BRK and BRK2 inputs. In this case, both outputs have active high polarities (CCxP = CCxNP = 0 in TIMx_CCER register).

Figure 178. PWM output state following BRK and BRK2 pins assertion (OSSI=1)

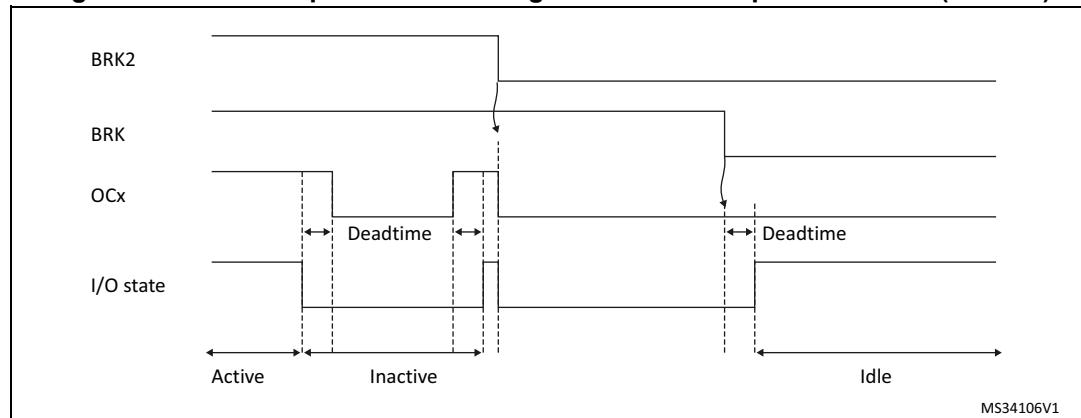
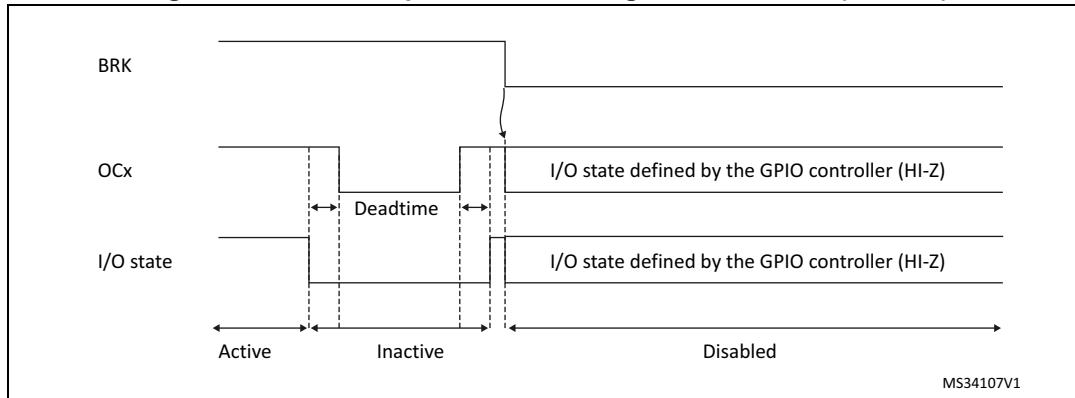


Figure 179. PWM output state following BRK assertion (OSSI=0)

18.3.17 Clearing the OCxREF signal on an external event

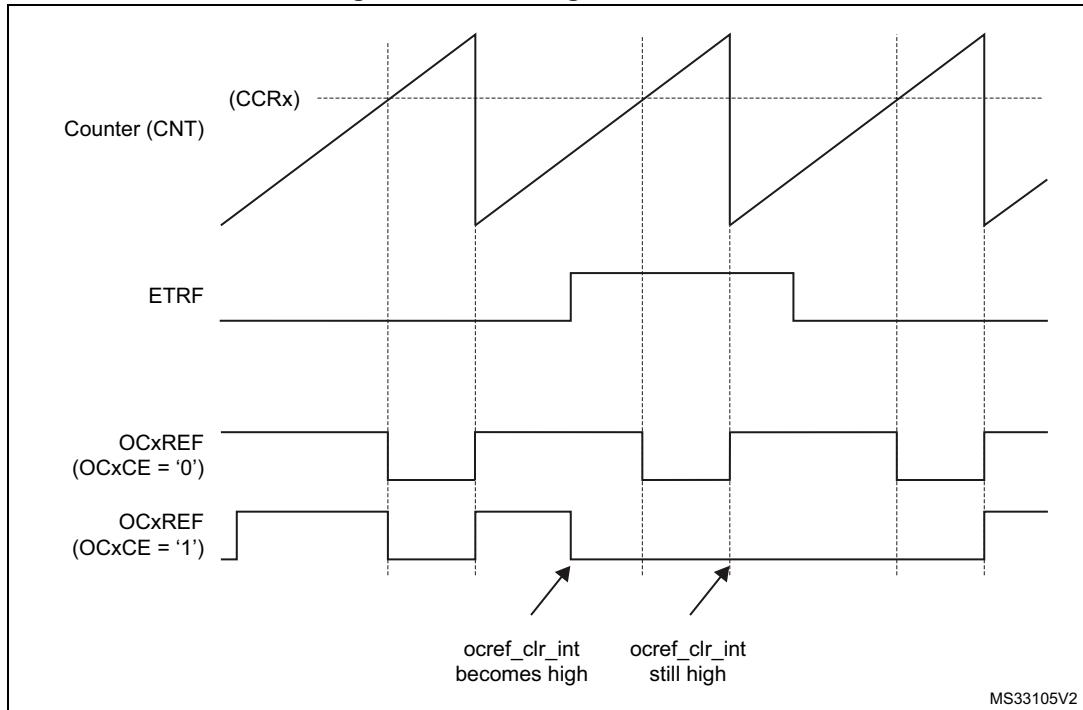
The OCxREF signal for a given channel can be driven low by applying a high level to the ETRF input (OCxCE enable bit of the corresponding TIMx_CCMRx register set to '1'). The OCxREF signal remains low until the next update event, UEV, occurs.

When ETRF is chosen, ETR must be configured as follows:

1. The External Trigger Prescaler should be kept off: bits ETPS[1:0] of the TIMx_SMCR register set to '00'.
2. The external clock mode 2 must be disabled: bit ECE of the TIMx_SMCR register set to '0'.
3. The External Trigger Polarity (ETP) and the External Trigger Filter (ETF) can be configured according to the user needs.

Figure 180 shows the behavior of the OCxREF signal when the ETRF Input becomes High, for both values of the enable bit OCxCE. In this example, the timer TIMx is programmed in PWM mode.

Figure 180. Clearing TIMx OCxREF



Note:

In case of a PWM with a 100% duty cycle (if CCRx>ARR), then OCxREF is enabled again at the next counter overflow.

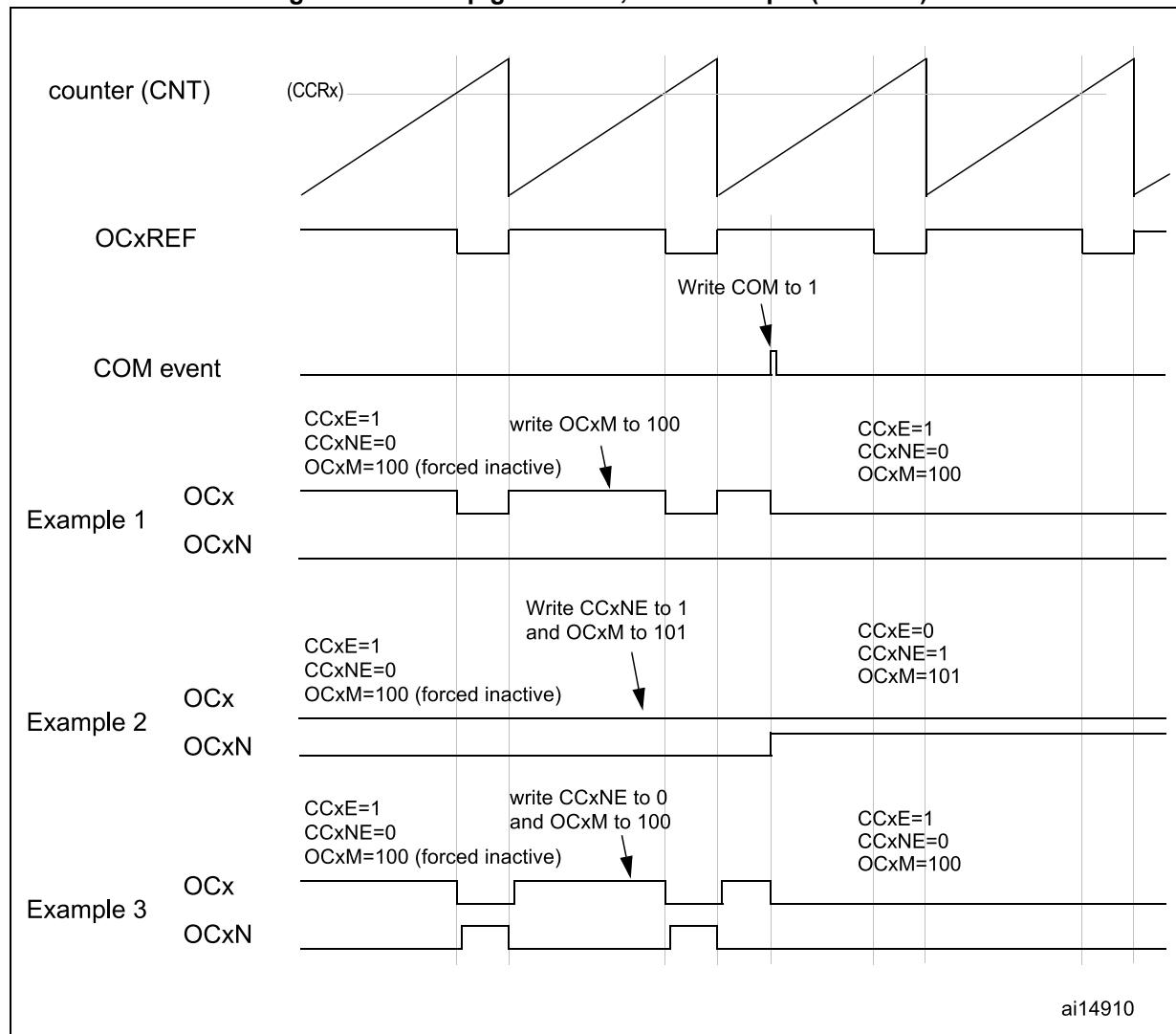
18.3.18 6-step PWM generation

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 TIMx_EGR register or by hardware (on TRGI rising edge).

A flag is set when the COM event occurs (COMIF bit in the TIMx_SR register), which can generate an interrupt (if the COMIE bit is set in the TIMx_DIER register) or a DMA request (if the COMDE bit is set in the TIMx_DIER register).

The [Figure 181](#) describes the behavior of the OCx and OCxN outputs when a COM event occurs, in 3 different examples of programmed configurations.

Figure 181. 6-step generation, COM example (OSSR=1)



ai14910

18.3.19 One-pulse mode

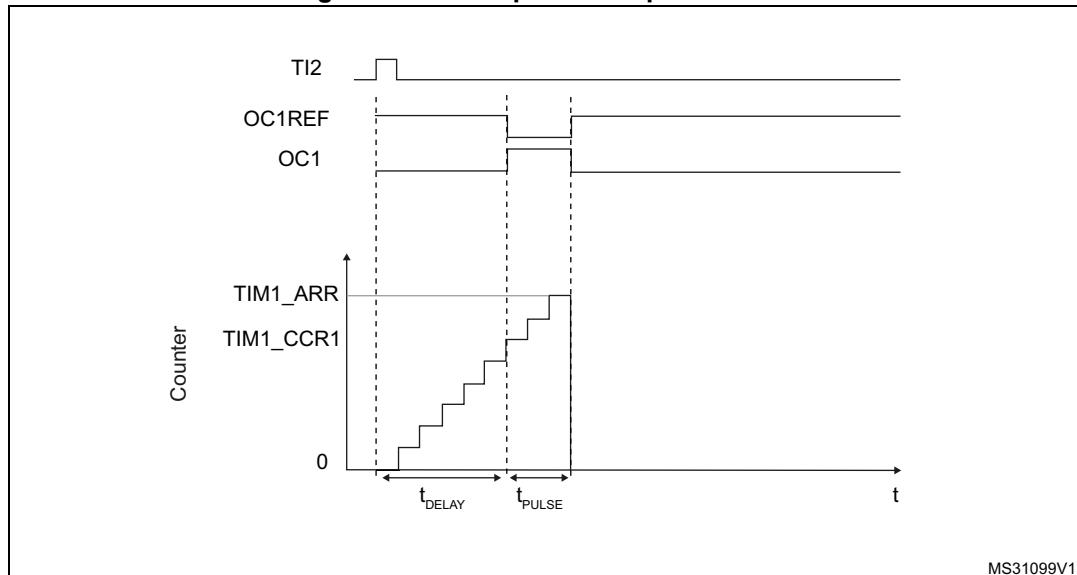
One-pulse mode (OPM) is a particular case of the previous modes. It allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length after a programmable delay.

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 < CCRx \leq ARR (in particular, 0 < CCRx)
- In downcounting: CNT > CCRx

Figure 182. Example of one pulse mode.



MS31099V1

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:

1. Map TI2FP2 to TI2 by writing CC2S='01' in the TIMx_CCMR1 register.
2. TI2FP2 must detect a rising edge, write CC2P='0' and CC2NP='0' in the TIMx_CCER register.
3. Configure TI2FP2 as trigger for the slave mode controller (TRGI) by writing TS=110 in the TIMx_SMCR register.
4. 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 (Single mode), 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). When OPM bit in the TIMx_CR1 register is set to '0', so the Repetitive Mode is selected.

Particular case: OCx fast enable:

In One-pulse mode, the edge detection on TIx input set the CEN bit which enables the counter. Then the comparison between the counter and the compare value makes the output toggle. But several clock cycles are needed for these operations and it limits the minimum delay t_{DELAY} min we can get.

If you want to output a waveform with the minimum delay, you can set the OCxFE bit in the TIMx_CCMRx register. Then OCxRef (and OCx) are forced in response to the stimulus, without taking in account the comparison. Its new level is the same as if a compare match had occurred. OCxFE acts only if the channel is configured in PWM1 or PWM2 mode.

18.3.20 Retriggerable one pulse mode (OPM)

This mode allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length, but with the following differences with Non-retriggerable one pulse mode described in [Section 18.3.19](#):

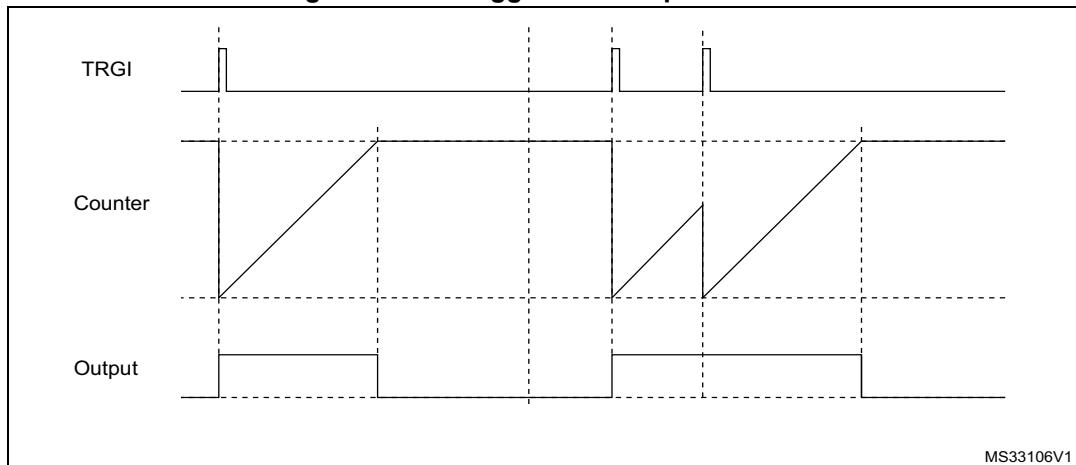
- The pulse starts as soon as the trigger occurs (no programmable delay)
- The pulse is extended if a new trigger occurs before the previous one is completed

The timer must be in Slave mode, with the bits SMS[3:0] = '1000' (Combined Reset + trigger mode) in the TIMx_SMCR register, and the OCxM[3:0] bits set to '1000' or '1001' for Retriggerable OPM mode 1 or 2.

If the timer is configured in Up-counting mode, the corresponding CCRx must be set to 0 (the ARR register sets the pulse length). If the timer is configured in Down-counting mode, CCRx must be above or equal to ARR.

Note: The OCxM[3:0] and SMS[3:0] bit fields are split into two parts for compatibility reasons, the most significant bit are not contiguous with the 3 least significant ones.

This mode must not be used with center-aligned PWM modes. It is mandatory to have CMS[1:0] = 00 in TIMx_CR1.

Figure 183. Retriggerable one pulse mode

18.3.21 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. CC1NP and CC2NP must be kept low.

The two inputs TI1 and TI2 are used to interface to an quadrature encoder. Refer to [Table 113](#). 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, 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.

Note:

The prescaler must be set to zero when encoder mode is enabled

In this mode, the counter is modified automatically following the speed and the direction of the quadrature 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 113. 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

A quadrature 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 184](#) gives an example of counter operation, showing count signal generation and direction control. It also shows how input jitter is compensated where both edges are selected. This might occur if the sensor is positioned near to one of the switching points. For this example we assume that the configuration is the following:

- CC1S='01' (TIMx_CCMR1 register, TI1FP1 mapped on TI1).
- CC2S='01' (TIMx_CCMR2 register, TI1FP2 mapped on TI2).
- CC1P='0' and CC1NP='0' (TIMx_CCER register, TI1FP1 non-inverted, TI1FP1=TI1).
- CC2P='0' and CC2NP='0' (TIMx_CCER register, TI1FP2 non-inverted, TI1FP2= TI2).
- SMS='011' (TIMx_SMCR register, both inputs are active on both rising and falling edges).
- CEN='1' (TIMx_CR1 register, Counter enabled).

Figure 184. Example of counter operation in encoder interface mode.

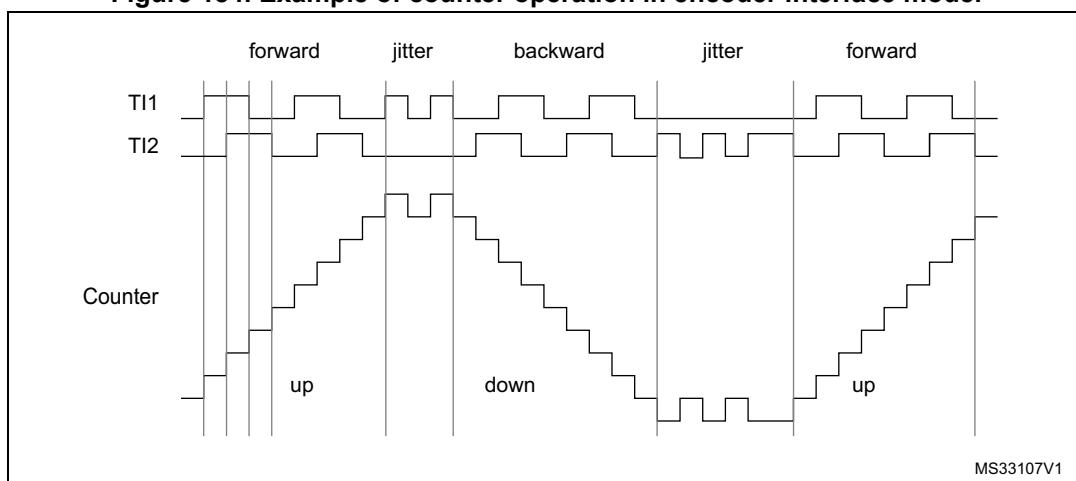
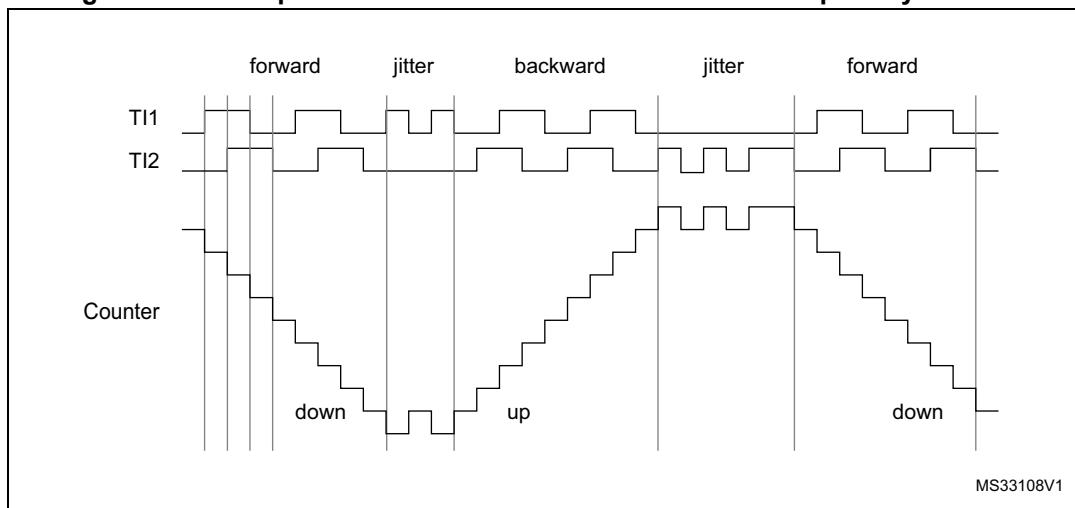


Figure 185 gives an example of counter behavior when TI1FP1 polarity is inverted (same configuration as above except CC1P='1').

Figure 185. 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.

The IUFREMAP bit in the TIMx_CR1 register forces a continuous copy of the update interrupt flag (UIF) into the timer counter register's bit 31 (TIMxCNT[31]). This allows both the counter value and a potential roll-over condition signaled by the UIFCPY flag to be read in an atomic way. It eases the calculation of angular speed by avoiding race conditions caused, for instance, by a processing shared between a background task (counter reading) and an interrupt (update interrupt).

There is no latency between the UIF and UIFCPY flag assertions.

In 32-bit timer implementations, when the IUFREMAP bit is set, bit 31 of the counter is overwritten by the UIFCPY flag upon read access (the counter's most significant bit is only accessible in write mode).

18.3.22 UIF bit remapping

The IUFREMAP bit in the TIMx_CR1 register forces a continuous copy of the Update Interrupt Flag UIF into the timer counter register's bit 31 (TIMxCNT[31]). This allows both the counter value and a potential roll-over condition signaled by the UIFCPY flag to be read in an atomic way. In particular cases, it can ease the calculations by avoiding race conditions, caused for instance by a processing shared between a background task (counter reading) and an interrupt (Update Interrupt).

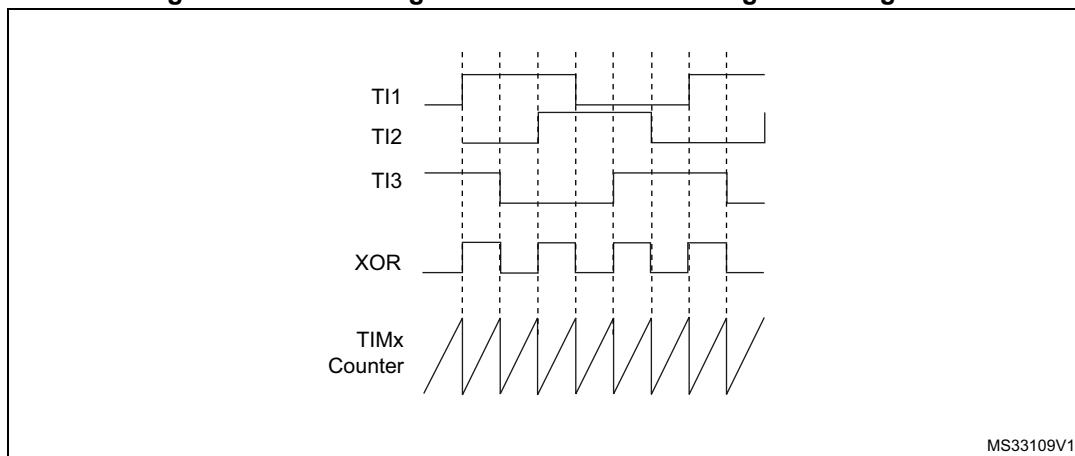
There is no latency between the UIF and UIFCPY flags assertion.

18.3.23 Timer input XOR function

The TI1S bit in the TIMx_CR2 register, allows the input filter of channel 1 to be connected to the output of an XOR gate, combining the three input pins TIMx_CH1, TIMx_CH2 and TIMx_CH3.

The XOR output can be used with all the timer input functions such as trigger or input capture. It is convenient to measure the interval between edges on two input signals, as per [Figure 186](#) below.

Figure 186. Measuring time interval between edges on 3 signals



18.3.24 Interfacing with Hall sensors

This is done using the advanced-control timers (TIM1 or TIM8) to generate PWM signals to drive the motor and another timer TIMx (TIM2, TIM3, TIM4) referred to as “interfacing timer” in [Figure 187](#). 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 161: Capture/compare channel \(example: channel 1 input stage\) on page 532](#)). 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 or TIM8) (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 or TIM8) through the TRGO output.

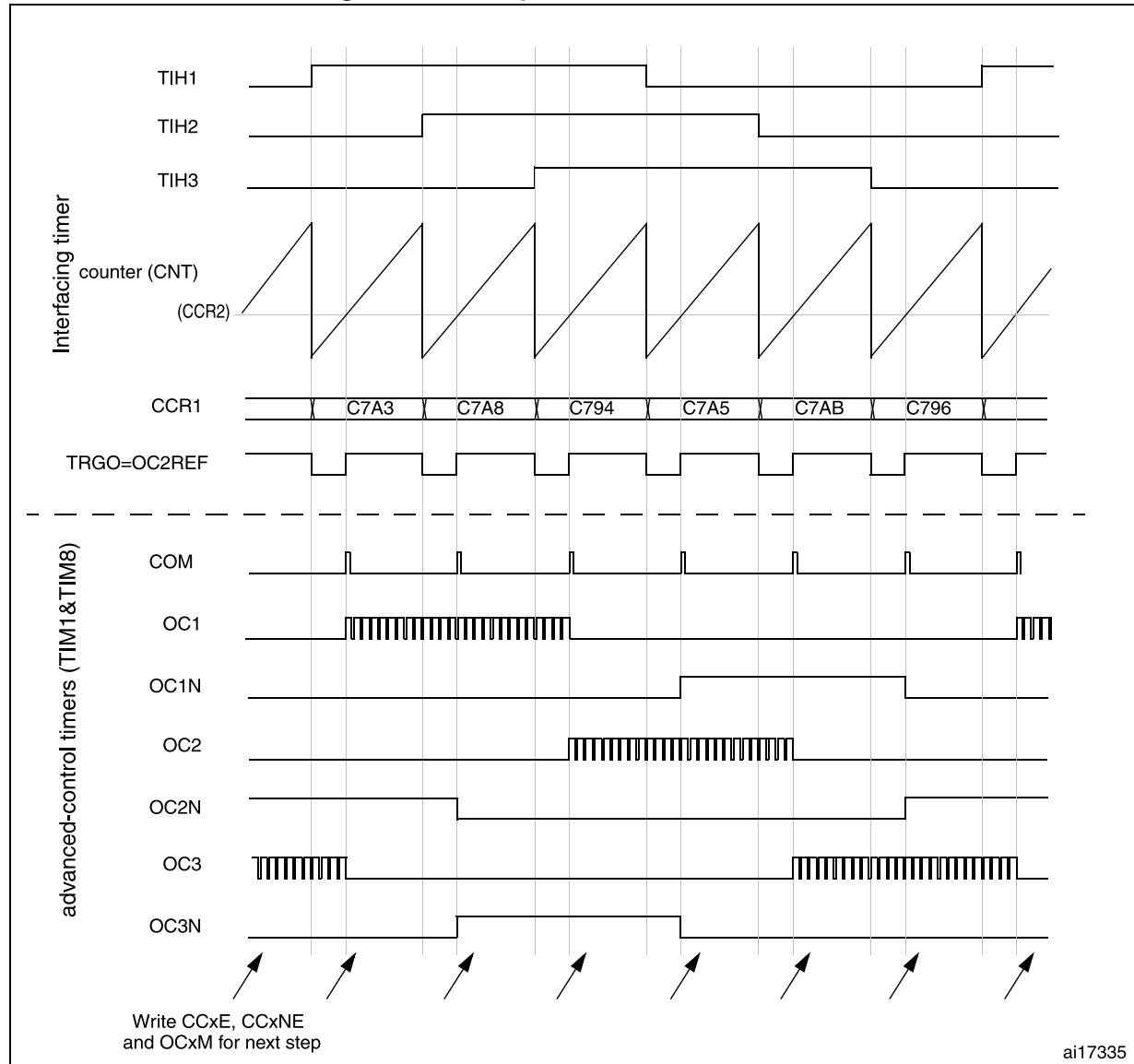
Example: you want to change the PWM configuration of your advanced-control timer TIM1 after a programmed delay each time a change occurs on the Hall inputs connected to one of the TIMx timers.

- Configure 3 timer inputs 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 TIMx_CCMR1 register,
- Select OC2REF as trigger output on TRGO: write the MMS bits in the TIMx_CR2 register to '101',

In the advanced-control timer TIM1, the right ITR input must be selected as trigger input, the timer is programmed to generate PWM signals, the capture/compare control signals are preloaded (CCPC=1 in the TIMx_CR2 register) and the COM event is controlled by the trigger input (CCUS=1 in the TIMx_CR2 register). The PWM control bits (CCxE, OCxM) are written after a COM event for the next step (this can be done in an interrupt subroutine generated by the rising edge of OC2REF).

The [Figure 187](#) describes this example.

Figure 187. Example of Hall sensor interface



18.3.25 Timer synchronization

The TIMx timers are linked together internally for timer synchronization or chaining. They can be synchronized 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.

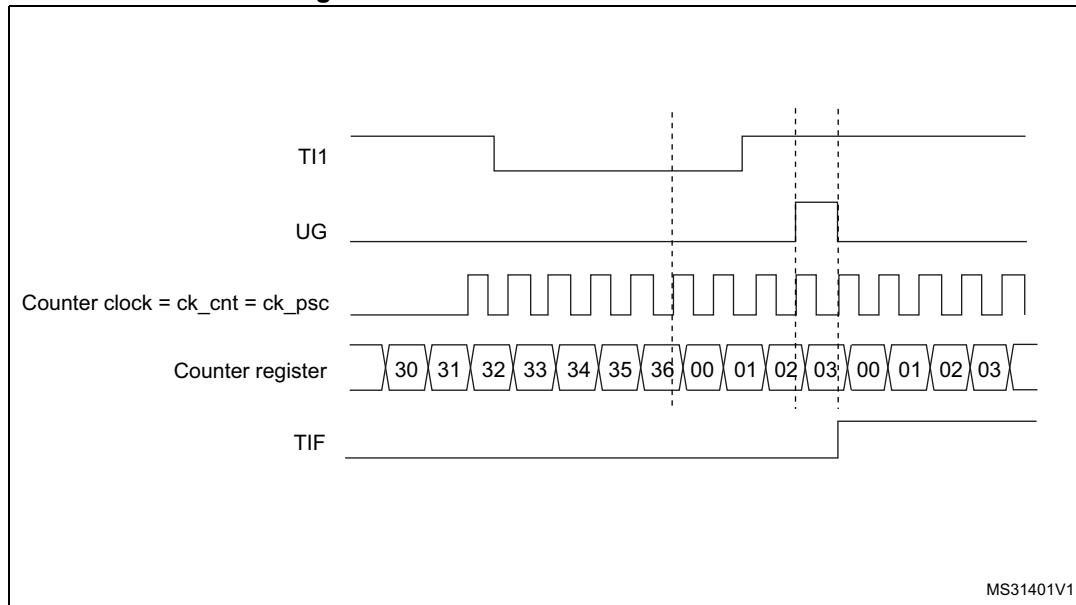
In the following example, the upcounter is cleared in response to a rising edge on TI1 input:

- 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 and CC1NP='0' in TIMx_CCER register to validate the polarity (and detect rising edges only).
- Configure the timer in reset mode by writing SMS=100 in TIMx_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx_SMCR register.
- Start the counter by writing CEN=1 in the TIMx_CR1 register.

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 188. Control circuit in reset mode



Slave mode: Gated mode

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

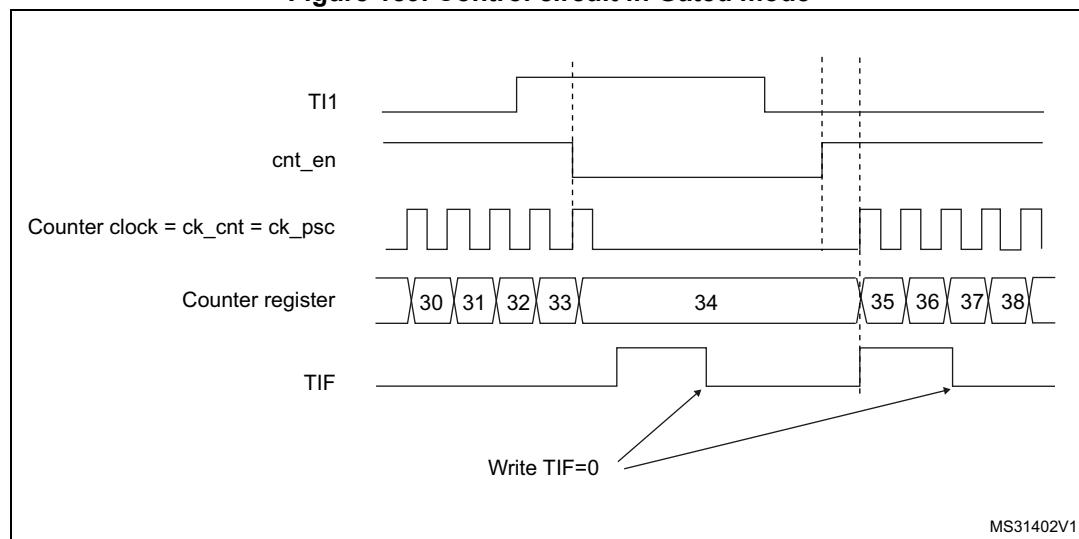
In the following example, the upcounter counts only when TI1 input is low:

- 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 and CC1NP='0' in TIMx_CCER register to validate the polarity (and detect low level only).
- Configure the timer in gated mode by writing SMS=101 in TIMx_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx_SMCR register.
- Enable the counter by writing CEN=1 in the TIMx_CR1 register (in gated mode, the counter doesn't start if CEN=0, whatever is the trigger input level).

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 189. 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 TIMx_CCMR1 register.

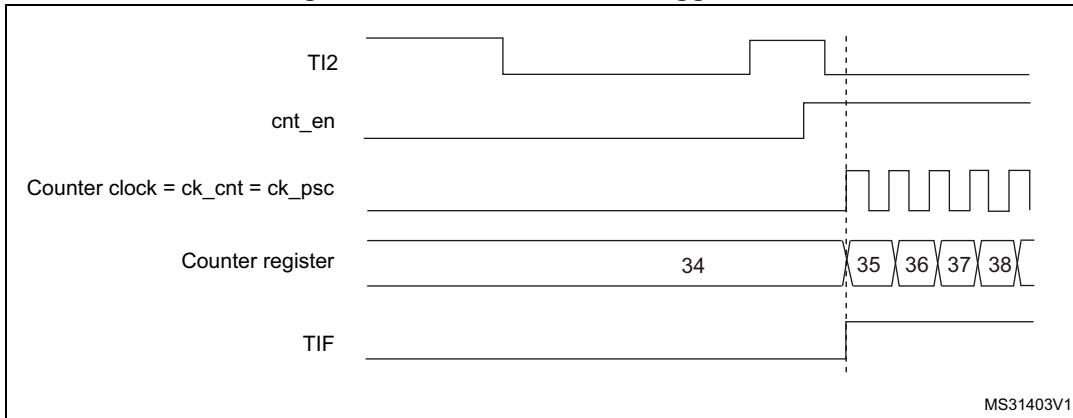
Write CC2P=1 and CC2NP=0 in TIMx_CCER register to validate the polarity (and detect low level only).

- Configure the timer in trigger mode by writing SMS=110 in TIMx_SMCR register. Select TI2 as the input source by writing TS=110 in TIMx_SMCR register.

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 190. Control circuit in trigger mode



Slave mode: Combined reset + trigger mode

In this case, a rising edge of the selected trigger input (TRGI) reinitializes the counter, generates an update of the registers, and starts the counter.

This mode is used for one-pulse mode.

Slave mode: external clock mode 2 + trigger mode

The external clock mode 2 can be used in addition to another slave mode (except external clock mode 1 and encoder mode). In this case, the ETR signal is used as external clock input, and another input can be selected as trigger input (in reset mode, gated mode or trigger mode). It is recommended not to select ETR as TRGI through the TS bits of TIMx_SMCR register.

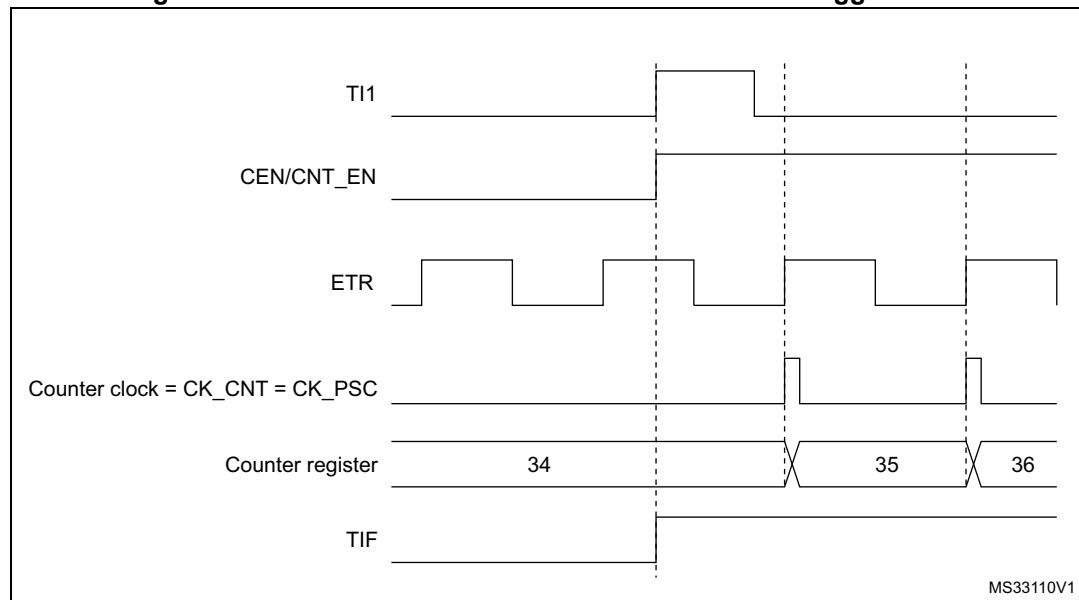
In the following example, the upcounter is incremented at each rising edge of the ETR signal as soon as a rising edge of TI1 occurs:

1. Configure the external trigger input circuit by programming the TIMx_SMCR register as follows:
 - ETF = 0000: no filter
 - ETPS=00: prescaler disabled
 - ETP=0: detection of rising edges on ETR and ECE=1 to enable the external clock mode 2.
2. Configure the channel 1 as follows, to detect rising edges on TI:
 - IC1F=0000: no filter.
 - The capture prescaler is not used for triggering and does not need to be configured.
 - CC1S=01in TIMx_CCMR1 register to select only the input capture source
 - CC1P=0 and CC1NP='0' in TIMx_CCER register to validate the polarity (and detect rising edge only).
3. Configure the timer in trigger mode by writing SMS=110 in TIMx_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx_SMCR register.

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 191. Control circuit in external clock mode 2 + trigger mode



Note:

The clock of the slave peripherals (timer, ADC, ...) receiving the TRGO or the TRGO2 signals must be enabled prior to receive events from the master timer, and the clock frequency (prescaler) must not be changed on-the-fly while triggers are received from the master timer.

18.3.26 ADC synchronization

The timer can generate an ADC triggering event with various internal signals, such as reset, enable or compare events. It is also possible to generate a pulse issued by internal edge detectors, such as:

- Rising and falling edges of OC4ref
- Rising edge on OC5ref or falling edge on OC6ref

The triggers are issued on the TRGO2 internal line which is redirected to the ADC. There is a total of 16 possible events, which can be selected using the MMS2[3:0] bits in the TIMx_CR2 register.

An example of an application for 3-phase motor drives is given in [Figure 172 on page 544](#).

Note: *The clock of the slave peripherals (timer, ADC, ...) receiving the TRGO or the TRGO2 signals must be enabled prior to receive events from the master timer, and the clock frequency (prescaler) must not be changed on-the-fly while triggers are received from the master timer.*

Note: *The clock of the ADC must be enabled prior to receive events from the master timer, and must not be changed on-the-fly while triggers are received from the timer.*

18.3.27 DMA burst mode

The TIMx timers have the capability to generate multiple DMA requests upon a single event. The main purpose is to be able to re-program part of the timer multiple times without software overhead, but it can also be used to read several registers in a row, at regular intervals.

The DMA controller destination is unique and must point to the virtual register TIMx_DMAR. On a given timer event, the timer launches a sequence of DMA requests (burst). Each write into the TIMx_DMAR register is actually redirected to one of the timer registers.

The DBL[4:0] bits in the TIMx_DCR register set the DMA burst length. The timer recognizes a burst transfer when a read or a write access is done to the TIMx_DMAR address), i.e. the number of transfers (either in half-words or in bytes).

The DBA[4:0] bits in the TIMx_DCR registers define the DMA base address for DMA transfers (when read/write access are done through the TIMx_DMAR address). DBA is defined as an offset starting from the address of the TIMx_CR1 register:

Example:

00000: TIMx_CR1

00001: TIMx_CR2

00010: TIMx_SMCR

As an example, the timer DMA burst feature is used to update the contents of the CCRx registers ($x = 2, 3, 4$) upon an update event, with the DMA transferring half words into the CCRx registers.

This is done in the following steps:

1. Configure the corresponding DMA channel as follows:
 - DMA channel peripheral address is the DMAR register address
 - DMA channel memory address is the address of the buffer in the RAM containing the data to be transferred by DMA into CCRx registers.
 - Number of data to transfer = 3 (See note below).
 - Circular mode disabled.
2. Configure the DCR register by configuring the DBA and DBL bit fields as follows:
DBL = 3 transfers, DBA = 0xE.
3. Enable the TIMx update DMA request (set the UDE bit in the DIER register).
4. Enable TIMx
5. Enable the DMA channel

This example is for the case where every CCRx register to be updated once. If every CCRx register is to be updated twice for example, the number of data to transfer should be 6. Let's take the example of a buffer in the RAM containing data1, data2, data3, data4, data5 and data6. The data is transferred to the CCRx registers as follows: on the first update DMA request, data1 is transferred to CCR2, data2 is transferred to CCR3, data3 is transferred to CCR4 and on the second update DMA request, data4 is transferred to CCR2, data5 is transferred to CCR3 and data6 is transferred to CCR4.

Note: A null value can be written to the reserved registers.

18.3.28 Debug mode

When the microcontroller enters debug mode (Cortex[®]-M7 core halted), the TIMx counter either continues to work normally or stops, depending on DBG_TIMx_STOP configuration bit in DBG module.

For safety purposes, when the counter is stopped (DBG_TIMx_STOP = 1), the outputs are disabled (as if the MOE bit was reset). The outputs can either be forced to an inactive state (OSSI bit = 1), or have their control taken over by the GPIO controller (OSSI bit = 0), typically to force a Hi-Z.

For more details, refer to [Section 40.16.2: Debug support for timers, watchdog, bxCAN and I2C](#).

For safety purposes, when the counter is stopped (DBG_TIMx_STOP = 1), the outputs are disabled (as if the MOE bit was reset). The outputs can either be forced to an inactive state (OSSI bit = 1), or have their control taken over by the GPIO controller (OSSI bit = 0) to force them to Hi-Z.

18.4 TIM1/TIM8 registers

Refer to for a list of abbreviations used in register descriptions.

18.4.1 TIMx control register 1 (TIMx_CR1)(x = 1, 8)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	UIFRE MAP	Res.	CKD[1:0]		ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN
				rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:12 Reserved, must be kept at reset value.

Bit 11 **UIFREMAP**: UIF status bit remapping

- 0: No remapping. UIF status bit is not copied to TIMx_CNT register bit 31.
- 1: Remapping enabled. UIF status bit is copied to TIMx_CNT register bit 31.

Bit 10 Reserved, must be kept at reset value.

Bits 9:8 **CKD[1:0]**: Clock division

This bit-field indicates the division ratio between the timer clock (CK_INT) frequency and the dead-time and sampling clock (t_{DTS}) used by the dead-time generators and the digital filters (ETR, TIx),

00: $t_{DTS}=t_{CK_INT}$

01: $t_{DTS}=2*t_{CK_INT}$

10: $t_{DTS}=4*t_{CK_INT}$

11: Reserved, do not program this value

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

- 0: TIMx_ARR register is not buffered
- 1: TIMx_ARR register is buffered

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

00: Edge-aligned mode. The counter counts up or down depending on the direction bit (DIR).

01: Center-aligned mode 1. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set only when the counter is counting down.

10: Center-aligned mode 2. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set only when the counter is counting up.

11: Center-aligned mode 3. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set both when the counter is counting up or down.

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

Bit 4 **DIR**: Direction

- 0: Counter used as upcounter
- 1: Counter used as downcounter

Note: This bit is read only when the timer is configured in Center-aligned mode or Encoder mode.

Bit 3 **OPM**: One pulse mode

- 0: Counter is not stopped at update event
 1: Counter stops counting at the next update event (clearing the bit CEN)

Bit 2 **URS**: Update request source

This bit is set and cleared by software to select the UEV event sources.

- 0: Any of the following events generate an update interrupt or DMA request if enabled.
 These events can be:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

1: Only counter overflow/underflow generates an update interrupt or DMA request if enabled.

Bit 1 **UDIS**: Update disable

This bit is set and cleared by software to enable/disable UEV event generation.

0: UEV enabled. The Update (UEV) event is generated by one of the following events:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

Buffered registers are then loaded with their preload values.

1: UEV disabled. The Update event is not generated, shadow registers keep their value (ARR, PSC, CCRx). However the counter and the prescaler are reinitialized if the UG bit is set or if a hardware reset is received from the slave mode controller.

Bit 0 **CEN**: Counter enable

0: Counter disabled

1: Counter enabled

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

18.4.2 TIMx control register 2 (TIMx_CR2)(x = 1, 8)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	MMS2[3:0]				Res.	OIS6	Res.	OIS5	
								rw	rw	rw	rw		rw		rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1	TI1S	MMS[2:0]				CCDS	CCUS	Res.	CCPC
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

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

Bits 23:20 **MMS2[3:0]**: Master mode selection 2

These bits allow the information to be sent to ADC for synchronization (TRGO2) to be selected. The combination is as follows:

0000: **Reset** - the UG bit from the TIMx_EGR register is used as trigger output (TRGO2). If the reset is generated by the trigger input (slave mode controller configured in reset mode), the signal on TRGO2 is delayed compared to the actual reset.

0001: **Enable** - the Counter Enable signal CNT_EN is used as trigger output (TRGO2). 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 the CEN control bit and the trigger input when configured in Gated mode. When the Counter Enable signal is controlled by the trigger input, there is a delay on TRGO2, except if the Master/Slave mode is selected (see the MSM bit description in TIMx_SMCR register).

0010: **Update** - the update event is selected as trigger output (TRGO2). For instance, a master timer can then be used as a prescaler for a slave timer.

0011: **Compare pulse** - the trigger output sends a positive pulse when the CC1IF flag is to be set (even if it was already high), as soon as a capture or compare match occurs (TRGO2).

0100: **Compare** - OC1REF signal is used as trigger output (TRGO2)

0101: **Compare** - OC2REF signal is used as trigger output (TRGO2)

0110: **Compare** - OC3REF signal is used as trigger output (TRGO2)

0111: **Compare** - OC4REF signal is used as trigger output (TRGO2)

1000: **Compare** - OC5REF signal is used as trigger output (TRGO2)

1001: **Compare** - OC6REF signal is used as trigger output (TRGO2)

1010: **Compare Pulse** - OC4REF rising or falling edges generate pulses on TRGO2

1011: **Compare Pulse** - OC6REF rising or falling edges generate pulses on TRGO2

1100: **Compare Pulse** - OC4REF or OC6REF rising edges generate pulses on TRGO2

1101: **Compare Pulse** - OC4REF rising or OC6REF falling edges generate pulses on TRGO2

1110: **Compare Pulse** - OC5REF or OC6REF rising edges generate pulses on TRGO2

1111: **Compare Pulse** - OC5REF rising or OC6REF falling edges generate pulses on TRGO2

Note: The clock of the slave timer or ADC must be enabled prior to receive events from the master timer, and must not be changed on-the-fly while triggers are received from the master timer.

Bit 19 Reserved, must be kept at reset value.

Bit 18 **OIS6**: Output Idle state 6 (OC6 output)

Refer to OIS1 bit

Bit 17 Reserved, must be kept at reset value.

Bit 16 **OIS5**: Output Idle state 5 (OC5 output)

Refer to OIS1 bit

Bit 15 Reserved, must be kept at reset value.

Bit 14 **OIS4**: Output Idle state 4 (OC4 output)

Refer to OIS1 bit

Bit 13 **OIS3N**: Output Idle state 3 (OC3N output)

Refer to OIS1N bit

Bit 12 **OIS3**: Output Idle state 3 (OC3 output)

Refer to OIS1 bit

Bit 11 **OIS2N**: Output Idle state 2 (OC2N output)

Refer to OIS1N bit

Bit 10 **OIS2**: Output Idle state 2 (OC2 output)

Refer to OIS1 bit

Bit 9 **OIS1N**: Output Idle state 1 (OC1N output)

0: OC1N=0 after a dead-time when MOE=0

1: OC1N=1 after a dead-time when MOE=0

Note: This bit can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx_BDTR register).

Bit 8 **OIS1**: Output Idle state 1 (OC1 output)

0: OC1=0 (after a dead-time if OC1N is implemented) when MOE=0

1: OC1=1 (after a dead-time if OC1N is implemented) when MOE=0

Note: This bit can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx_BDTR register).

Bit 7 **TI1S**: TI1 selection

0: The TIMx_CH1 pin is connected to TI1 input

1: The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

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

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

000: **Reset** - the UG bit from the TIMx_EGR register is used as trigger output (TRGO). If the reset is generated by the trigger input (slave mode controller configured in reset mode) then the signal on TRGO is delayed compared to the actual reset.

001: **Enable** - the Counter Enable signal CNT_EN is used as trigger output (TRGO). It is useful to start several timers at the same time or to control a window in which a slave timer is enable. The Counter Enable signal is generated by a logic OR between CEN control bit and the trigger input when configured in gated mode. When the Counter Enable signal is controlled by the trigger input, there is a delay on TRGO, except if the master/slave mode is selected (see the MSM bit description in TIMx_SMCR register).

010: **Update** - The update event is selected as trigger output (TRGO). For instance a master timer can then be used as a prescaler for a slave timer.

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

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

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

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

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

Note: The clock of the slave timer or ADC must be enabled prior to receive events from the master timer, and must not be changed on-the-fly while triggers are received from the master timer.

Bit 3 **CCDS**: Capture/compare DMA selection

0: CCx DMA request sent when CCx event occurs

1: CCx DMA requests sent when update event occurs

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

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

1: When capture/compare control bits are preloaded (CCPC=1), they are updated by setting the COMG bit or when an rising edge occurs on TRGI

Note: This bit acts only on channels that have a complementary output.

Bit 1 Reserved, must be kept at reset value.

Bit 0 **CCPC**: Capture/compare preloaded control

0: CCxE, CCxNE and OCxM bits are not preloaded

1: CCxE, CCxNE and OCxM bits are preloaded, after having been written, they are updated only when a commutation event (COM) occurs (COMG bit set or rising edge detected on TRGI, depending on the CCUS bit).

Note: This bit acts only on channels that have a complementary output.

18.4.3 TIMx slave mode control register (TIMx_SMCR)(x = 1, 8)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SMS[3]
															rw
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	rw

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

Bits 19:17 Reserved, must be kept at reset value.

Bit 15 **ETP**: External trigger polarity

This bit selects whether ETR or \overline{ETR} is used for trigger operations

0: ETR is non-inverted, active at high level or rising edge.

1: ETR is inverted, active at low level or falling edge.

Bit 14 **ECE**: External clock enable

This bit enables External clock mode 2.

0: External clock mode 2 disabled

1: External clock mode 2 enabled. The counter is clocked by any active edge on the ETRF signal.

Note: 1: Setting the ECE bit has the same effect as selecting external clock mode 1 with TRGI connected to ETRF (SMS=111 and TS=111).

2: It is possible to simultaneously use external clock mode 2 with the following slave modes: reset mode, gated mode and trigger mode. Nevertheless, TRGI must not be connected to ETRF in this case (TS bits must not be 111).

3: If external clock mode 1 and external clock mode 2 are enabled at the same time, the external clock input is ETRF.

Bits 13:12 **ETPS[1:0]**: External trigger prescaler

External trigger signal ETRP frequency must be at most 1/4 of TIMxCLK frequency. A prescaler can be enabled to reduce ETRP frequency. It is useful when inputting fast external clocks.

- 00: Prescaler OFF
- 01: ETRP frequency divided by 2
- 10: ETRP frequency divided by 4
- 11: ETRP frequency divided by 8

Bits 11:8 **ETF[3:0]**: External trigger filter

This bit-field then defines the frequency used to sample ETRP signal and the length of the digital filter applied to ETRP. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output:

- 0000: No filter, sampling is done at f_{DTS}
- 0001: $f_{SAMPLING} = f_{CK_INT}$, N=2
- 0010: $f_{SAMPLING} = f_{CK_INT}$, N=4
- 0011: $f_{SAMPLING} = f_{CK_INT}$, N=8
- 0100: $f_{SAMPLING} = f_{DTS}/2$, N=6
- 0101: $f_{SAMPLING} = f_{DTS}/2$, N=8
- 0110: $f_{SAMPLING} = f_{DTS}/4$, N=6
- 0111: $f_{SAMPLING} = f_{DTS}/4$, N=8
- 1000: $f_{SAMPLING} = f_{DTS}/8$, N=6
- 1001: $f_{SAMPLING} = f_{DTS}/8$, N=8
- 1010: $f_{SAMPLING} = f_{DTS}/16$, N=5
- 1011: $f_{SAMPLING} = f_{DTS}/16$, N=6
- 1100: $f_{SAMPLING} = f_{DTS}/16$, N=8
- 1101: $f_{SAMPLING} = f_{DTS}/32$, N=5
- 1110: $f_{SAMPLING} = f_{DTS}/32$, N=6
- 1111: $f_{SAMPLING} = f_{DTS}/32$, N=8

Bit 7 **MSM**: Master/slave mode

- 0: No action
- 1: The effect of an event on the trigger input (TRGI) is delayed to allow a perfect synchronization between the current timer and its slaves (through TRGO). It is useful if we want to synchronize several timers on a single external event.

Bits 6:4 **TS[2:0]**: Trigger selection

This bit-field selects the trigger input to be used to synchronize the counter.

- 000: Internal Trigger 0 (ITR0)
- 001: Internal Trigger 1 (ITR1)
- 010: Internal Trigger 2 (ITR2)
- 011: Internal Trigger 3 (ITR3)
- 100: T11 Edge Detector (T11F_ED)
- 101: Filtered Timer Input 1 (T11FP1)
- 110: Filtered Timer Input 2 (T11FP2)
- 111: External Trigger input (ETRF)

See [Table 114: TIMx internal trigger connection on page 575](#) for more details on ITRx meaning for each Timer.

Note: These bits must be changed only when they are not used (e.g. when SMS=000) to avoid wrong edge detections at the transition.

Note: The other bit is at position 16 in the same register

Bit 3 Reserved, must be kept at reset value.

Bits 16, 2, 1, 0 **SMS[3:0]**: Slave mode selection

When external signals are selected the active edge of the trigger signal (TRGI) is linked to the polarity selected on the external input (see Input Control register and Control Register description).

0000: Slave mode disabled - if CEN = '1' then the prescaler is clocked directly by the internal clock.

0001: Encoder mode 1 - Counter counts up/down on TI1FP1 edge depending on TI2FP2 level.

0010: Encoder mode 2 - Counter counts up/down on TI2FP2 edge depending on TI1FP1 level.

0011: Encoder mode 3 - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input.

0100: Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter and generates an update of the registers.

0101: 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.

0110: 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.

0111: External Clock Mode 1 - Rising edges of the selected trigger (TRGI) clock the counter.

1000: Combined reset + trigger mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter, generates an update of the registers and starts the counter.

Codes above 1000: Reserved.

Note: The gated mode must not be used if TI1F_ED is selected as the trigger input (TS=100). Indeed, TI1F_ED outputs 1 pulse for each transition on TI1F, whereas the gated mode checks the level of the trigger signal.

Note: The clock of the slave peripherals (timer, ADC, ...) receiving the TRGO or the TRGO2 signals must be enabled prior to receive events from the master timer, and the clock frequency (prescaler) must not be changed on-the-fly while triggers are received from the master timer.

Table 114. TIMx internal trigger connection

Slave TIM	ITR0 (TS = 000)	ITR1 (TS = 001)	ITR2 (TS = 010)	ITR3 (TS = 011)
TIM1	TIM5	TIM2	TIM3	TIM4
TIM8	TIM1	TIM2	TIM4	TIM5

18.4.4 TIMx DMA/interrupt enable register (TIMx_DIER)(x = 1, 8)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	COMDE	CC4DE	CC3DE	CC2DE	CC1DE	UDE	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 Reserved, must be kept at reset value.

Bit 14 **TDE**: Trigger DMA request enable

- 0: Trigger DMA request disabled
- 1: Trigger DMA request enabled

Bit 13 **COMDE**: COM DMA request enable

- 0: COM DMA request disabled
- 1: COM DMA request enabled

Bit 12 **CC4DE**: Capture/Compare 4 DMA request enable

- 0: CC4 DMA request disabled
- 1: CC4 DMA request enabled

Bit 11 **CC3DE**: Capture/Compare 3 DMA request enable

- 0: CC3 DMA request disabled
- 1: CC3 DMA request enabled

Bit 10 **CC2DE**: Capture/Compare 2 DMA request enable

- 0: CC2 DMA request disabled
- 1: CC2 DMA request enabled

Bit 9 **CC1DE**: Capture/Compare 1 DMA request enable

- 0: CC1 DMA request disabled
- 1: CC1 DMA request enabled

Bit 8 **UDE**: Update DMA request enable

- 0: Update DMA request disabled
- 1: Update DMA request enabled

Bit 7 **BIE**: Break interrupt enable

- 0: Break interrupt disabled
- 1: Break interrupt enabled

Bit 6 **TIE**: Trigger interrupt enable

- 0: Trigger interrupt disabled
- 1: Trigger interrupt enabled

Bit 5 **COMIE**: COM interrupt enable

- 0: COM interrupt disabled
- 1: COM interrupt enabled

Bit 4 **CC4IE**: Capture/Compare 4 interrupt enable

- 0: CC4 interrupt disabled
- 1: CC4 interrupt enabled

Bit 3 **CC3IE**: Capture/Compare 3 interrupt enable

- 0: CC3 interrupt disabled
- 1: CC3 interrupt enabled

Bit 2 **CC2IE**: Capture/Compare 2 interrupt enable

- 0: CC2 interrupt disabled
- 1: CC2 interrupt enabled

Bit 1 **CC1IE**: Capture/Compare 1 interrupt enable

- 0: CC1 interrupt disabled
- 1: CC1 interrupt enabled

Bit 0 **UIE**: Update interrupt enable

- 0: Update interrupt disabled
- 1: Update interrupt enabled

18.4.5 TIMx status register (TIMx_SR)(x = 1, 8)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CC6IF	CC5IF
														rc_w0	rc_w0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	Res		CC4OF	CC3OF	CC2OF	CC1OF	B2IF	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF
			rc_w0												

Bits 31:18 Reserved, must be kept at reset value.

Bit 17 **CC6IF**: Compare 6 interrupt flag

Refer to CC1IF description (Note: Channel 6 can only be configured as output)

Bit 16 **CC5IF**: Compare 5 interrupt flag

Refer to CC1IF description (Note: Channel 5 can only be configured as output)

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

Bit 12 **CC4OF**: Capture/Compare 4 overcapture flag

Refer to CC1OF description

Bit 11 **CC3OF**: Capture/Compare 3 overcapture flag

Refer to CC1OF description

Bit 10 **CC2OF**: Capture/Compare 2 overcapture flag

Refer to CC1OF description

Bit 9 **CC1OF**: Capture/Compare 1 overcapture flag

This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to '0'.

0: No overcapture has been detected.

1: The counter value has been captured in TIMx_CCR1 register while CC1IF flag was already set

Bit 8 **B2IF**: Break 2 interrupt flag

This flag is set by hardware as soon as the break 2 input goes active. It can be cleared by software if the break 2 input is not active.

0: No break event occurred.

1: An active level has been detected on the break 2 input. An interrupt is generated if BIE=1 in the TIMx_DIER register.

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. An interrupt is generated if BIE=1 in the TIMx_DIER register.

Bit 6 **TIF**: Trigger interrupt flag

This flag is set by hardware on trigger event (active edge detected on TRGI input when the slave mode controller is enabled in all modes but gated mode. It is set when the counter starts or stops when gated mode is selected. It is cleared by software.

0: No trigger event occurred.

1: Trigger interrupt pending.

Bit 5 **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 TIMx_CR1 register description). It is cleared by software.

0: No match.

1: The content of the counter TIMx_CNT matches the content of the TIMx_CCR1 register. When the contents of TIMx_CCR1 are greater than the contents of TIMx_ARR, the CC1IF bit goes high on the counter overflow (in upcounting and up/down-counting modes) or underflow (in downcounting mode)

If channel CC1 is configured as input: This bit is set by hardware on a capture. It is cleared by software or by reading the TIMx_CCR1 register.

0: No input capture occurred

1: The counter value has been captured in TIMx_CCR1 register (An edge has been detected on IC1 which matches the selected polarity)

Bit 0 **UIF**: Update interrupt flag

This bit is set by hardware on an update event. It is cleared by software.

0: No update occurred.

1: Update interrupt pending. This bit is set by hardware when the registers are updated:

- At overflow or underflow regarding the repetition counter value (update if repetition counter = 0) and if the UDIS=0 in the TIMx_CR1 register.
- When CNT is reinitialized by software using the UG bit in TIMx_EGR register, if URS=0 and UDIS=0 in the TIMx_CR1 register.
- When CNT is reinitialized by a trigger event (refer to [Section 18.4.3: TIMx slave mode control register \(TIMx_SMCR\)\(x = 1, 8\)](#)), if URS=0 and UDIS=0 in the TIMx_CR1 register.

18.4.6 TIMx event generation register (TIMx_EGR)(x = 1, 8)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	B2G	BG	TG	COMG	CC4G	CC3G	CC2G	CC1G	UG						
							w	w	w	w	w	w	w	w	w

Bits 15:9 Reserved, must be kept at reset value.

Bit 8 **B2G:** Break 2 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 2 event is generated. MOE bit is cleared and B2IF flag is set. Related interrupt can occur if enabled.

Bit 7 **BG:** Break generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: A break event is generated. MOE bit is cleared and BIF flag is set. Related interrupt or DMA transfer can occur if enabled.

Bit 6 **TG:** Trigger generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: The TIF flag is set in TIMx_SR register. Related interrupt or DMA transfer can occur if enabled.

Bit 5 **COMG:** Capture/Compare control update generation

This bit can be set by software, it is automatically cleared by hardware

0: No action

1: When CCPC bit is set, it allows to update CCxE, CCxNE and OCxM bits

Note: This bit acts only on channels having a complementary output.

Bit 4 **CC4G:** Capture/Compare 4 generation

Refer to CC1G description

Bit 3 **CC3G:** Capture/Compare 3 generation

Refer to CC1G description

Bit 2 **CC2G:** Capture/Compare 2 generation

Refer to CC1G description

Bit 1 **CC1G**: Capture/Compare 1 generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: A capture/compare event is generated on channel 1:

If channel CC1 is configured as output:

CC1IF flag is set, Corresponding interrupt or DMA request is sent if enabled.

If channel CC1 is configured as input:

The current value of the counter is captured in TIMx_CCR1 register. The CC1IF flag is set, the corresponding interrupt or DMA request is sent if enabled. The CC1OF flag is set if the CC1IF flag was already high.

Bit 0 **UG**: Update generation

This bit can be set by software, it is automatically cleared by hardware.

0: No action

1: Reinitialize the counter and generates an update of the registers. Note that the prescaler counter is cleared too (anyway the prescaler ratio is not affected). The counter is cleared if the center-aligned mode is selected or if DIR=0 (upcounting), else it takes the auto-reload value (TIMx_ARR) if DIR=1 (downcounting).

18.4.7 TIMx capture/compare mode register 1 (TIMx_CCMR1)(x = 1, 8)

Address offset: 0x18

Reset value: 0x0000 0000

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.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC2M[3]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC1M[3]
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
							rw								rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2 CE	OC2M[2:0]			OC2 PE	OC2 FE	CC2S[1:0]	OC1 CE	OC1M[2:0]			OC1 PE	OC1 FE	CC1S[1:0]		
IC2F[3:0]				IC2PSC[1:0]			IC1F[3:0]			IC1PSC[1:0]					
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Output compare mode

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

Bit 24 **OC2M[3]**: Output Compare 2 mode - bit 3

Refer to OC2M description on bits 14:12.

Bits 23:17 Reserved, must be kept at reset value.

Bits16 **OC1M[3]**: Output Compare 1 mode - bit 3

Refer to OC1M description on bits 6:4

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

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.

0000: Frozen - The comparison between the output compare register TIMx_CCR1 and the counter TIMx_CNT has no effect on the outputs.(this mode is used to generate a timing base).

0001: 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).

0010: 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).

0011: Toggle - OC1REF toggles when TIMx_CNT=TIMx_CCR1.

0100: Force inactive level - OC1REF is forced low.

0101: Force active level - OC1REF is forced high.

0110: 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').

0111: 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.

1000: Retriggerable OPM mode 1 - In up-counting mode, the channel is active until a trigger event is detected (on TRGI signal). Then, a comparison is performed as in PWM mode 1 and the channels becomes active again at the next update. In down-counting mode, the channel is inactive until a trigger event is detected (on TRGI signal). Then, a comparison is performed as in PWM mode 1 and the channels becomes inactive again at the next update.

1001: Retriggerable OPM mode 2 - In up-counting mode, the channel is inactive until a trigger event is detected (on TRGI signal). Then, a comparison is performed as in PWM mode 2 and the channels becomes inactive again at the next update. In down-counting mode, the channel is active until a trigger event is detected (on TRGI signal). Then, a comparison is performed as in PWM mode 1 and the channels becomes active again at the next update.

1010: Reserved,

1011: Reserved,

1100: Combined PWM mode 1 - OC1REF has the same behavior as in PWM mode 1. OC1REFC is the logical OR between OC1REF and OC2REF.

1101: Combined PWM mode 2 - OC1REF has the same behavior as in PWM mode 2. OC1REFC is the logical AND between OC1REF and OC2REF.

1110: Asymmetric PWM mode 1 - OC1REF has the same behavior as in PWM mode 1. OC1REFC outputs OC1REF when the counter is counting up, OC2REF when it is counting down.

1111: Asymmetric PWM mode 2 - OC1REF has the same behavior as in PWM mode 2. OC1REFC outputs OC1REF when the counter is counting up, OC2REF when it is counting down.

Note: These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx_BDTR register) and CC1S='00' (the channel is configured in output).

Note: In PWM mode, 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.

Note: On channels having a complementary output, this bit field is preloaded. If the CCPC bit is set in the TIMx_CR2 register then the OC1M active bits take the new value from the preloaded bits only when a COM event is generated.

Bit 3 **OC1PE**: Output Compare 1 preload enable

0: Preload register on TIMx_CCR1 disabled. TIMx_CCR1 can be written at anytime, the new value is taken in account immediately.

1: Preload register on TIMx_CCR1 enabled. Read/Write operations access the preload register. TIMx_CCR1 preload value is loaded in the active register at each update event.

Note: 1: These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx_BDTR register) and CC1S='00' (the channel is configured in output).

2: 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 input on the CC output.

0: CC1 behaves normally depending on counter and CCR1 values even when the trigger is ON. The minimum delay to activate CC1 output when an edge occurs on the trigger input is 5 clock cycles.

1: An active edge on the trigger input acts like a compare match on CC1 output. Then, OC is set to the compare level independently from the result of the comparison. Delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. OCFE acts only if the channel is configured in PWM1 or PWM2 mode.

Bits 1:0 **CC1S**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output

01: CC1 channel is configured as input, IC1 is mapped on TI1

10: CC1 channel is configured as input, IC1 is mapped on TI2

11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC1S bits are writable only when the channel is OFF (CC1E = '0' in TIMx_CCER).

Input capture mode

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

Bits 15:12 **IC2F**: Input capture 2 filter

Bits 11:10 **IC2PSC[1:0]**: Input capture 2 prescaler

Bits 9:8 **CC2S**: Capture/Compare 2 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output

01: CC2 channel is configured as input, IC2 is mapped on TI2

10: CC2 channel is configured as input, IC2 is mapped on TI1

11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC2S bits are writable only when the channel is OFF (CC2E = '0' in TIMx_CCER).

Bits 7:4 IC1F[3:0]: Input capture 1 filter

This bit-field defines the frequency used to sample TI1 input and the length of the digital filter applied to TI1. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output:

- 0000: No filter, sampling is done at f_{DTS}
- 0001: $f_{SAMPLING} = f_{CK_INT}$, N=2
- 0010: $f_{SAMPLING} = f_{CK_INT}$, N=4
- 0011: $f_{SAMPLING} = f_{CK_INT}$, N=8
- 0100: $f_{SAMPLING} = f_{DTS}/2$, N=6
- 0101: $f_{SAMPLING} = f_{DTS}/2$, N=8
- 0110: $f_{SAMPLING} = f_{DTS}/4$, N=6
- 0111: $f_{SAMPLING} = f_{DTS}/4$, N=8
- 1000: $f_{SAMPLING} = f_{DTS}/8$, N=6
- 1001: $f_{SAMPLING} = f_{DTS}/8$, N=8
- 1010: $f_{SAMPLING} = f_{DTS}/16$, N=5
- 1011: $f_{SAMPLING} = f_{DTS}/16$, N=6
- 1100: $f_{SAMPLING} = f_{DTS}/16$, N=8
- 1101: $f_{SAMPLING} = f_{DTS}/32$, N=5
- 1110: $f_{SAMPLING} = f_{DTS}/32$, N=6
- 1111: $f_{SAMPLING} = f_{DTS}/32$, N=8

Bits 3:2 IC1PSC: Input capture 1 prescaler

This bit-field defines the ratio of the prescaler acting on CC1 input (IC1). The prescaler is reset as soon as CC1E='0' (TIMx_CCER register).

- 00: no prescaler, capture is done each time an edge is detected on the capture input
- 01: capture is done once every 2 events
- 10: capture is done once every 4 events
- 11: capture is done once every 8 events

Bits 1:0 CC1S: Capture/Compare 1 Selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

- 00: CC1 channel is configured as output
- 01: CC1 channel is configured as input, IC1 is mapped on TI1
- 10: CC1 channel is configured as input, IC1 is mapped on TI2
- 11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC1S bits are writable only when the channel is OFF (CC1E = '0' in TIMx_CCER).

18.4.8 TIMx capture/compare mode register 2 (TIMx_CCMR2)(x = 1, 8)

Address offset: 0x1C

Reset value: 0x0000 0000

Refer to the above CCMR1 register description.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16				
Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC4M[3]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC3M[3]				
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
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			

Output compare mode

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

Bit 24 **OC4M[3]**: Output Compare 4 mode - bit 3

Bits 23:17 Reserved, must be kept at reset value.

Bit 16 **OC3M[3]**: Output Compare 3 mode - bit 3

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

Bits 15:12 **IC4F**: Input capture 4 filter

Bits 11:10 **IC4PSC**: Input capture 4 prescaler

Bits 9:8 **CC4S**: Capture/Compare 4 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC4 channel is configured as output

01: CC4 channel is configured as input, IC4 is mapped on TI4

10: CC4 channel is configured as input, IC4 is mapped on TI3

11: CC4 channel is configured as input, IC4 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC4S bits are writable only when the channel is OFF (CC4E = '0' in TIMx_CCER).

Bits 7:4 **IC3F**: Input capture 3 filter

Bits 3:2 **IC3PSC**: Input capture 3 prescaler

Bits 1:0 **CC3S**: Capture/compare 3 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC3 channel is configured as output

01: CC3 channel is configured as input, IC3 is mapped on TI3

10: CC3 channel is configured as input, IC3 is mapped on TI4

11: CC3 channel is configured as input, IC3 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC3S bits are writable only when the channel is OFF (CC3E = '0' in TIMx_CCER).

18.4.9 TIMx capture/compare enable register (TIMx_CCER)(x = 1, 8)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CC6P	CC6E	Res.	Res.	CC5P	CC5E
										rw	rw			rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC4NP	Res.	CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E
rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:22 Reserved, must be kept at reset value.

Bit 21 **CC6P**: Capture/Compare 6 output polarity

Refer to CC1P description

Bit 20 **CC6E**: Capture/Compare 6 output enable

Refer to CC1E description

Bits 19:18 Reserved, must be kept at reset value.

Bit 17 **CC5P**: Capture/Compare 5 output polarity

Refer to CC1P description

Bit 16 **CC5E**: Capture/Compare 5 output enable

Refer to CC1E description

Bit 15 **CC4NP**: Capture/Compare 4 complementary output polarity

Refer to CC1NP description

Bit 14 Reserved, must be kept at reset value.

Bit 13 **CC4P**: Capture/Compare 4 output polarity

Refer to CC1P description

Bit 12 **CC4E**: Capture/Compare 4 output enable

Refer to CC1E description

Bit 11 **CC3NP**: Capture/Compare 3 complementary output polarity

Refer to CC1NP description

Bit 10 **CC3NE**: Capture/Compare 3 complementary output enable

Refer to CC1NE description

- Bit 9 **CC3P**: Capture/Compare 3 output polarity
Refer to CC1P description
- Bit 8 **CC3E**: Capture/Compare 3 output enable
Refer to CC1E description
- Bit 7 **CC2NP**: Capture/Compare 2 complementary output polarity
Refer to CC1NP description
- Bit 6 **CC2NE**: Capture/Compare 2 complementary output enable
Refer to CC1NE description
- Bit 5 **CC2P**: Capture/Compare 2 output polarity
Refer to CC1P description
- Bit 4 **CC2E**: Capture/Compare 2 output enable
Refer to CC1E description
- Bit 3 **CC1NP**: Capture/Compare 1 complementary output polarity
CC1 channel configured as output:
0: OC1N active high.
1: OC1N active low.
CC1 channel configured as input:
This bit is used in conjunction with CC1P to define the polarity of TI1FP1 and TI2FP1. Refer to CC1P description.
Note: This bit is not writable as soon as LOCK level 2 or 3 has been programmed (LOCK bits in TIMx_BDTR register) and CC1S="00" (channel configured as output).
On channels having a complementary output, this bit is preloaded. If the CCPC bit is set in the TIMx_CR2 register then the CC1NP active bit takes the new value from the preloaded bit only when a Commutation event is generated.
- 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.
On channels having a complementary output, this bit is preloaded. If the CCPC bit is set in the TIMx_CR2 register then the CC1NE active bit takes the new value from the preloaded bit only when a Commutation event is generated.

Bit 1 **CC1P**: Capture/Compare 1 output polarity

CC1 channel configured as output:

0: OC1 active high

1: OC1 active low

CC1 channel configured as input: CC1NP/CC1P bits select the active polarity of TI1FP1 and TI2FP1 for trigger or capture operations.

00: non-inverted/rising edge. The circuit is sensitive to TIxFP1 rising edge (capture or trigger operations in reset, external clock or trigger mode), TIxFP1 is not inverted (trigger operation in gated mode or encoder mode).

01: inverted/falling edge. The circuit is sensitive to TIxFP1 falling edge (capture or trigger operations in reset, external clock or trigger mode), TIxFP1 is inverted (trigger operation in gated mode or encoder mode).

10: reserved, do not use this configuration.

11: non-inverted/both edges/ The circuit is sensitive to both TIxFP1 rising and falling edges (capture or trigger operations in reset, external clock or trigger mode), TIxFP1 is not inverted (trigger operation in gated mode). This configuration must not be used in encoder mode.

Note: This bit is not writable as soon as LOCK level 2 or 3 has been programmed (LOCK bits in TIMx_BDTR register).

On channels having a complementary output, this bit is preloaded. If the CCPC bit is set in the TIMx_CR2 register then the CC1P active bit takes the new value from the preloaded bit only when a Commutation event is generated.

Bit 0 **CC1E**: Capture/Compare 1 output enable

CC1 channel configured as output:

0: Off - OC1 is not active. OC1 level is then function of MOE, OSS1, OSSR, OIS1, OIS1N and CC1NE bits.

1: On - OC1 signal is output on the corresponding output pin depending on MOE, OSS1, OSSR, OIS1, OIS1N and CC1NE bits.

CC1 channel configured as input: This bit determines if a capture of the counter value can actually be done into the input capture/compare register 1 (TIMx_CCR1) or not.

0: Capture disabled.

1: Capture enabled.

On channels having a complementary output, this bit is preloaded. If the CCPC bit is set in the TIMx_CR2 register then the CC1E active bit takes the new value from the preloaded bit only when a Commutation event is generated.

Table 115. 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	X	0	0	Output disabled (not driven by the timer: Hi-Z) OCx=0, OCxN=0	
		0	0	1	Output disabled (not driven by the timer: Hi-Z) OCx=0	OCxREF + Polarity OCxN = OCxREF xor CCxNP
		0	1	0	OCxREF + Polarity OCx=OCxREF xor CCxP	Output Disabled (not driven by the timer: Hi-Z) OCxN=0
		X	1	1	OCREF + Polarity + dead-time	Complementary to OCREF (not OCREF) + Polarity + dead-time
		1	0	1	Off-State (output enabled with inactive state) OCx=CCxP	OCxREF + Polarity OCxN = OCxREF x or CCxNP
		1	1	0	OCxREF + Polarity OCx=OCxREF xor CCxP	Off-State (output enabled with inactive state) OCxN=CCxNP
0	1	0	X	X	Output disabled (not driven by the timer anymore). The output state is defined by the GPIO controller and can be High, Low or Hi-Z.	
		0	0		Off-State (output enabled with inactive state) Asynchronously: OCx=CCxP, OCxN=CCxNP (if BRK or BRK2 is triggered).	
		0	1			
		1	0			
		1	1		Then (this is valid only if BRK is triggered), 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 (may cause a short circuit when driving switches in half-bridge configuration). Note: BRK2 can only be used if OSSI = OSSR = 1.	

1. When both outputs of a channel are not used (control taken over by GPIO), the OISx, OISxN, CCxP and CCxNP bits must be kept cleared.

Note: *The state of the external I/O pins connected to the complementary OCx and OCxN channels depends on the OCx and OCxN channel state and the GPIO registers.*

18.4.10 TIMx counter (TIMx_CNT)(x = 1, 8)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UIF CPY	Res.														
r															
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **UIFCPY**: UIF copy

This bit is a read-only copy of the UIF bit of the TIMx_ISR register. If the UIFREMAP bit in the TIMxCR1 is reset, bit 31 is reserved and read at 0.

Bits 30:16 Reserved, must be kept at reset value.

Bits 15:0 **CNT[15:0]**: Counter value

18.4.11 TIMx prescaler (TIMx_PSC)(x = 1, 8)

Address offset: 0x28

Reset value: 0x0000

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

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency (CK_CNT) is equal to $f_{CK_PSC} / (PSC[15:0] + 1)$.

PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx_EGR register or through trigger controller when configured in “reset mode”).

18.4.12 TIMx auto-reload register (TIMx_ARR)(x = 1, 8)

Address offset: 0x2C

Reset value: 0xFFFF

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

Bits 15:0 **ARR[15:0]**: Auto-reload value

ARR is the value to be loaded in the actual auto-reload register.

Refer to the [Section 18.3.1: Time-base unit on page 512](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

18.4.13 TIMx repetition counter register (TIMx_RCR)(x = 1, 8)

Address offset: 0x30

Reset value: 0x0000

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

Bits 15:0 **REP[15:0]**: Repetition counter value

These bits allow the user to set-up the update rate of the compare registers (i.e. periodic transfers from preload to active registers) when preload registers are enable, as well as the update interrupt generation rate, if this interrupt is enable.

Each time the REP_CNT related downcounter reaches zero, an update event is generated and it restarts counting from REP value. As REP_CNT is reloaded with REP value only at the repetition update event U_RC, any write to the TIMx_RCR register is not taken in account until the next repetition update event.

It means in PWM mode (REP+1) corresponds to:
the number of PWM periods in edge-aligned mode
the number of half PWM period in center-aligned mode.

18.4.14 TIMx capture/compare register 1 (TIMx_CCR1)(x = 1, 8)

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: CR1 is the counter value transferred by the last input capture 1 event (IC1). The TIMx_CCR1 register is read-only and cannot be programmed.

18.4.15 TIMx capture/compare register 2 (TIMx_CCR2)(x = 1, 8)

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_CCMR1 register (bit OC2PE). Else the preload value is copied in the active capture/compare 2 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx_CNT and signaled on OC2 output.

If channel CC2 is configured as input: CCR2 is the counter value transferred by the last input capture 2 event (IC2). The TIMx_CCR2 register is read-only and cannot be programmed.

18.4.16 TIMx capture/compare register 3 (TIMx_CCR3)(x = 1, 8)

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_CCMR2 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). The TIMx_CCR3 register is read-only and cannot be programmed.

18.4.17 TIMx capture/compare register 4 (TIMx_CCR4)(x = 1, 8)

Address offset: 0x40

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR4[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR4[15:0]**: Capture/Compare value

If channel CC4 is configured as output: CCR4 is the value to be loaded in the actual capture/compare 4 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx_CCMR2 register (bit OC4PE). Else the preload value is copied in the active capture/compare 4 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx_CNT and signalled on OC4 output.

If channel CC4 is configured as input: CCR4 is the counter value transferred by the last input capture 4 event (IC4). The TIMx_CCR4 register is read-only and cannot be programmed.

18.4.18 TIMx break and dead-time register (TIMx_BDTR)(x = 1, 8)

Address offset: 0x44

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	BK2P	BK2E	BK2F[3:0]				BKF[3:0]			
						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
MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK[1:0]		DTG[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw						

Note: As the bits BK2P, BK2E, BK2F[3:0], BKF[3:0], AOE, BKP, BKE, OSSR and DTG[7:0] can be write-locked depending on the LOCK configuration, it can be necessary to configure all of them during the first write access to the TIMx_BDTR register.

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

Bit 25 **BK2P**: Break 2 polarity

0: Break input BRK2 is active low

1: Break input BRK2 is active high

Note: This bit cannot be modified as long as LOCK level 1 has been programmed (LOCK bits in TIMx_BDTR register).

Note: Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.

Bit 24 **BK2E**: Break 2 enable

This bit enables the complete break 2 protection (including all sources connected to bk_acth and BKIN sources, as per [Figure 176: Break and Break2 circuitry overview](#)).

- 0: Break2 function disabled
- 1: Break2 function enabled

Note: The BRKIN2 must only be used with OSSR = OSSI = 1.

Note: This bit cannot be modified when LOCK level 1 has been programmed (LOCK bits in TIMx_BDTR register).

Note: Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.

Bits 23:20 **BK2F[3:0]**: Break 2 filter

This bit-field defines the frequency used to sample BRK2 input and the length of the digital filter applied to BRK2. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output:

- 0000: No filter, BRK2 acts asynchronously
- 0001: $f_{\text{SAMPLING}}=f_{\text{CK_INT}}$, N=2
- 0010: $f_{\text{SAMPLING}}=f_{\text{CK_INT}}$, N=4
- 0011: $f_{\text{SAMPLING}}=f_{\text{CK_INT}}$, N=8
- 0100: $f_{\text{SAMPLING}}=f_{\text{DTS}}/2$, N=6
- 0101: $f_{\text{SAMPLING}}=f_{\text{DTS}}/2$, N=8
- 0110: $f_{\text{SAMPLING}}=f_{\text{DTS}}/4$, N=6
- 0111: $f_{\text{SAMPLING}}=f_{\text{DTS}}/4$, N=8
- 1000: $f_{\text{SAMPLING}}=f_{\text{DTS}}/8$, N=6
- 1001: $f_{\text{SAMPLING}}=f_{\text{DTS}}/8$, N=8
- 1010: $f_{\text{SAMPLING}}=f_{\text{DTS}}/16$, N=5
- 1011: $f_{\text{SAMPLING}}=f_{\text{DTS}}/16$, N=6
- 1100: $f_{\text{SAMPLING}}=f_{\text{DTS}}/16$, N=8
- 1101: $f_{\text{SAMPLING}}=f_{\text{DTS}}/32$, N=5
- 1110: $f_{\text{SAMPLING}}=f_{\text{DTS}}/32$, N=6
- 1111: $f_{\text{SAMPLING}}=f_{\text{DTS}}/32$, N=8

Note: This bit cannot be modified when LOCK level 1 has been programmed (LOCK bits in TIMx_BDTR register).

Bits 19:16 **BKF[3:0]**: Break filter

This bit-field defines the frequency used to sample BRK input and the length of the digital filter applied to BRK. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output:

- 0000: No filter, BRK acts asynchronously
- 0001: $f_{\text{SAMPLING}}=f_{\text{CK_INT}}$, N=2
- 0010: $f_{\text{SAMPLING}}=f_{\text{CK_INT}}$, N=4
- 0011: $f_{\text{SAMPLING}}=f_{\text{CK_INT}}$, N=8
- 0100: $f_{\text{SAMPLING}}=f_{\text{DTS}}/2$, N=6
- 0101: $f_{\text{SAMPLING}}=f_{\text{DTS}}/2$, N=8
- 0110: $f_{\text{SAMPLING}}=f_{\text{DTS}}/4$, N=6
- 0111: $f_{\text{SAMPLING}}=f_{\text{DTS}}/4$, N=8
- 1000: $f_{\text{SAMPLING}}=f_{\text{DTS}}/8$, N=6
- 1001: $f_{\text{SAMPLING}}=f_{\text{DTS}}/8$, N=8
- 1010: $f_{\text{SAMPLING}}=f_{\text{DTS}}/16$, N=5
- 1011: $f_{\text{SAMPLING}}=f_{\text{DTS}}/16$, N=6
- 1100: $f_{\text{SAMPLING}}=f_{\text{DTS}}/16$, N=8
- 1101: $f_{\text{SAMPLING}}=f_{\text{DTS}}/32$, N=5
- 1110: $f_{\text{SAMPLING}}=f_{\text{DTS}}/32$, N=6
- 1111: $f_{\text{SAMPLING}}=f_{\text{DTS}}/32$, N=8

Note: This bit cannot be modified when LOCK level 1 has been programmed (LOCK bits in TIMx_BDTR register).

Bit 15 **MOE**: Main output enable

This bit is cleared asynchronously by hardware as soon as one of the break inputs is active (BRK or BRK2). 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: In response to a break 2 event. OC and OCN outputs are disabled

In response to a break event or if MOE is written to 0: OC and OCN outputs are disabled or forced to idle state depending on the OSS1 bit.

1: OC and OCN outputs are enabled if their respective enable bits are set (CCxE, CCxNE in TIMx_CCER register).

See OC/OCN enable description for more details ([Section 18.4.9: TIMx capture/compare enable register \(TIMx_CCER\)\(x = 1, 8\)](#)).

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 none of the break inputs BRK and BRK2 is active)

Note: This bit can not be modified as long as LOCK level 1 has been programmed (LOCK bits in TIMx_BDTR register).

Bit 13 **BKP**: Break polarity

0: Break input BRK is active low

1: Break input BRK is active high

Note: This bit can not be modified as long as LOCK level 1 has been programmed (LOCK bits in TIMx_BDTR register).

Note: Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.

Bit 12 **BKE**: Break enable

This bit enables the complete break protection (including all sources connected to bk_acth and BKIN sources, as per [Figure 176: Break and Break2 circuitry overview](#)).

0: Break function disabled

1: Break function enabled

Note: This bit cannot be modified when LOCK level 1 has been programmed (LOCK bits in TIMx_BDTR register).

Note: Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.

Bit 11 **OSSR**: Off-state selection for Run mode

This bit is used when MOE=1 on channels having a complementary output which are configured as outputs. OSSR is not implemented if no complementary output is implemented in the timer.

See OC/OCN enable description for more details ([Section 18.4.9: TIMx capture/compare enable register \(TIMx_CCER\)\(x = 1, 8\)](#)).

0: When inactive, OC/OCN outputs are disabled (the timer releases the output control which is taken over by the GPIO logic, which forces a Hi-Z state).

1: When inactive, OC/OCN outputs are enabled with their inactive level as soon as CCxE=1 or CCxNE=1 (the output is still controlled by the timer).

Note: This bit can not be modified as soon as the LOCK level 2 has been programmed (LOCK bits in TIMx_BDTR register).

Bit 10 **OSSI**: Off-state selection for Idle mode

This bit is used when MOE=0 due to a break event or by a software write, on channels configured as outputs.

See OC/OCN enable description for more details ([Section 18.4.9: TIMx capture/compare enable register \(TIMx_CCER\)\(x = 1, 8\)](#)).

0: When inactive, OC/OCN outputs are disabled (the timer releases the output control which is taken over by the GPIO logic and which imposes a Hi-Z state).

1: When inactive, OC/OCN outputs are first forced with their inactive level then forced to their idle level after the deadtime. The timer maintains its control over the output.

Note: This bit can not be modified as soon as the LOCK level 2 has been programmed (LOCK bits in TIMx_BDTR register).

Bits 9:8 **LOCK[1:0]**: Lock configuration

These bits offer a write protection against software errors.

00: LOCK OFF - No bit is write protected.

01: LOCK Level 1 = DTG bits in TIMx_BDTR register, OISx and OISxN bits in TIMx_CR2 register and BKE/BKP/AOE bits in TIMx_BDTR register can no longer be written.

10: LOCK Level 2 = LOCK Level 1 + CC Polarity bits (CCxP/CCxNP bits in TIMx_CCER register, as long as the related channel is configured in output through the CCxS bits) as well as OSSR and OSSI bits can no longer be written.

11: LOCK Level 3 = LOCK Level 2 + CC Control bits (OCxM and OCxPE bits in TIMx_CCMRx registers, as long as the related channel is configured in output through the CCxS bits) can no longer be written.

Note: The LOCK bits can be written only once after the reset. Once the TIMx_BDTR register has been written, their content is frozen until the next reset.

Bits 7:0 **DTG[7:0]**: Dead-time generator setup

This bit-field defines the duration of the dead-time inserted between the complementary outputs. DT correspond to this duration.

DTG[7:5]=0xx => DT=DTG[7:0]x t_{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 TIMx_BDTR register).

18.4.19 TIMx DMA control register (TIMx_DCR)(x = 1, 8)

Address offset: 0x48

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.			DBL[4:0]		Res.	Res.	Res.		DBA[4:0]				

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

Bits 12:8 DBL[4:0]: DMA burst length

This 5-bit vector defines the length of DMA transfers (the timer recognizes a burst transfer when a read or a write access is done to the TIMx_DMAR address), i.e. the number of transfers. Transfers can be in half-words or in bytes (see example below).

00000: 1 transfer

00001: 2 transfers

00010: 3 transfers

...

10001: 18 transfers

Example: Let us consider the following transfer: DBL = 7 bytes & DBA = TIM2_CR1.

- If DBL = 7 bytes and DBA = TIM2_CR1 represents the address of the byte to be transferred, the address of the transfer should be given by the following equation:

(TIMx_CR1 address) + DBA + (DMA index), where DMA index = DBL

In this example, 7 bytes are added to (TIMx_CR1 address) + DBA, which gives us the address from/to which the data will be copied. In this case, the transfer is done to 7 registers starting from the following address: (TIMx_CR1 address) + DBA

According to the configuration of the DMA Data Size, several cases may occur:

- If you configure the DMA Data Size in half-words, 16-bit data will be transferred to each of the 7 registers.
- If you configure the DMA Data Size in bytes, the data will also be transferred to 7 registers: the first register will contain the first MSB byte, the second register, the first LSB byte and so on. So with the transfer Timer, you also have to specify the size of data transferred by DMA.

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

Bits 4:0 DBA[4:0]: DMA base address

This 5-bits vector defines the base-address for DMA transfers (when read/write access are done through the TIMx_DMAR address). DBA is defined as an offset starting from the address of the TIMx_CR1 register.

Example:

00000: TIMx_CR1,

00001: TIMx_CR2,

00010: TIMx_SMCR,

...

18.4.20 TIMx DMA address for full transfer (TIMx_DMAR)(x = 1, 8)

Address offset: 0x4C

Reset value: 0x0000 0000

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

Bits 31:0 **DMAB[31:0]**: DMA register for burst accesses

A read or write operation to the DMAR register accesses the register located at the address (TIMx_CR1 address) + (DBA + DMA index) × 4

where TIMx_CR1 address is the address of the control register 1, DBA is the DMA base address configured in TIMx_DCR register, DMA index is automatically controlled by the DMA transfer, and ranges from 0 to DBL (DBL configured in TIMx_DCR).

18.4.21 TIMx capture/compare mode register 3 (TIMx_CCMR3)(x = 1, 8)

Address offset: 0x54

Reset value: 0x0000 0000

Refer to the above CCMR1 register description. Channels 5 and 6 can only be configured in output.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC6M[3]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC5M[3]
							rw								rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC6 CE	OC6M[2:0]			OC6 PE	OC6FE	Res.	Res.	OC5 CE	OC5M[2:0]			OC5PE	OC5FE	Res.	Res.
rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw		

Output compare mode

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

Bits 23:17 Reserved, must be kept at reset value.

Bit 15 **OC6CE**: Output compare 6 clear enable

Bits 24, 14, 13, 12 **OC6M[3:0]**: Output compare 6 mode

Bit 11 **OC6PE**: Output compare 6 preload enable

Bit 10 **OC6FE**: Output compare 6 fast enable

Bits 9:8 Reserved, must be kept at reset value.

Bit 7 **OC5CE**: Output compare 5 clear enable

Bits 16, 6, 5, 4 **OC5M[3:0]**: Output compare 5 mode

Bit 3 **OC5PE**: Output compare 5 preload enable

Bit 2 **OC5FE**: Output compare 5 fast enable

Bits 1:0 Reserved, must be kept at reset value.

18.4.22 TIMx capture/compare register 5 (TIMx_CCR5)(x = 1, 8)

Address offset: 0x58

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
GC5C3	GC5C2	GC5C1	Res.												
rw	rw	rw													
CCR5[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **GC5C3**: Group Channel 5 and Channel 3

Distortion on Channel 3 output:

0: No effect of OC5REF on OC3REFC

1: OC3REFC is the logical AND of OC3REFC and OC5REF

This bit can either have immediate effect or be preloaded and taken into account after an update event (if preload feature is selected in TIMxCCMR2).

Note: it is also possible to apply this distortion on combined PWM signals.

Bit 30 **GC5C2**: Group Channel 5 and Channel 2

Distortion on Channel 2 output:

0: No effect of OC5REF on OC2REFC

1: OC2REFC is the logical AND of OC2REFC and OC5REF

This bit can either have immediate effect or be preloaded and taken into account after an update event (if preload feature is selected in TIMxCCMR1).

Note: it is also possible to apply this distortion on combined PWM signals.

Bit 29 **GC5C1**: Group Channel 5 and Channel 1

Distortion on Channel 1 output:

0: No effect of OC5REF on OC1REFC5

1: OC1REFC is the logical AND of OC1REFC and OC5REF

This bit can either have immediate effect or be preloaded and taken into account after an update event (if preload feature is selected in TIMxCCMR1).

Note: it is also possible to apply this distortion on combined PWM signals.

Bits 28:16 Reserved, must be kept at reset value.

Bits 15:0 **CCR5[15:0]**: Capture/Compare 5 value

CCR5 is the value to be loaded in the actual capture/compare 5 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx_CCMR3 register (bit OC5PE). Else the preload value is copied in the active capture/compare 5 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 OC5 output.

18.4.23 TIMx capture/compare register 6 (TIMx_CCR6)(x = 1, 8)

Address offset: 0x5C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR6[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR6[15:0]**: Capture/Compare 6 value

CCR6 is the value to be loaded in the actual capture/compare 6 register (preload value). It is loaded permanently if the preload feature is not selected in the TIMx_CCMR3 register (bit OC6PE). Else the preload value is copied in the active capture/compare 6 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 OC6 output.

18.4.24 TIM1 register map

TIM1 registers are mapped as 16-bit addressable registers as described in the table below:

Table 116. TIM1 register map and reset values

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x00	TIM1_CR1	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																																
0x04	TIM1_CR2	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																																
0x08	TIM1_SMCR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																																
0x0C	TIM1_DIER	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																																
0x10	TIM1_SR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																																
0x14	TIM1_EGR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																																
0x18	TIM1_CCMR1 Output Compare mode	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value																																	
	TIM1_CCMR1 Input Capture mode	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value																																	
0x1C	TIM1_CCMR2 Output Compare mode	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value																																	
	TIM1_CCMR2 Input Capture mode	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value																																	
0x20	TIM1_CCER	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																																
0x24	TIM1_CNT	0	UIFCPY	Res.	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	0	0	

Table 116. TIM1 register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x28	TIM1_PSC	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x2C	TIM1_ARR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x30	TIM1_RCR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x34	TIM1_CCR1	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x38	TIM1_CCR2	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x3C	TIM1_CCR3	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x40	TIM1_CCR4	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x44	TIM1_BDTR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x48	TIM1_DCR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x4C	TIM1_DMAR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x54	TIM1_CCMR3 Output Compare mode	Res.	GC5C3	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value	0	0	GC5C2	GC5C1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x58	TIM1_CCR5	Res.	0	0	GC5CE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
	Reset value																																
0x5C	TIM1_CCR6	Res.	0	0	OC5M[3]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
	Reset value																																

Refer to [Section 1.5.2: Memory map and register boundary addresses](#) for the register boundary addresses.

18.4.25 TIM8 register map

TIM8 registers are mapped as 16-bit addressable registers as described in the table below:

Table 117. TIM8 register map and reset values

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x00	TIM8_CR1	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	
		Reset value																																
0x04	TIM8_CR2	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																																
0x08	TIM8_SMCR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																																
0x0C	TIM8_DIER	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																																
0x10	TIM8_SR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																																
0x14	TIM8_EGR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																																
0x18	TIM8_CCMR1 Output Compare mode	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																																
0x1C	TIM8_CCMR1 Input Capture mode	0	OC4M[3]	Res.																														
		Reset value																																
0x20	TIM8_CCMR2 Output Compare mode	0	OC4CE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		Reset value																																
0x24	TIM8_CCER	0	CC4P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		Reset value																																
0x24	TIM8_CNT	0	UIFCPY	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		Reset value																																
CNT[15:0]																																		

Table 117. TIM8 register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x28	TIM8_PSC	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x2C	TIM8_ARR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x30	TIM8_RCR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x34	TIMx_CCR1	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x38	TIM8_CCR2	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x3C	TIM8_CCR3	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x40	TIM8_CCR4	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x44	TIM8_BDTR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x48	TIM8_DCR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x4C	TIM8_DMAR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x54	TIM8_CCMR3 Output Compare mode	Res.	GC5C3	GC5C2	GC5C1	Res.																											
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x58	TIM8_CCR5	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x5C	TIM8_CCR6	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

19 General-purpose timers (TIM2/TIM3/TIM4/TIM5)

19.1 TIM2/TIM3/TIM4/TIM5 introduction

The general-purpose timers consist of a 16-bit or 32-bit auto-reload counter driven by a programmable prescaler.

They may be used for a variety of purposes, including measuring the pulse lengths of input signals (*input capture*) or generating output waveforms (*output compare and PWM*).

Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the RCC clock controller prescalers.

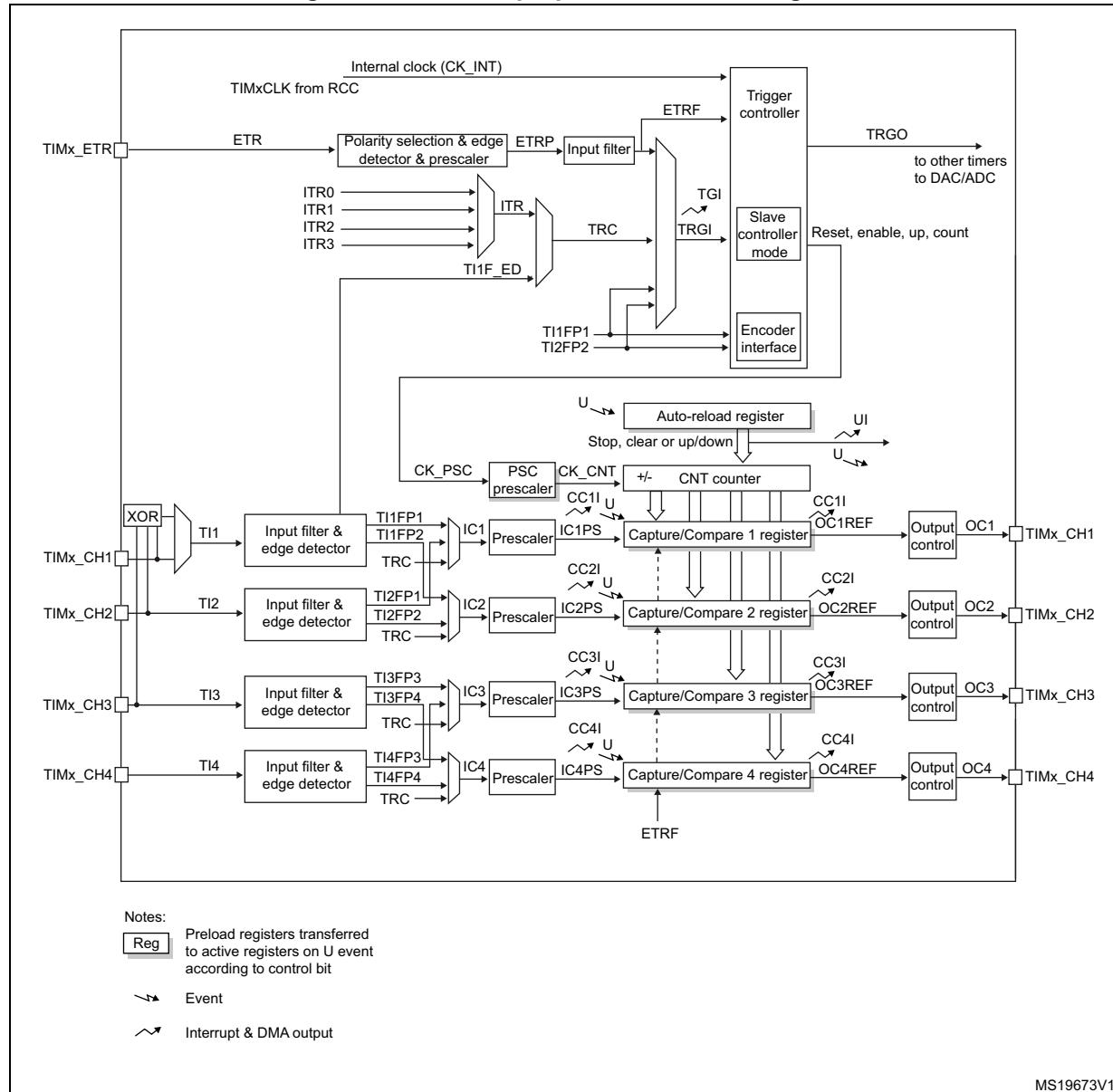
The timers are completely independent, and do not share any resources. They can be synchronized together as described in [Section 19.3.19: Timer synchronization](#).

19.2 TIM2/TIM3/TIM4/TIM5 main features

General-purpose TIMx timer features include:

- 16-bit (TIM3, TIM4) or 32-bit (TIM2 and TIM5) up, down, up/down auto-reload counter.
- 16-bit programmable prescaler used to divide (also “on the fly”) the counter clock frequency by any factor between 1 and 65535.
- Up to 4 independent channels for:
 - Input capture
 - Output compare
 - PWM generation (Edge- and Center-aligned modes)
 - One-pulse mode output
- Synchronization circuit to control the timer with external signals and to interconnect several timers.
- Interrupt/DMA generation on the following events:
 - Update: counter overflow/underflow, counter initialization (by software or internal/external trigger)
 - Trigger event (counter start, stop, initialization or count by internal/external trigger)
 - Input capture
 - Output compare
- Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes
- Trigger input for external clock or cycle-by-cycle current management

Figure 192. General-purpose timer block diagram



19.3 TIM2/TIM3/TIM4/TIM5 functional description

19.3.1 Time-base unit

The main block of the programmable timer is a 16-bit/32-bit counter with its related auto-reload register. The counter can count up, down or both up and down but also down or both up and down. The counter clock can be divided by a prescaler.

The counter, the auto-reload register and the prescaler register can be written or read by software. This is true even when the counter is running.

The time-base unit includes:

- Counter Register (TIMx_CNT)
- Prescaler Register (TIMx_PSC):
- Auto-Reload Register (TIMx_ARR)

The auto-reload register is preloaded. Writing to or reading from the auto-reload register accesses the preload register. The content of the preload register are transferred into the shadow register permanently or at each update event (UEV), depending on the auto-reload preload enable bit (ARPE) in TIMx_CR1 register. The update event is sent when the counter reaches the overflow (or underflow when downcounting) and if the UDIS bit equals 0 in the TIMx_CR1 register. It can also be generated by software. The generation of the update event is described in detail for each configuration.

The counter is clocked by the prescaler output CK_CNT, which is enabled only when the counter enable bit (CEN) in TIMx_CR1 register is set (refer also to the slave mode controller description to get more details on counter enabling).

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

Prescaler description

The prescaler can divide the counter clock frequency by any factor between 1 and 65536. It is based on a 16-bit counter controlled through a 16-bit/32-bit register (in the TIMx_PSC register). It can be changed on the fly as this control register is buffered. The new prescaler ratio is taken into account at the next update event.

Figure 193 and *Figure 194* give some examples of the counter behavior when the prescaler ratio is changed on the fly:

Figure 193. Counter timing diagram with prescaler division change from 1 to 2

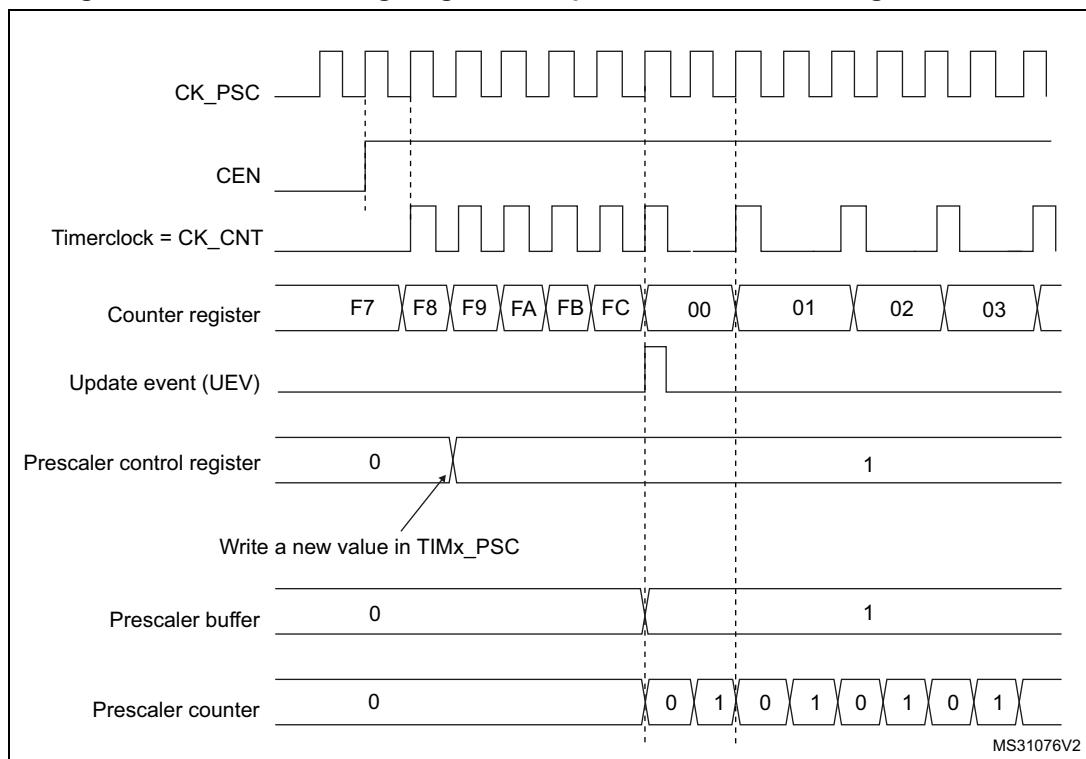
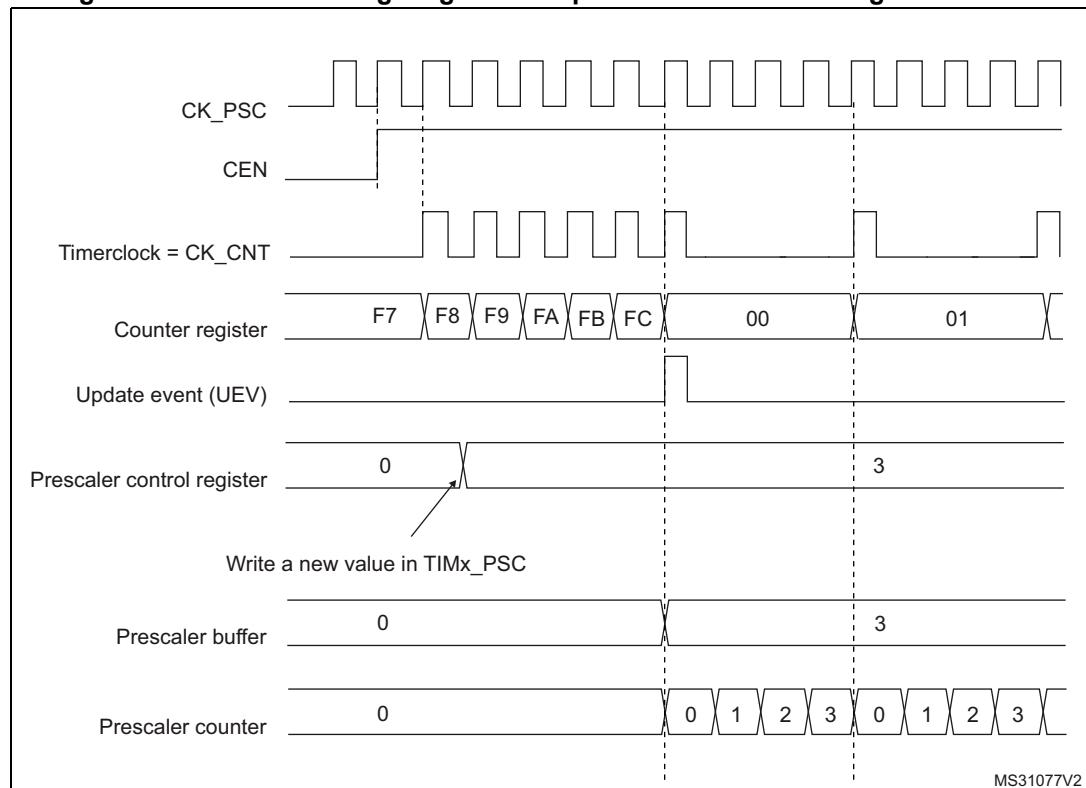


Figure 194. Counter timing diagram with prescaler division change from 1 to 4



19.3.2 Counter modes

Upcounting mode

In upcounting mode, the counter counts from 0 to the auto-reload value (content of the TIMx_ARR register), then restarts from 0 and generates a counter overflow event.

An Update event can be generated at each counter overflow or by setting the UG bit in the TIMx_EGR register (by software or by using the slave mode controller).

The UEV event can be disabled by software by setting the UDIS bit in TIMx_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until the UDIS bit has been written to 0. However, the counter restarts from 0, as well as the counter of the prescaler (but the prescale rate does not change). In addition, if the URS bit (update request selection) in TIMx_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

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

- The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSC register)
- The auto-reload shadow register is updated with the preload value (TIMx_ARR)

The following figures show some examples of the counter behavior for different clock frequencies when TIMx_ARR=0x36.

Figure 195. Counter timing diagram, internal clock divided by 1

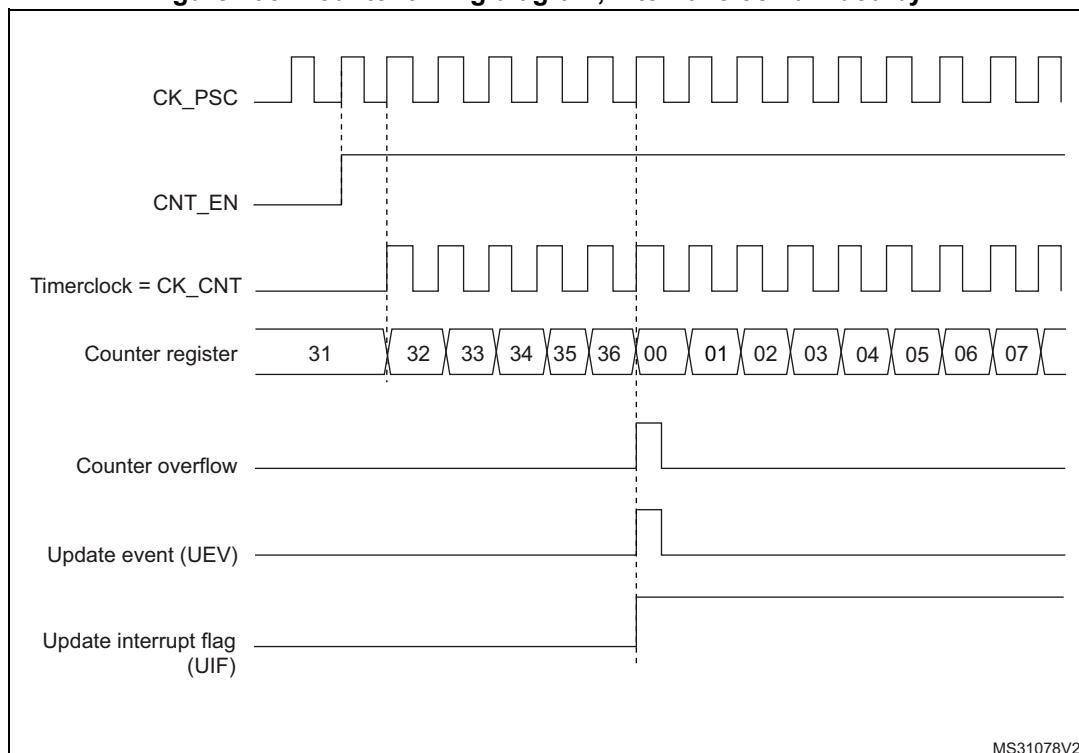


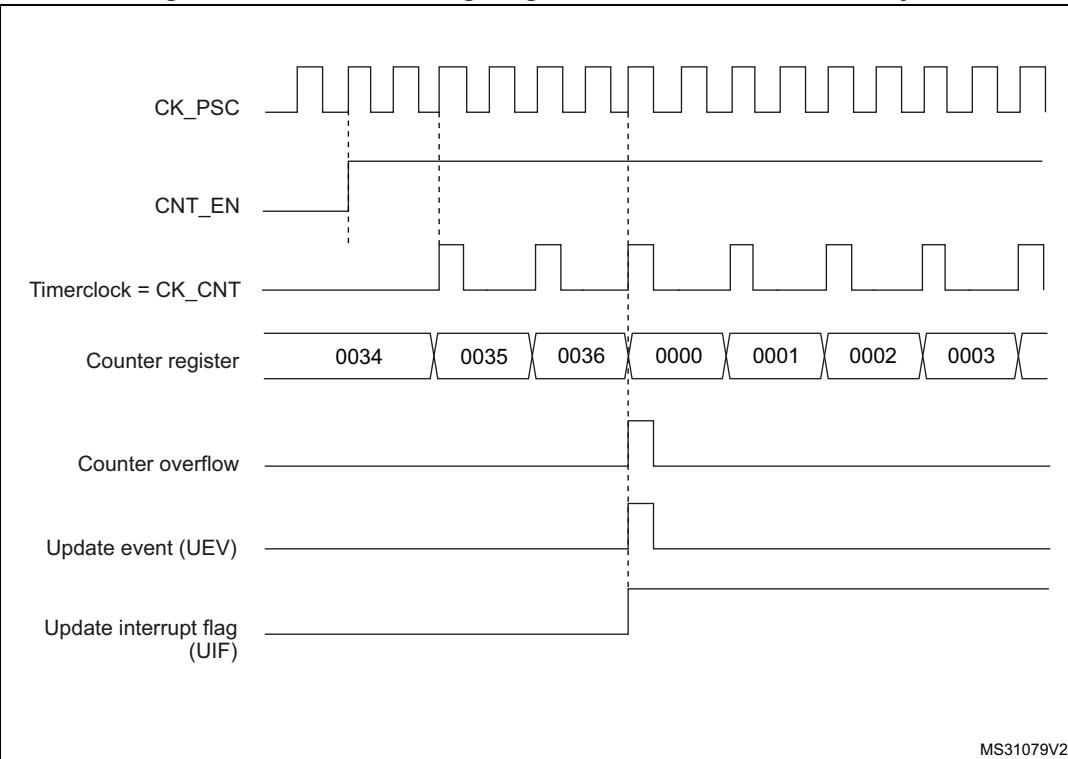
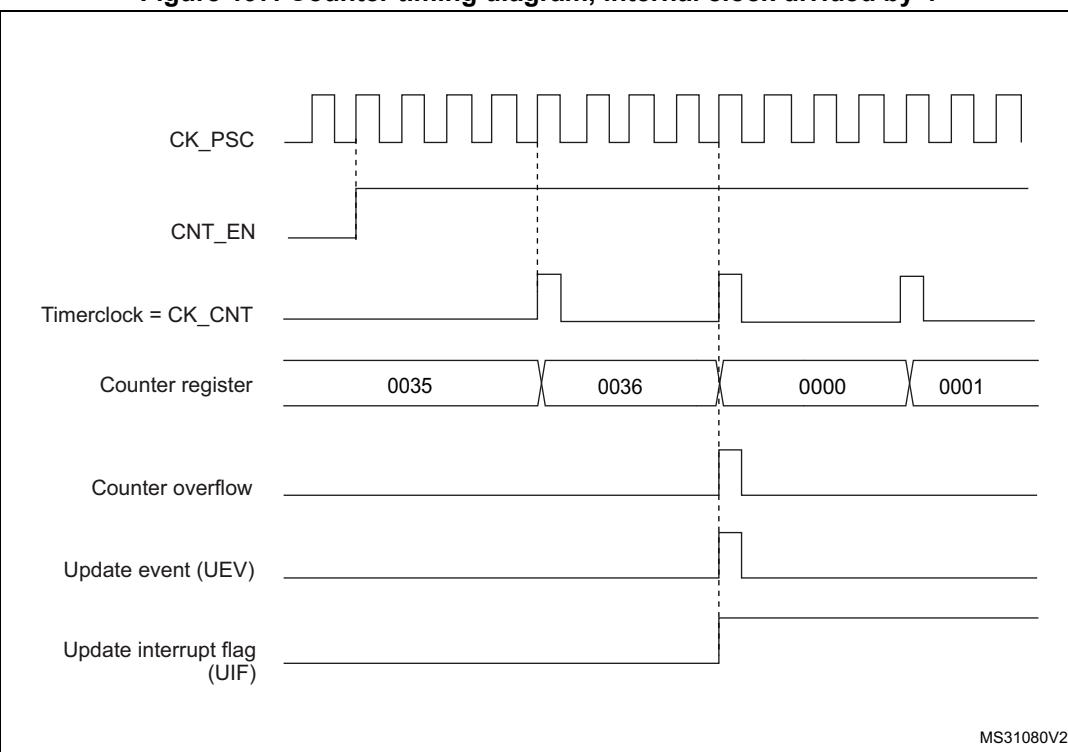
Figure 196. Counter timing diagram, internal clock divided by 2**Figure 197. Counter timing diagram, internal clock divided by 4**

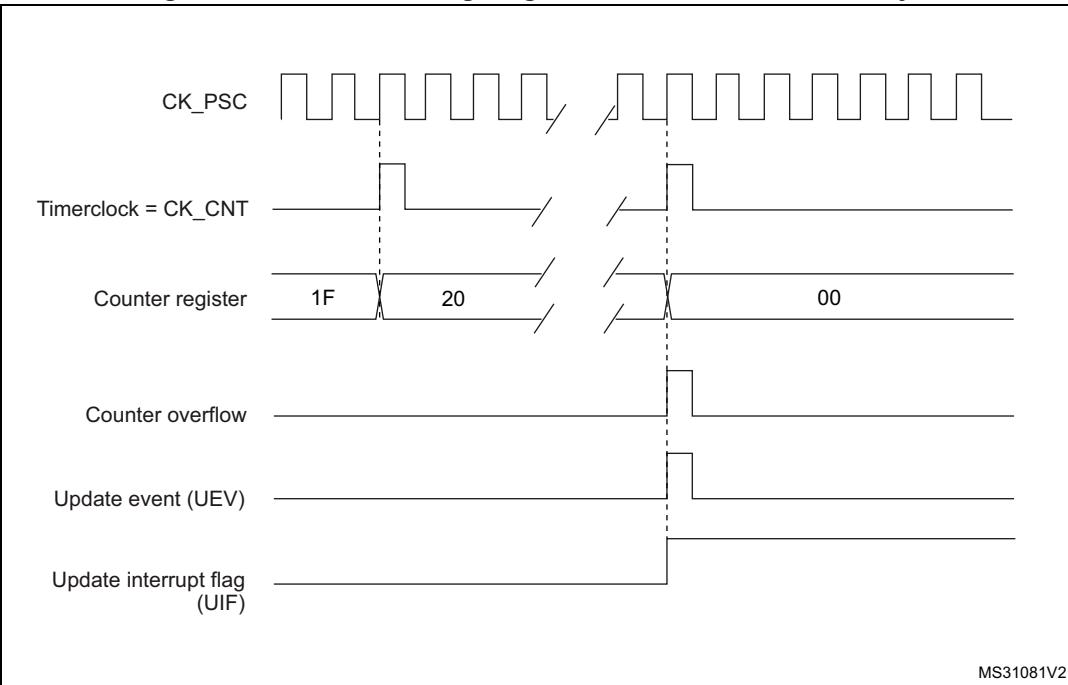
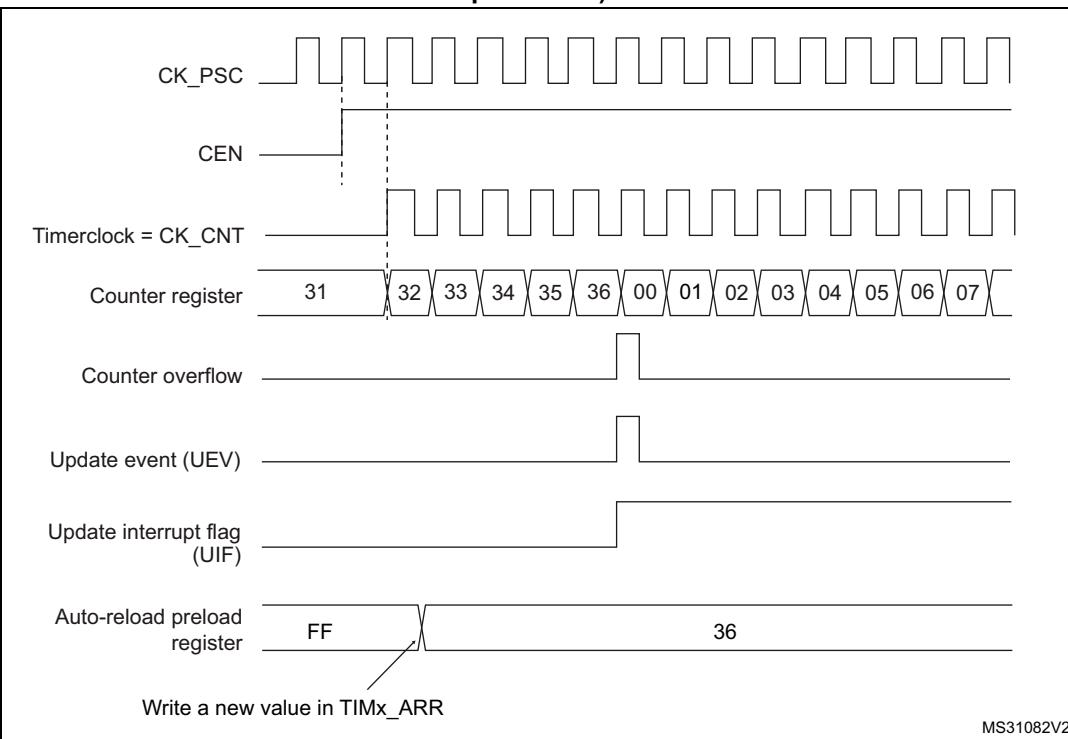
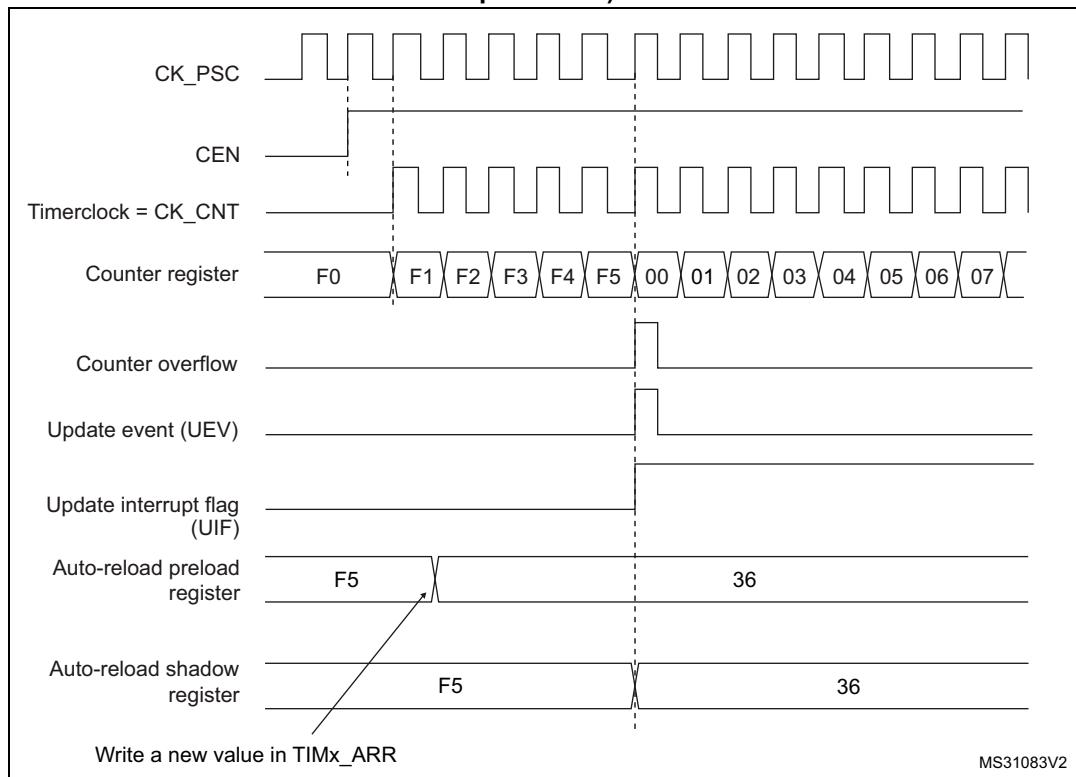
Figure 198. Counter timing diagram, internal clock divided by N**Figure 199. Counter timing diagram, Update event when ARPE=0 (TIMx_ARR not preloaded)**

Figure 200. Counter timing diagram, Update event when ARPE=1 (TIMx_ARR preloaded)



Downcounting mode

In downcounting mode, the counter counts from the auto-reload value (content of the TIMx_ARR register) down to 0, then restarts from the auto-reload value and generates a counter underflow event.

An Update event can be generated at each counter underflow or by setting the UG bit in the TIMx_EGR register (by software or by using the slave mode controller).

The UEV update event can be disabled by software by setting the UDIS bit in TIMx_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until UDIS bit has been written to 0. However, the counter restarts from the current auto-reload value, whereas the counter of the prescaler restarts from 0 (but the prescale rate doesn't change).

In addition, if the URS bit (update request selection) in TIMx_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

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

- The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSC register).
- The auto-reload active register is updated with the preload value (content of the TIMx_ARR register). Note that the auto-reload is updated before the counter is reloaded, so that the next period is the expected one.

The following figures show some examples of the counter behavior for different clock frequencies when TIMx_ARR=0x36.

Figure 201. Counter timing diagram, internal clock divided by 1

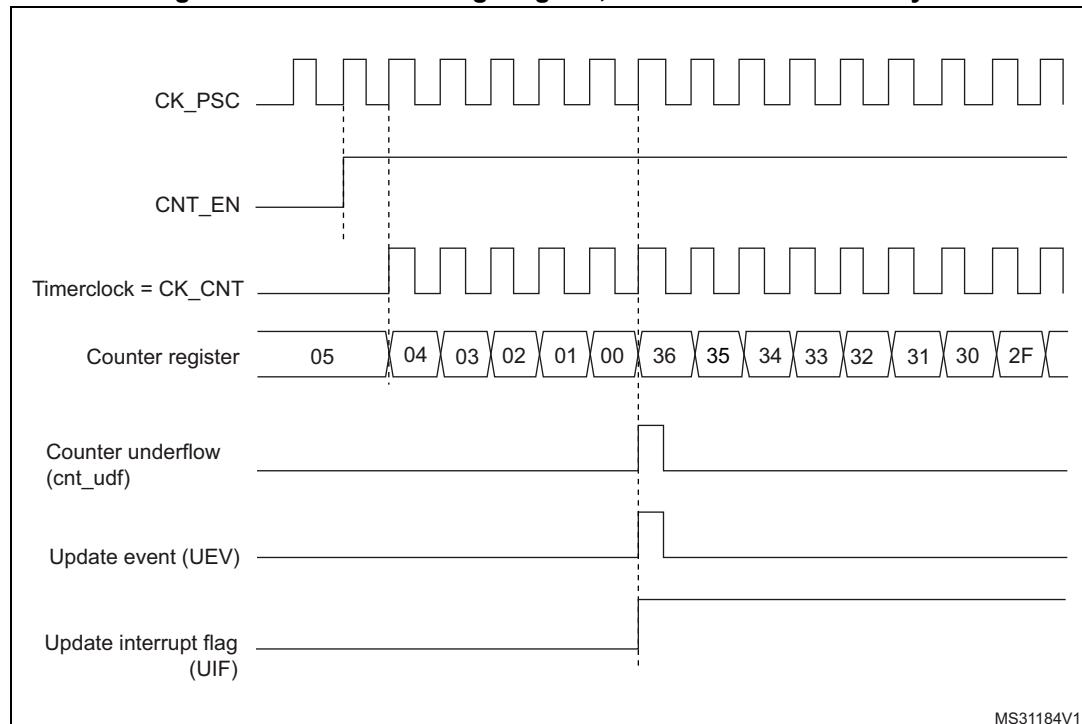


Figure 202. Counter timing diagram, internal clock divided by 2

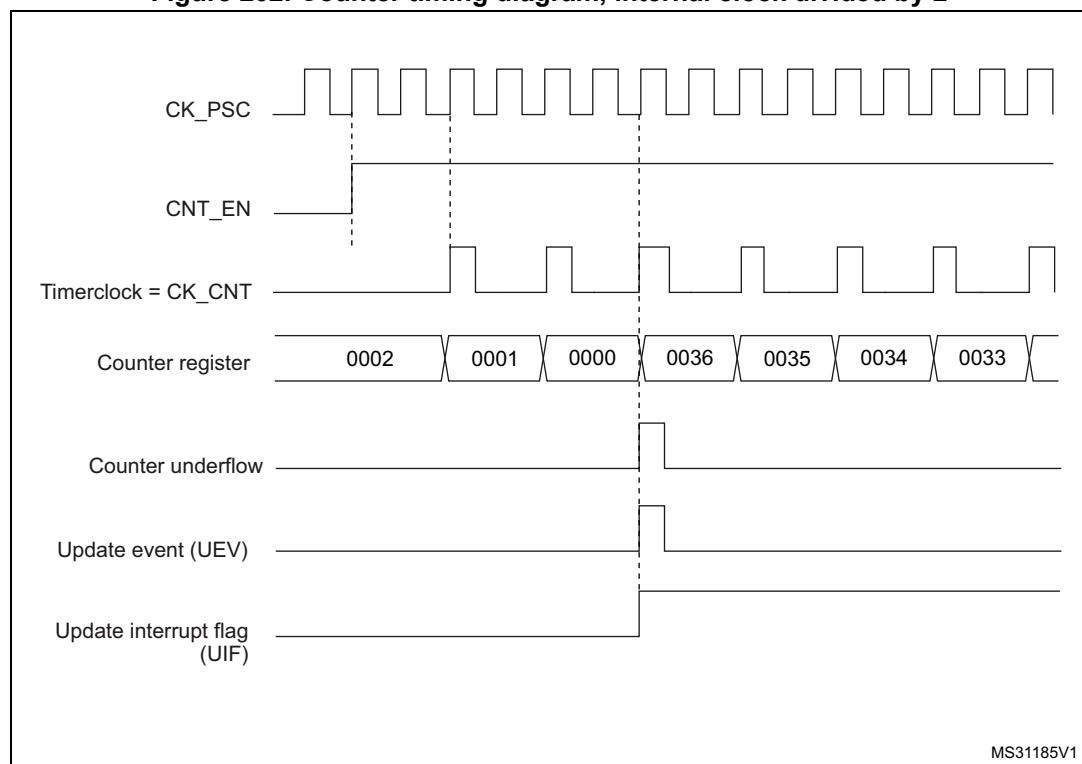


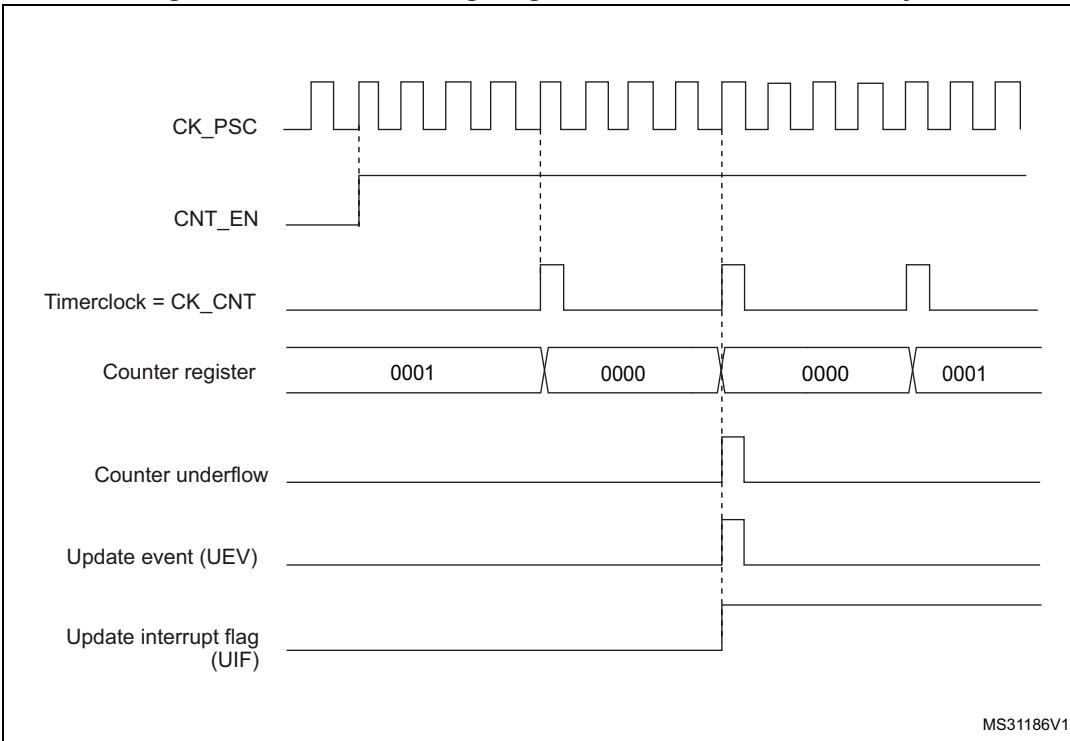
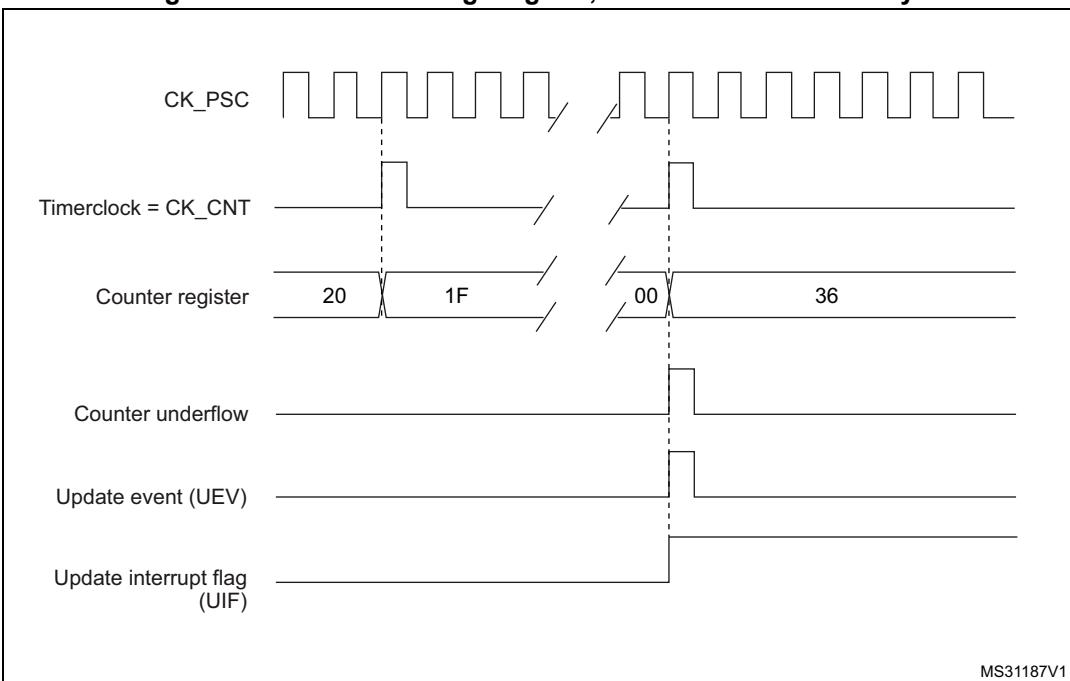
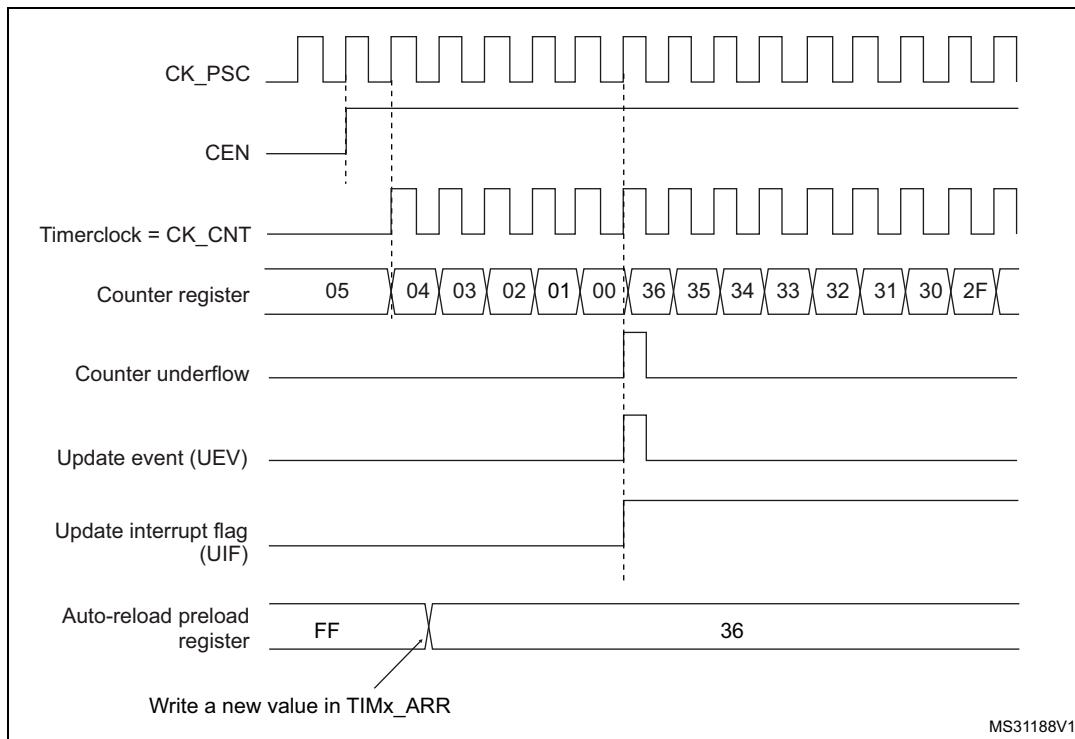
Figure 203. Counter timing diagram, internal clock divided by 4**Figure 204. Counter timing diagram, internal clock divided by N**

Figure 205. Counter timing diagram, Update event when repetition counter is not used



Center-aligned mode (up/down counting)

In center-aligned mode, the counter counts from 0 to the auto-reload value (content of the TIMx_ARR register) – 1, generates a counter overflow event, then counts from the auto-reload value down to 1 and generates a counter underflow event. Then it restarts counting from 0.

Center-aligned mode is active when the CMS bits in TIMx_CR1 register are not equal to '00'. The Output compare interrupt flag of channels configured in output is set when: the counter counts down (Center aligned mode 1, CMS = "01"), the counter counts up (Center aligned mode 2, CMS = "10") the counter counts up and down (Center aligned mode 3, CMS = "11").

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

The update event can be generated at each counter overflow and at each counter underflow or by setting the UG bit in the TIMx_EGR register (by software or by using the slave mode controller) also generates an update event. In this case, the counter restarts counting from 0, as well as the counter of the prescaler.

The UEV update event can be disabled by software by setting the UDIS bit in TIMx_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until the UDIS bit has been written to 0. However, the counter continues counting up and down, based on the current auto-reload value.

In addition, if the URS bit (update request selection) in TIMx_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or

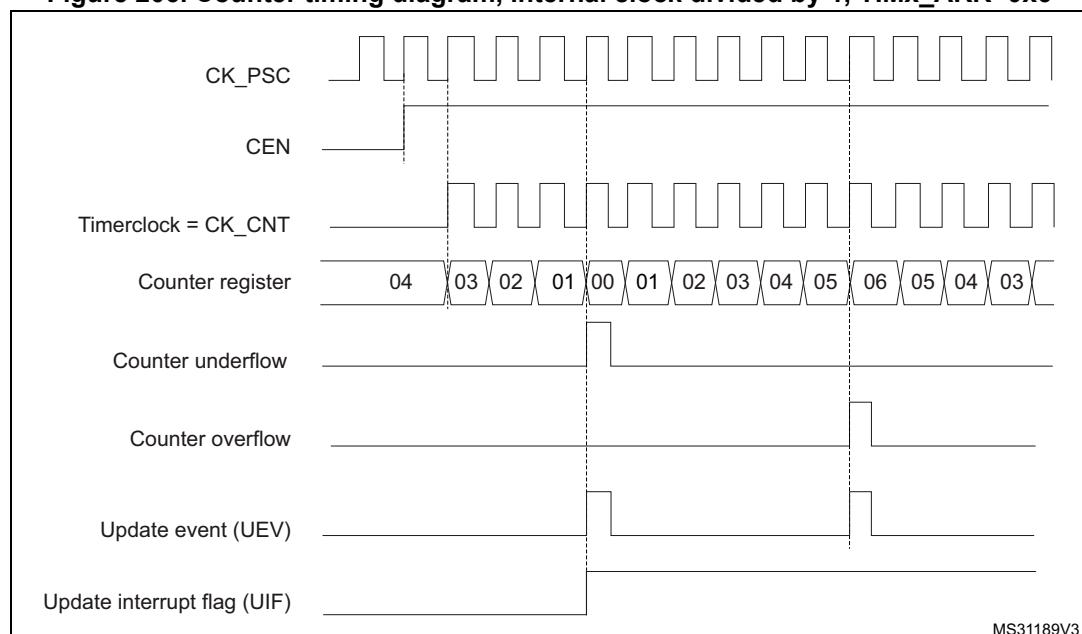
DMA request is sent). This is to avoid generating both update and capture interrupt when clearing the counter on the capture event.

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

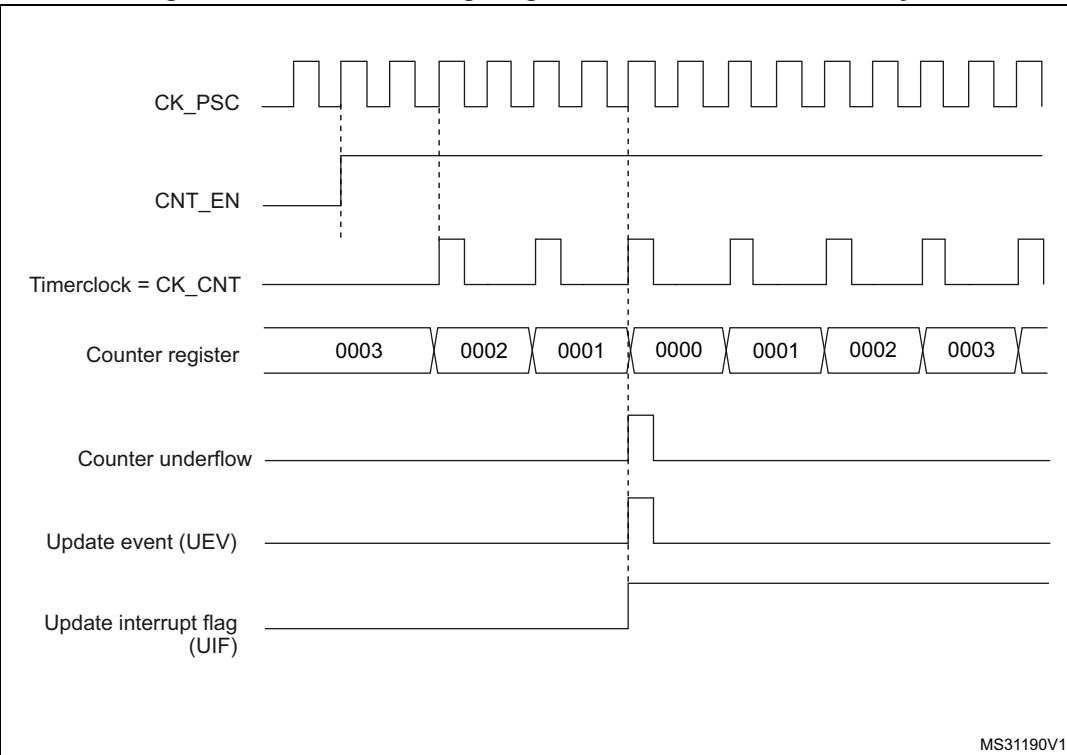
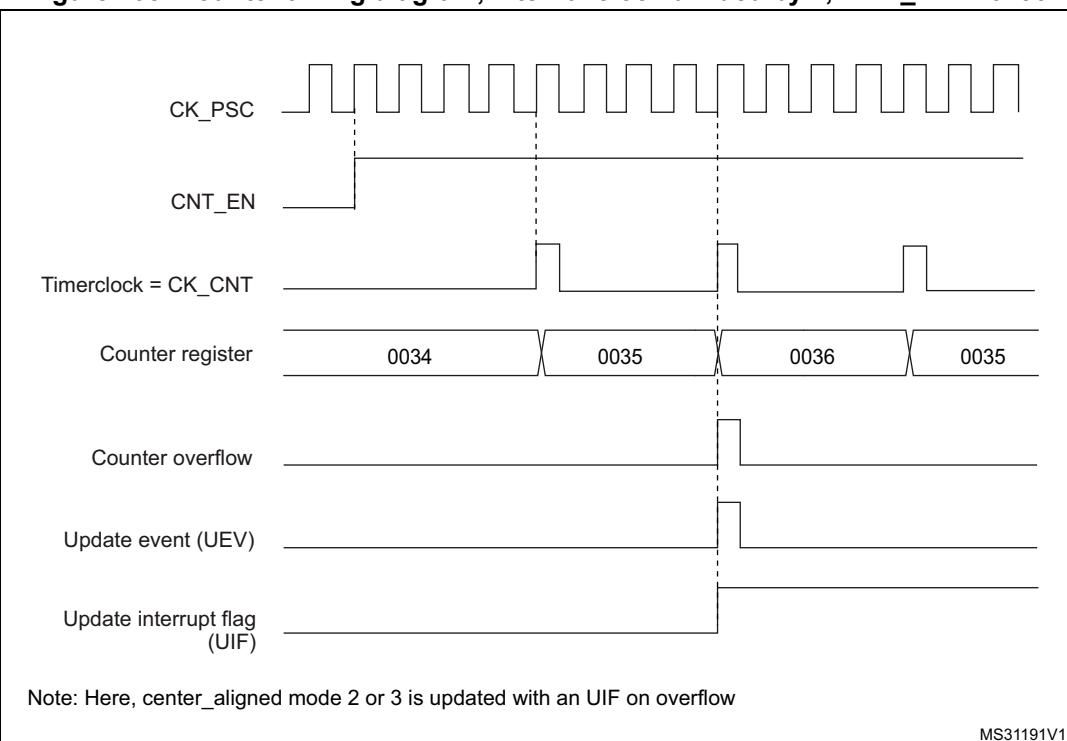
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSC register).
- The auto-reload active register is updated with the preload value (content of the TIMx_ARR register). Note that if the update source is a counter overflow, the auto-reload is updated before the counter is reloaded, so that the next period is the expected one (the counter is loaded with the new value).

The following figures show some examples of the counter behavior for different clock frequencies.

Figure 206. Counter timing diagram, internal clock divided by 1, TIMx_ARR=0x6



1. Here, center-aligned mode 1 is used (for more details refer to [Section 19.4.1: TIMx control register 1 \(TIMx_CR1\)\(x = 2 to 5\) on page 650](#)).

Figure 207. Counter timing diagram, internal clock divided by 2**Figure 208. Counter timing diagram, internal clock divided by 4, TIMx_ARR=0x36**

1. Center-aligned mode 2 or 3 is used with an UIF on overflow.

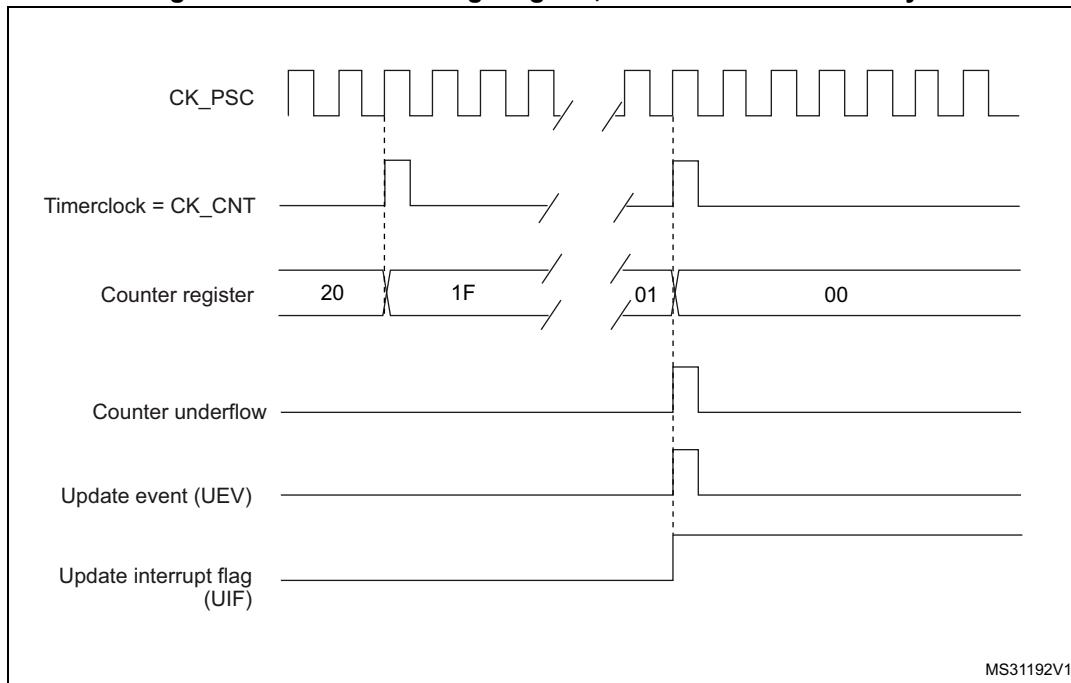
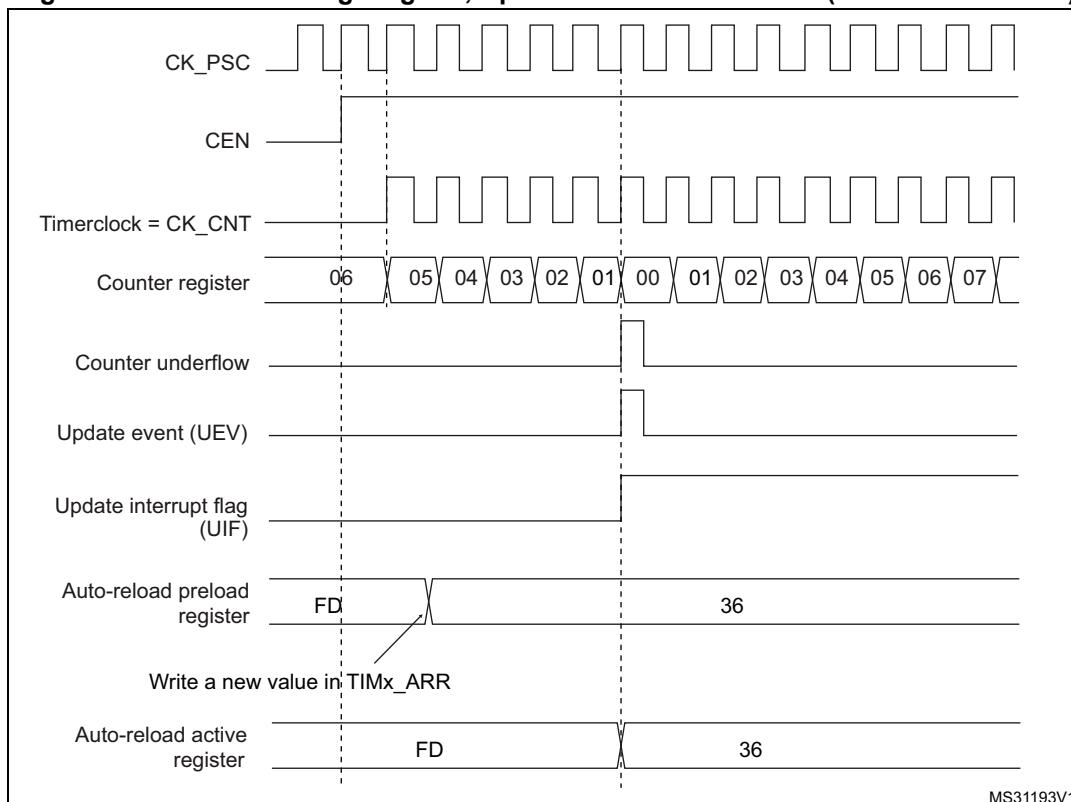
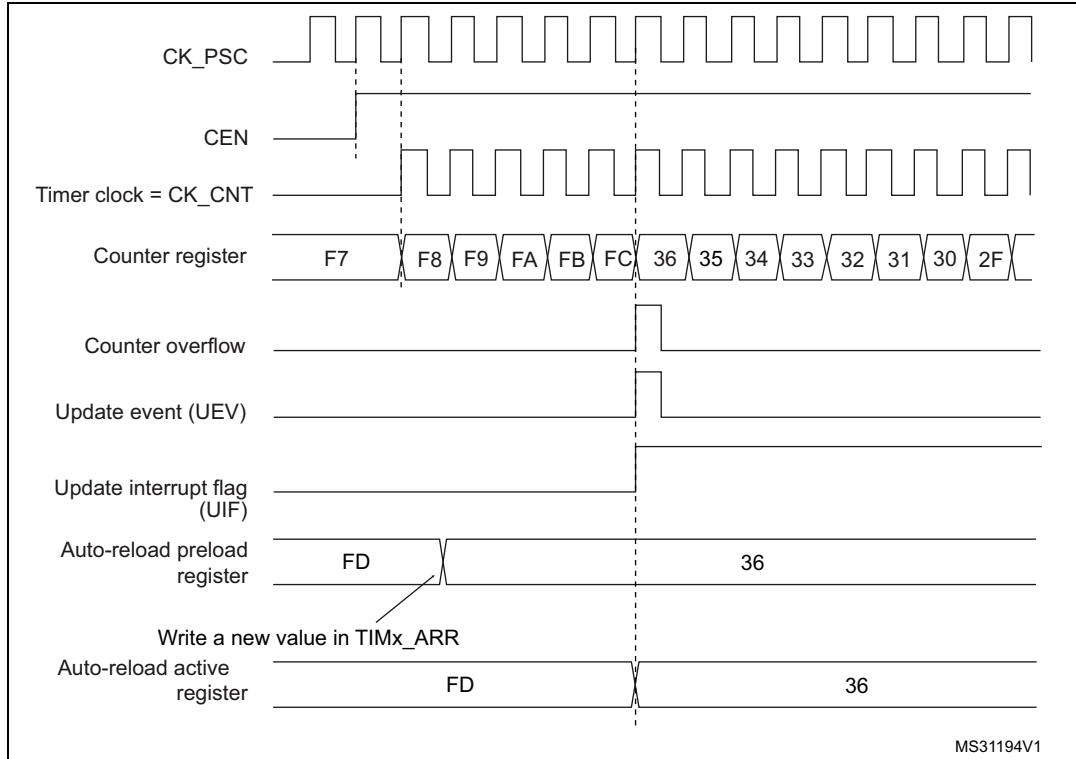
Figure 209. Counter timing diagram, internal clock divided by N**Figure 210. Counter timing diagram, Update event with ARPE=1 (counter underflow)**

Figure 211. Counter timing diagram, Update event with ARPE=1 (counter overflow)

19.3.3 Clock selection

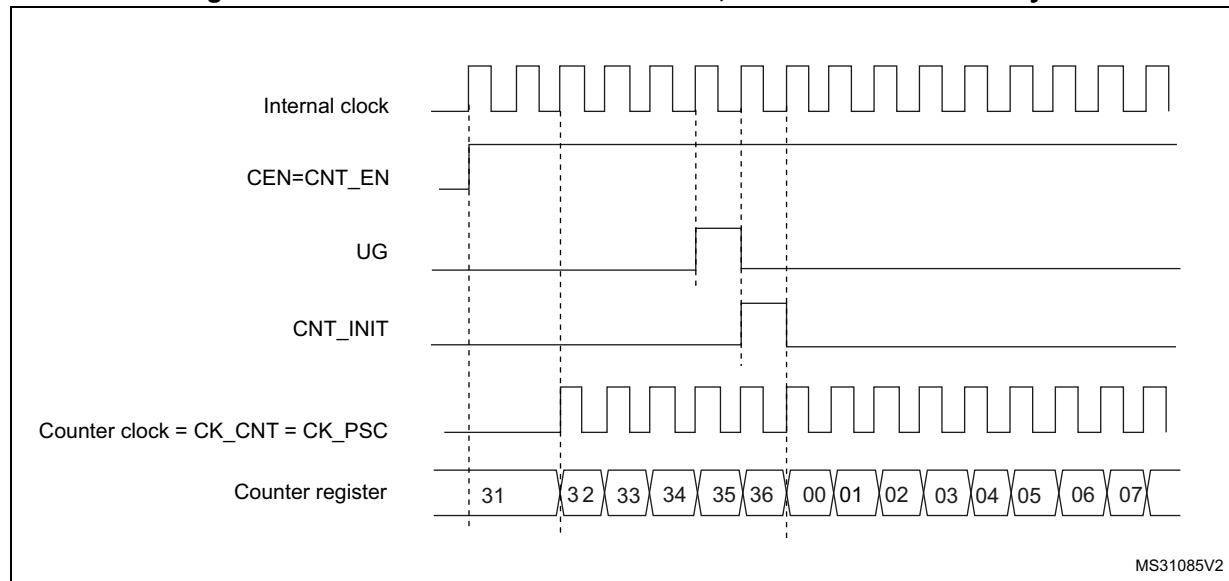
The counter clock can be provided by the following clock sources:

- 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 13 to act as a prescaler for Timer 2. Refer to : [Using one timer as prescaler for another timer on page 645](#) for more details.

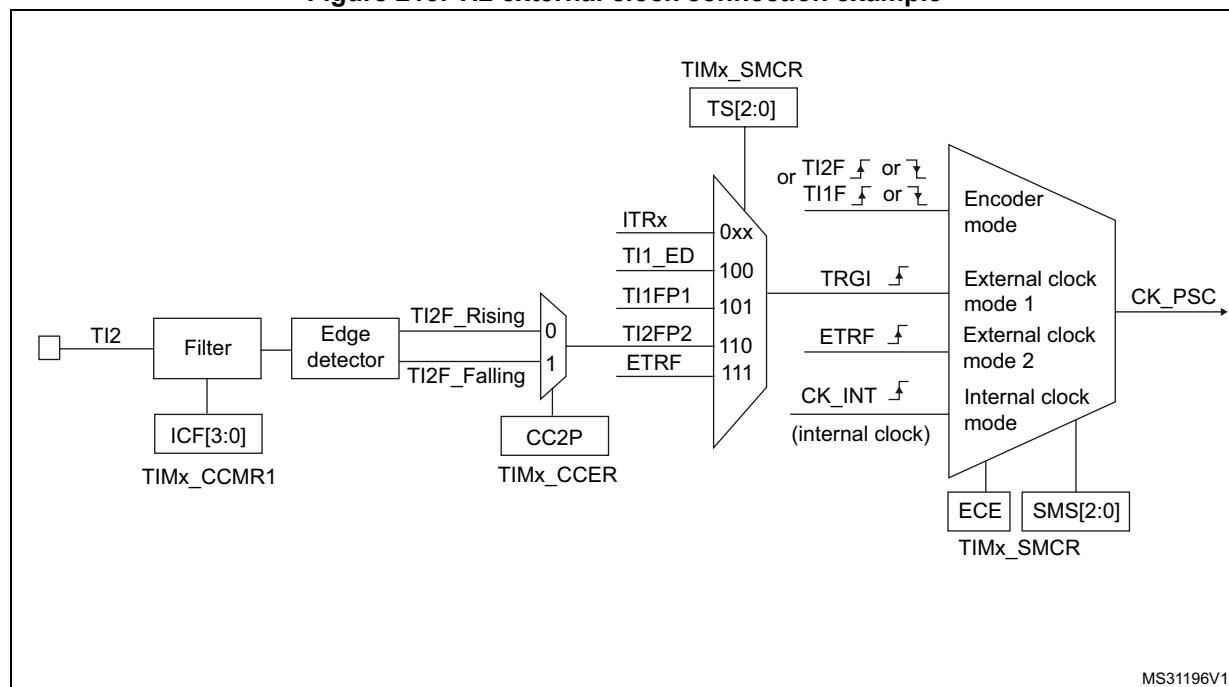
Internal clock source (CK_INT)

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 212](#) shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

Figure 212. 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 213. 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).

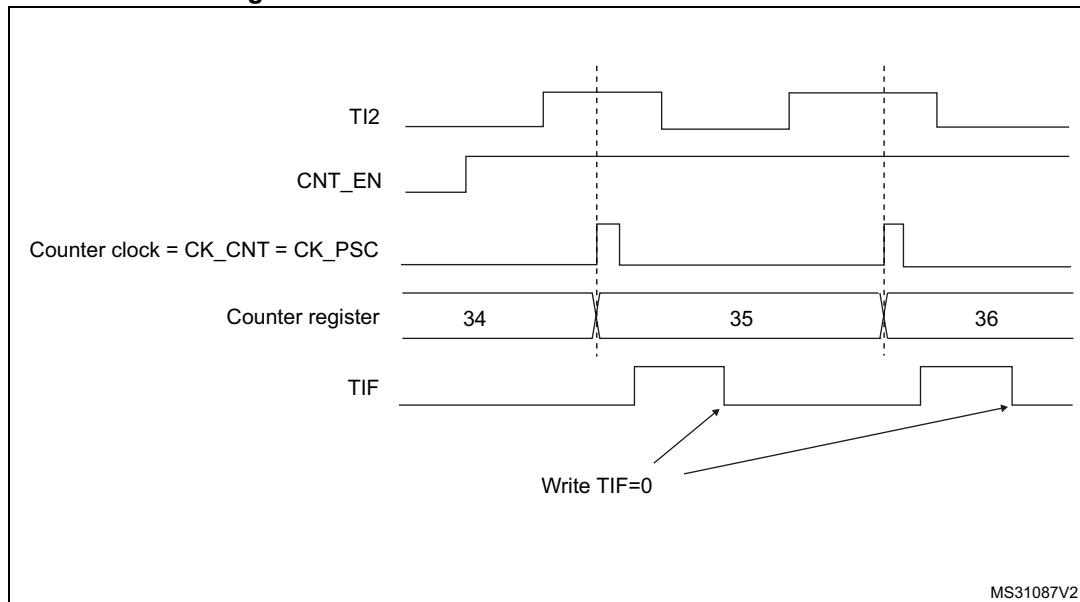
Note:

- 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 and CC2NP=0 and CC2NP=0 in the TIMx_CCER register.
 4. Configure the timer in external clock mode 1 by writing SMS=111 in the TIMx_SMCR register.
 5. Select TI2 as the input source by writing TS=110 in the TIMx_SMCR register.
 6. Enable the counter by writing CEN=1 in the TIMx_CR1 register.

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 214. Control circuit in external clock mode 1



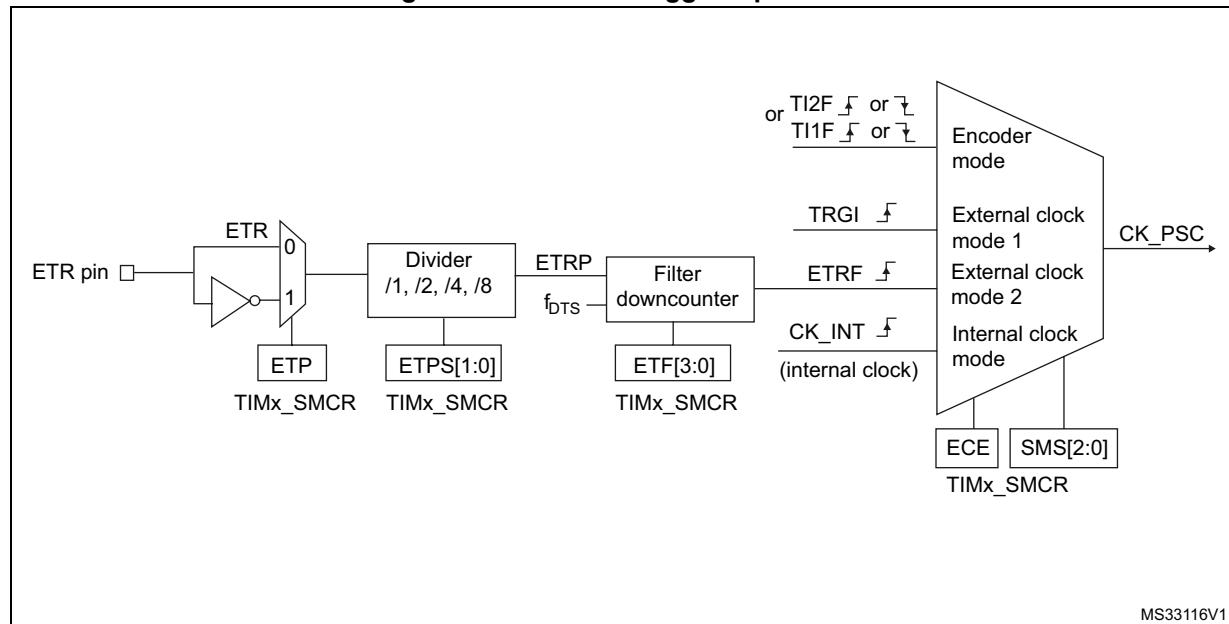
External clock source mode 2

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.

[Figure 215](#) gives an overview of the external trigger input block.

Figure 215. External trigger input block

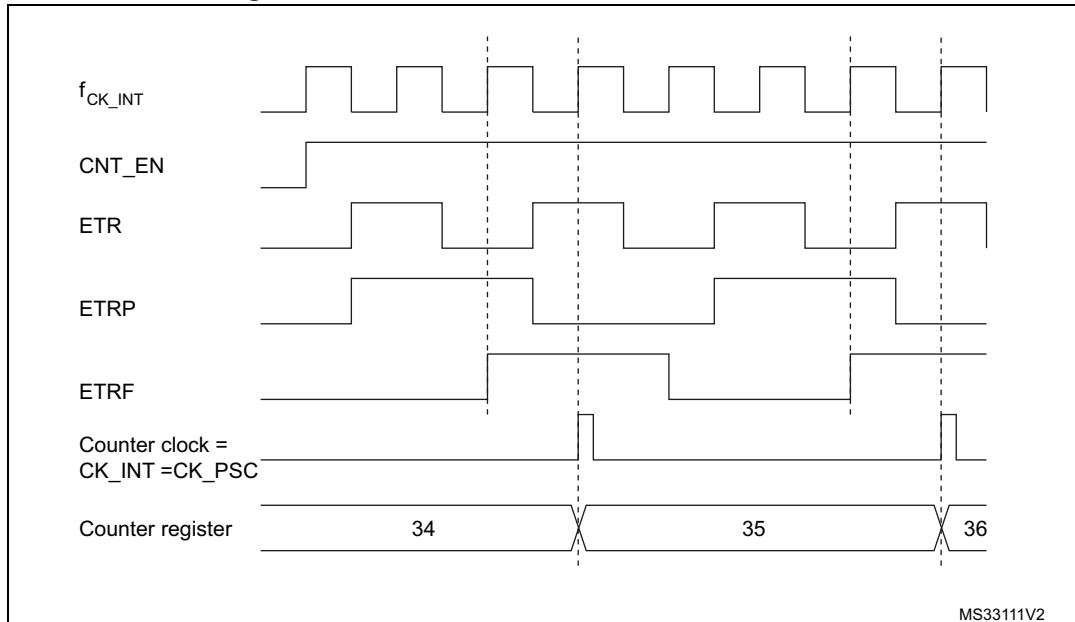


For example, to configure the upcounter to count each 2 rising edges on ETR, use the following procedure:

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 ETSPS[1:0]=01 in the TIMx_SMCR register
3. Select rising edge detection on the ETR pin by writing ETP=0 in the TIMx_SMCR register
4. Enable external clock mode 2 by writing ECE=1 in the TIMx_SMCR register.
5. Enable the counter by writing CEN=1 in the TIMx_CR1 register.

The counter counts once each 2 ETR rising edges.

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

Figure 216. Control circuit in external clock mode 2

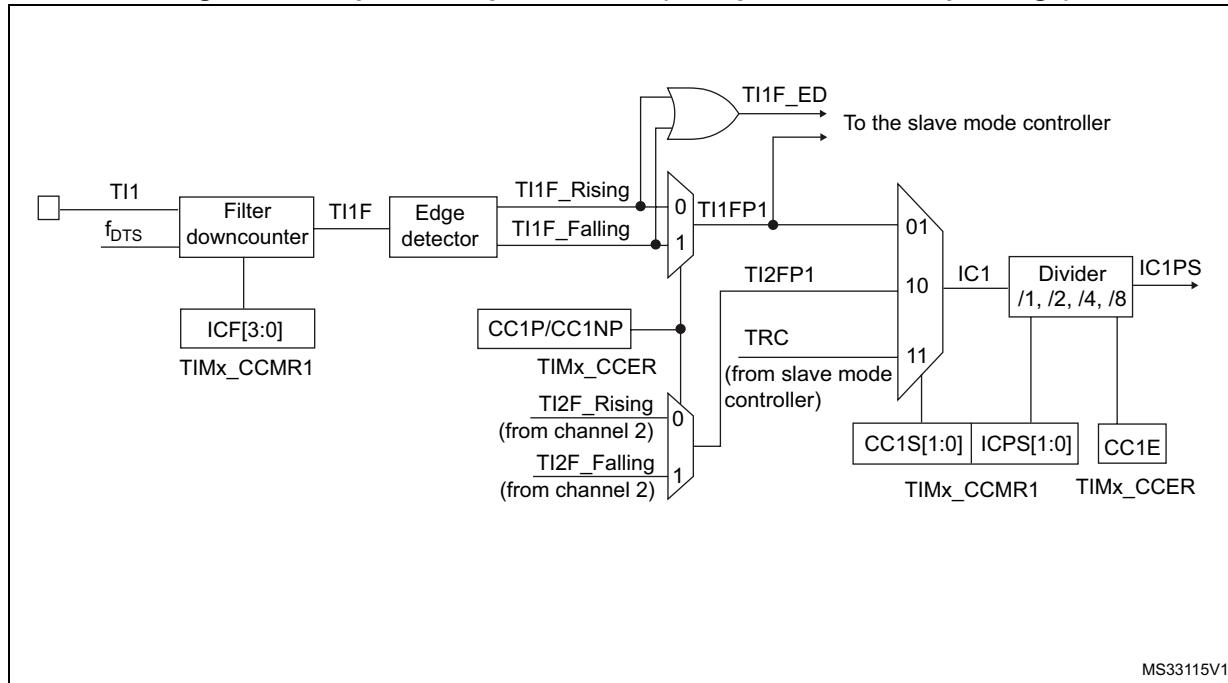
19.3.4 Capture/Compare channels

Each Capture/Compare channel is built around a capture/compare register (including a shadow register), a input stage for capture (with digital filter, multiplexing and prescaler) and an output stage (with comparator and output control).

The following figure gives an overview of one Capture/Compare channel.

The input stage samples the corresponding TIx input to generate a filtered signal $TIx F$. Then, an edge detector with polarity selection generates a signal ($TIx FPx$) 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 217. 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 218. Capture/Compare channel 1 main circuit

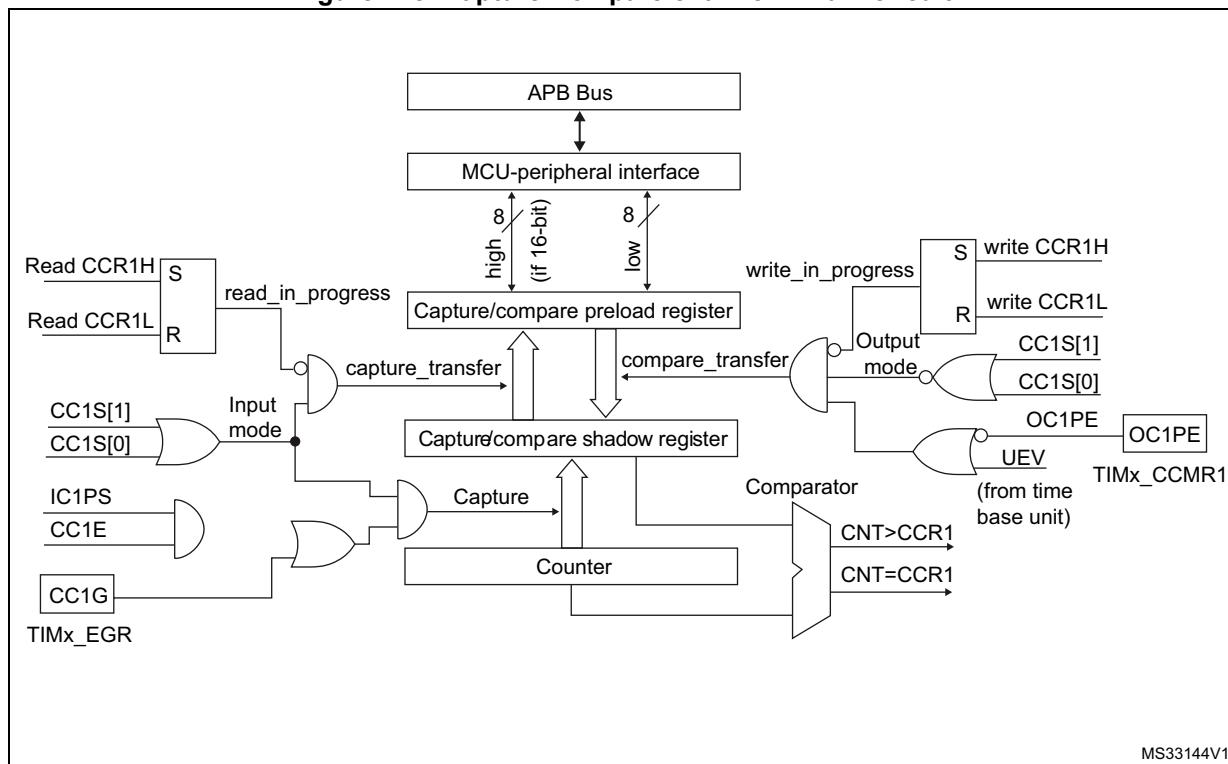
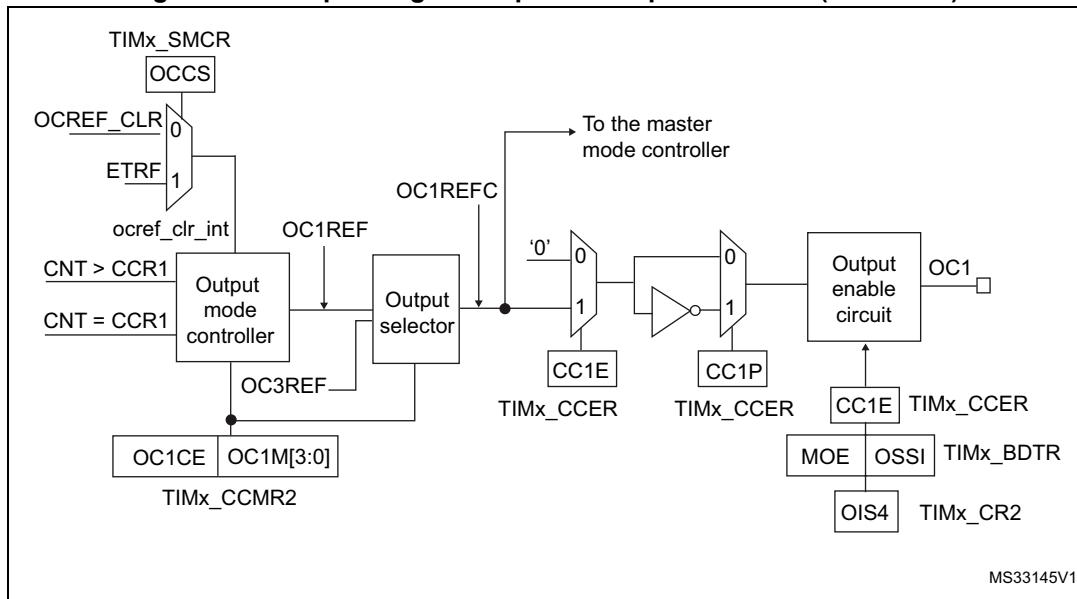


Figure 219. Output stage of Capture/Compare channel (channel 1)



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.

19.3.5 Input capture mode

In Input capture mode, the Capture/Compare Registers ($\text{TIMx_CCR}x$) are used to latch the value of the counter after a transition detected by the corresponding $\text{IC}x$ signal. When a capture occurs, the corresponding $\text{CC}x\text{IF}$ flag (TIMx_SR register) is set and an interrupt or a DMA request can be sent if they are enabled. If a capture occurs while the $\text{CC}x\text{IF}$ flag was already high, then the over-capture flag $\text{CC}x\text{OF}$ (TIMx_SR register) is set. $\text{CC}x\text{IF}$ can be cleared by software by writing it to 0 or by reading the captured data stored in the $\text{TIMx_CCR}x$ register. $\text{CC}x\text{OF}$ is cleared when you write it to 0.

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

1. Select the active input: 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.
 2. 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.

3. Select the edge of the active transition on the TI1 channel by writing the CC1P and CC1NP and CC1NP bits to 000 in the TIMx_CCER register (rising edge in this case).
4. Program the input prescaler. In our example, we wish the capture to be performed at each valid transition, so the prescaler is disabled (write IC1PS bits to 00 in the TIMx_CCMR1 register).
5. Enable capture from the counter into the capture register by setting the CC1E bit in the TIMx_CCER register.
6. If needed, enable the related interrupt request by setting the CC1IE bit in the TIMx_DIER register, 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.

19.3.6 PWM input mode

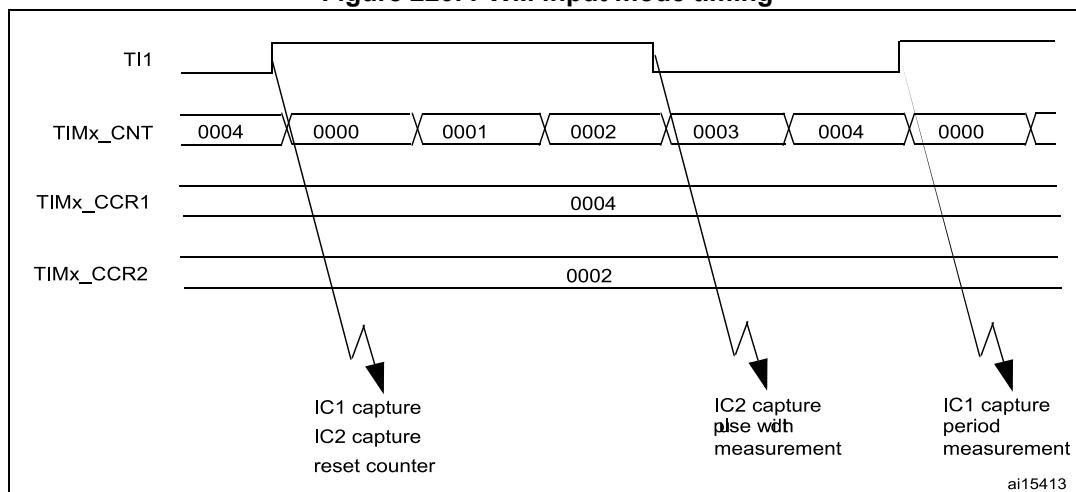
This mode is a particular case of input capture mode. The procedure is the same except:

- Two ICx signals are mapped on the same TIx input.
- These 2 ICx signals are active on edges with opposite polarity.
- One of the two TIxFP signals is selected as trigger input and the slave mode controller is configured in reset mode.

For example, 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):

1. Select the active input for TIMx_CCR1: write the CC1S bits to 01 in the TIMx_CCMR1 register (TI1 selected).
2. Select the active polarity for TI1FP1 (used both for capture in TIMx_CCR1 and counter clear): write the CC1P to '0' and the CC1NP bit to '0' (active on rising edge).
3. Select the active input for TIMx_CCR2: write the CC2S bits to 10 in the TIMx_CCMR1 register (TI1 selected).
4. Select the active polarity for TI1FP2 (used for capture in TIMx_CCR2): write the CC2P bit to '1' and the CC2NP bit to '0' (active on falling edge).
5. Select the valid trigger input: write the TS bits to 101 in the TIMx_SMCR register (TI1FP1 selected).
6. Configure the slave mode controller in reset mode: write the SMS bits to 100 in the TIMx_SMCR register.
7. Enable the captures: write the CC1E and CC2E bits to '1' in the TIMx_CCER register.

Figure 220. PWM input mode timing



1. The PWM input mode can be used only with the TIMx_CH1/TIMx_CH2 signals due to the fact that only TI1FP1 and TI2FP2 are connected to the slave mode controller.

19.3.7 Forced output mode

In output mode (CCxS bits = 00 in the TIMx_CCMRx register), each output compare signal (OCxREF and then OCx) can be forced to active or inactive level directly by software, independently of any comparison between the output compare register and the counter.

To force an output compare signal (ocxref/OCx) to its active level, you just need to write 101 in the OCxM bits in the corresponding TIMx_CCMRx register. Thus ocxref is forced high (OCxREF is always active high) and OCx get opposite value to CCxP polarity bit.

e.g.: CCxP=0 (OCx active high) => OCx is forced to high level.

ocxref signal can be forced low by writing the OCxM bits to 100 in the TIMx_CCMRx register.

Anyway, the comparison between the TIMx_CCRx shadow register and the counter is still performed and allows the flag to be set. Interrupt and DMA requests can be sent accordingly. This is described in the Output Compare Mode section.

19.3.8 Output compare mode

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

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

- Assigns the corresponding output pin to a programmable value defined by the output compare mode (OCxM bits in the TIMx_CCMRx register) and the output polarity (CCxP bit in the TIMx_CCER register). The output pin can keep its level (OCXM=000), be set active (OCXM=001), be set inactive (OCXM=010) or can toggle (OCXM=011) on match.
- Sets a flag in the interrupt status register (CCxIF bit in the TIMx_SR register).
- Generates an interrupt if the corresponding interrupt mask is set (CCXIE bit in the TIMx_DIER register).
- Sends a DMA request if the corresponding enable bit is set (CCxDE bit in the TIMx_DIER register, CCDS bit in the TIMx_CR2 register for the DMA request selection).

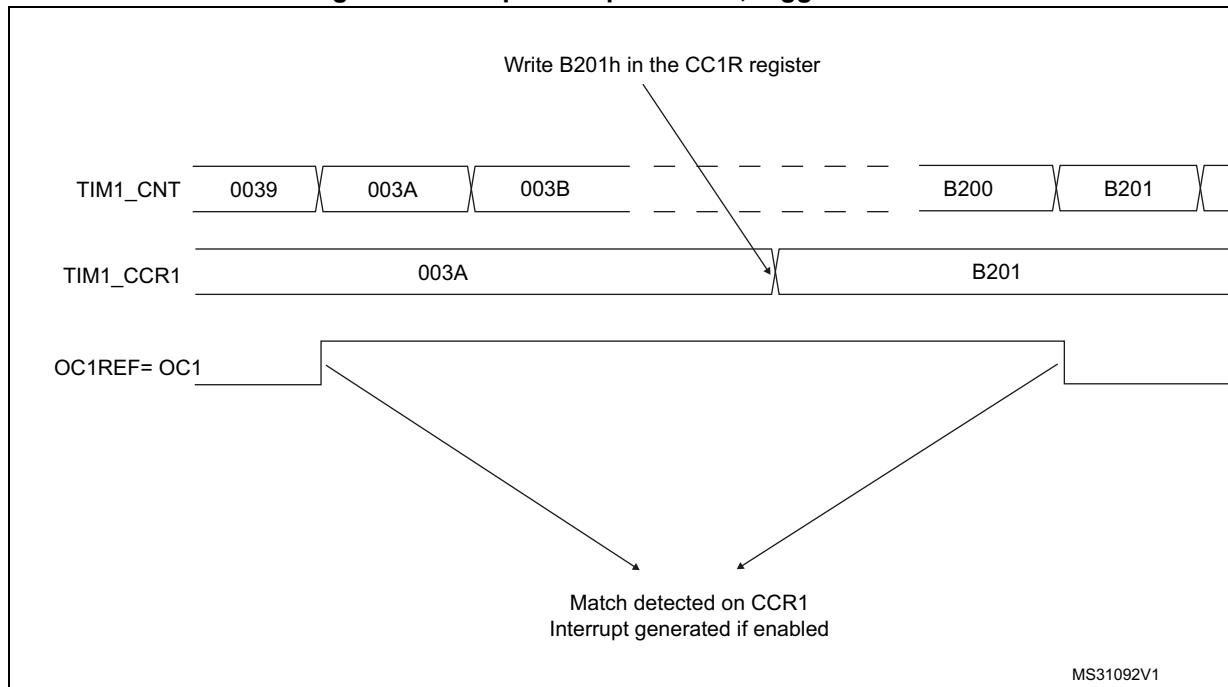
The TIMx_CCRx registers can be programmed with or without preload registers using the OCxPE bit in the TIMx_CCMRx register.

In output compare mode, the update event UEV has no effect on ocxref and OCx output. The timing resolution is one count of the counter. Output compare mode can also be used to output a single pulse (in One-pulse mode).

Procedure

1. Select the counter clock (internal, external, prescaler).
2. Write the desired data in the TIMx_ARR and TIMx_CCRx registers.
3. Set the CCxIE and/or CCxDE bits if an interrupt and/or a DMA request is to be generated.
4. Select the output mode. For example, 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 221](#).

Figure 221. Output compare mode, toggle on OC1

19.3.9 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_CCRx register.

The PWM mode can be selected independently on each channel (one PWM per OCx output) by writing 110 (PWM mode 1) or '111 (PWM mode 2) in the OCxM bits in the TIMx_CCMRx register. 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_CCMRx register description for more details.

In PWM mode (1 or 2), TIMx_CNT and TIMx_CCRx are always compared to determine whether TIMx_CCRx ≤ TIMx_CNT or TIMx_CNT ≤ TIMx_CCRx (depending on the direction of the counter). However, to comply with the OCREF_CLR functionality (OCREF can be cleared by an external event through the ETR signal until the next PWM period), the OCREF signal is asserted only:

- When the result of the comparison or
- When the output compare mode (OCxM bits in TIMx_CCMRx register) switches from the “frozen” configuration (no comparison, OCxM='000) to one of the PWM modes (OCxM='110 or '111).

This forces the PWM by software while the timer is running.

The timer is able to generate PWM in edge-aligned mode or center-aligned mode depending on the CMS bits in the TIMx_CR1 register.

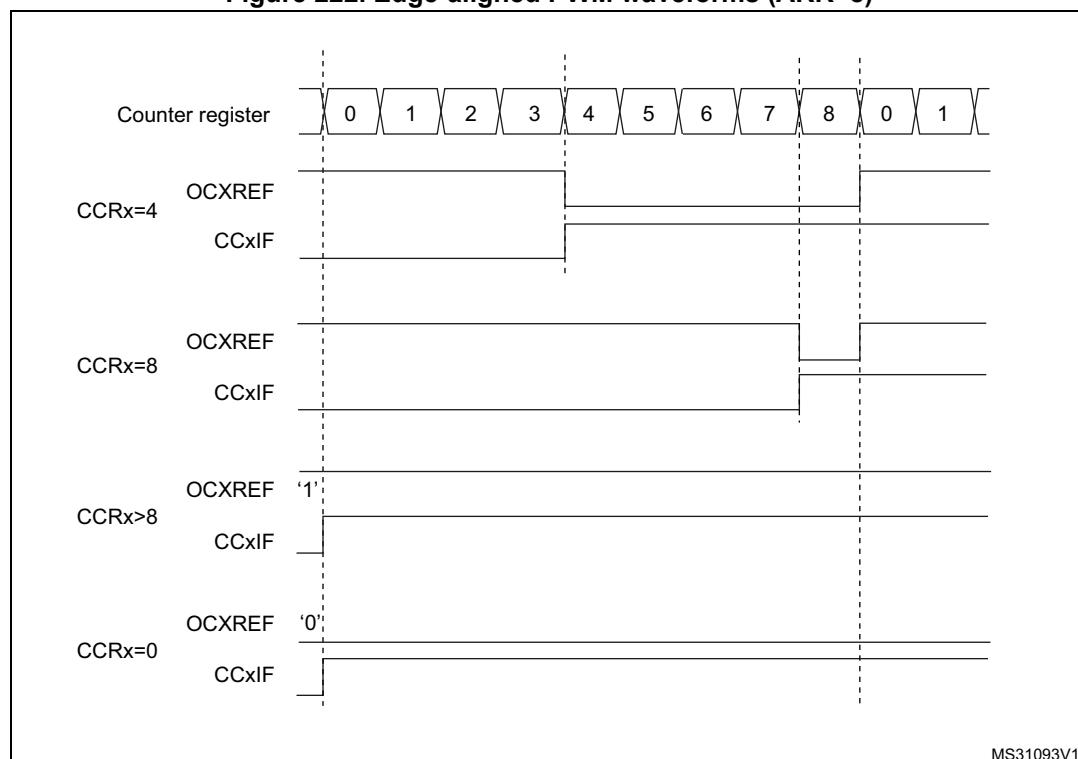
PWM edge-aligned mode

Upcounting configuration

Upcounting is active when the DIR bit in the TIMx_CR1 register is low. Refer to [Upcounting mode on page 609](#).

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 222](#) shows some edge-aligned PWM waveforms in an example where TIMx_ARR=8.

Figure 222. 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 612](#).

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 100%. PWM is not possible in this mode.

PWM center-aligned mode

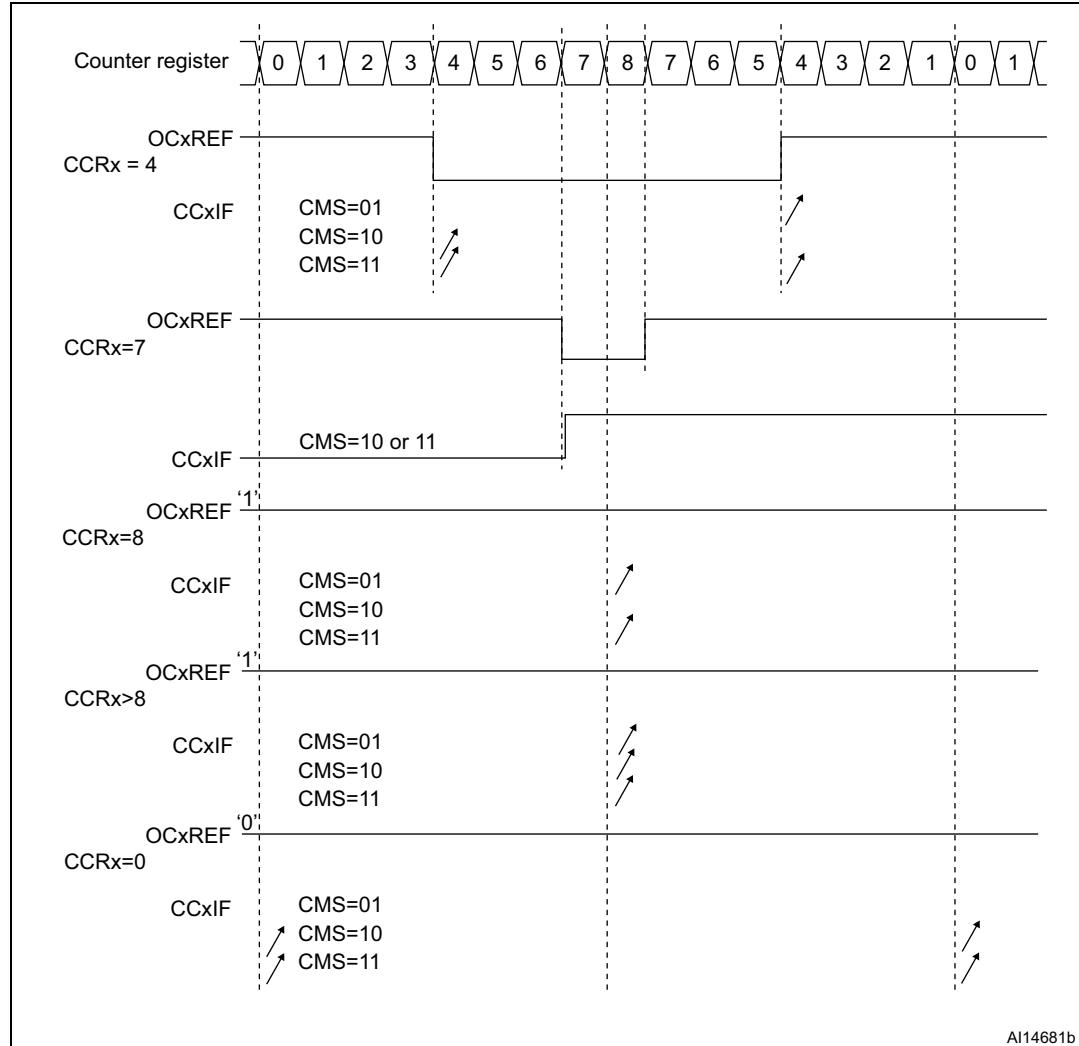
Center-aligned mode is active when the CMS bits in TIMx_CR1 register are different from '00 (all the remaining configurations having the same effect on the ocxref/OCx signals). The

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 [Center-aligned mode \(up/down counting\) on page 615](#).

[Figure 223](#) 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 223. Center-aligned PWM waveforms (ARR=8)



AI14681b

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.

19.3.10 Asymmetric PWM mode

Asymmetric mode allows two center-aligned PWM signals to be generated with a programmable phase shift. While the frequency is determined by the value of the TIMx_ARR register, the duty cycle and the phase-shift are determined by a pair of TIMx_CCRx registers. One register controls the PWM during up-counting, the second during down counting, so that PWM is adjusted every half PWM cycle:

- OC1REFC (or OC2REFC) is controlled by TIMx_CCR1 and TIMx_CCR2
- OC3REFC (or OC4REFC) is controlled by TIMx_CCR3 and TIMx_CCR4

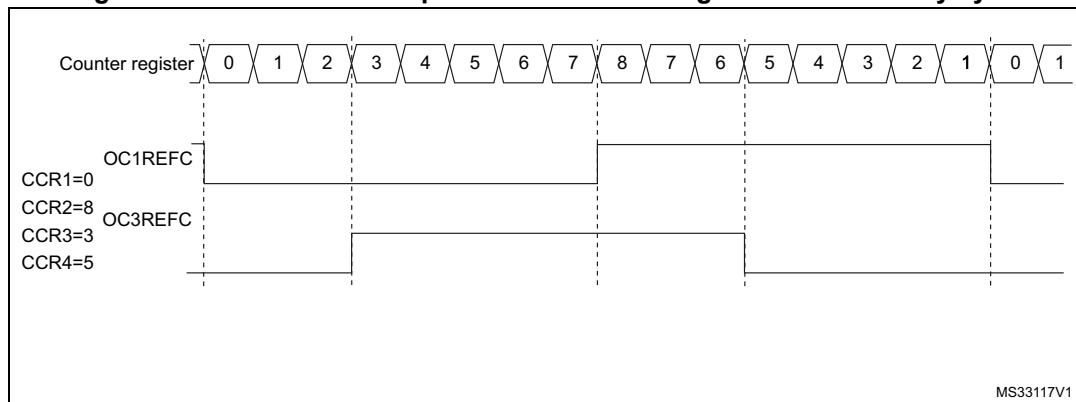
Asymmetric PWM mode can be selected independently on two channels (one OCx output per pair of CCR registers) by writing '1110' (Asymmetric PWM mode 1) or '1111' (Asymmetric PWM mode 2) in the OCxM bits in the TIMx_CCMRx register.

Note: The OCxM[3:0] bit field is split into two parts for compatibility reasons, the most significant bit is not contiguous with the 3 least significant ones.

When a given channel is used as asymmetric PWM channel, its secondary channel can also be used. For instance, if an OC1REFC signal is generated on channel 1 (Asymmetric PWM mode 1), it is possible to output either the OC2REF signal on channel 2, or an OC2REFC signal resulting from asymmetric PWM mode 2.

Figure 224 shows an example of signals that can be generated using Asymmetric PWM mode (channels 1 to 4 are configured in Asymmetric PWM mode 1).

Figure 224. Generation of 2 phase-shifted PWM signals with 50% duty cycle



19.3.11 Combined PWM mode

Combined PWM mode allows two edge or center-aligned PWM signals to be generated with programmable delay and phase shift between respective pulses. While the frequency is determined by the value of the TIMx_ARR register, the duty cycle and delay are determined by the two TIMx_CCRx registers. The resulting signals, OCxREFC, are made of an OR or AND logical combination of two reference PWMs:

- OC1REFC (or OC2REFC) is controlled by TIMx_CCR1 and TIMx_CCR2
- OC3REFC (or OC4REFC) is controlled by TIMx_CCR3 and TIMx_CCR4

Combined PWM mode can be selected independently on two channels (one OCx output per pair of CCR registers) by writing ‘1100’ (Combined PWM mode 1) or ‘1101’ (Combined PWM mode 2) in the OCxM bits in the TIMx_CCMRx register.

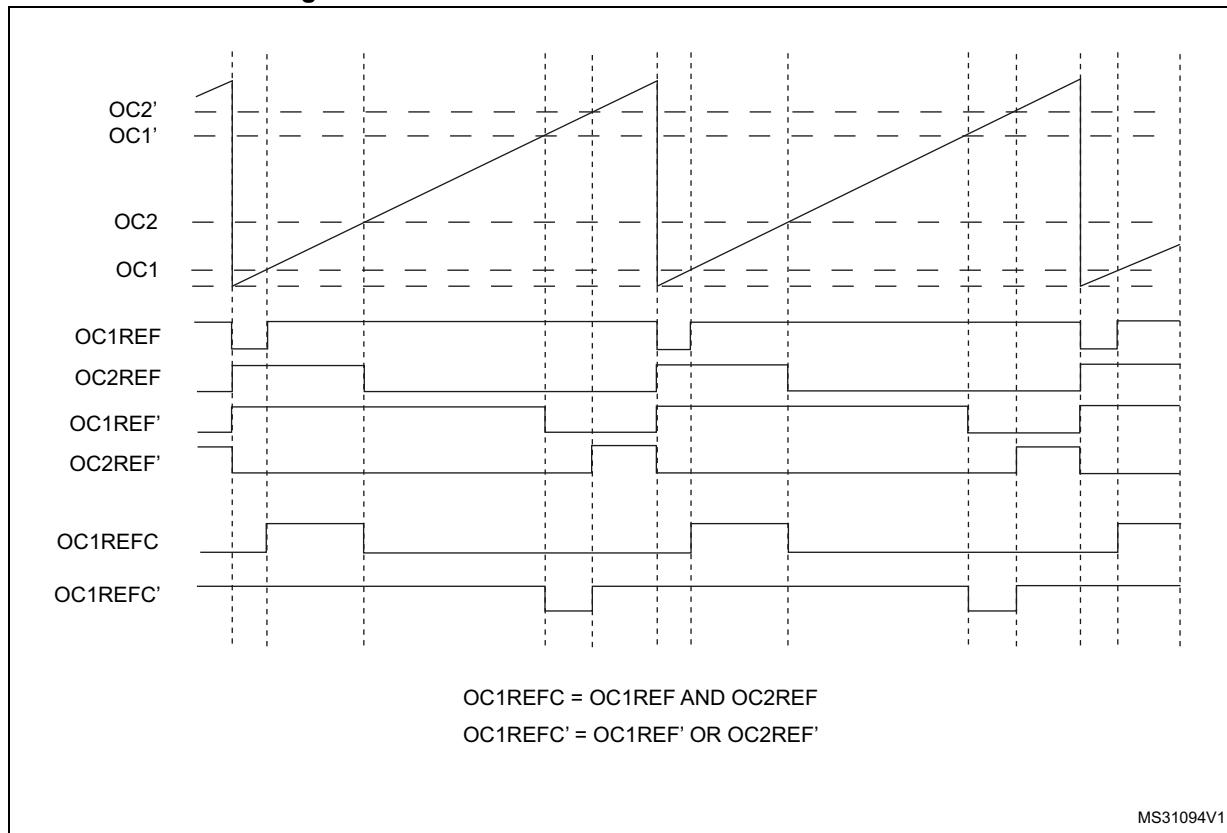
When a given channel is used as combined PWM channel, its secondary channel must be configured in the opposite PWM mode (for instance, one in Combined PWM mode 1 and the other in Combined PWM mode 2).

Note: *The OCxM[3:0] bit field is split into two parts for compatibility reasons, the most significant bit is not contiguous with the 3 least significant ones.*

Figure 225 shows an example of signals that can be generated using Asymmetric PWM mode, obtained with the following configuration:

- Channel 1 is configured in Combined PWM mode 2,
- Channel 2 is configured in PWM mode 1,
- Channel 3 is configured in Combined PWM mode 2,
- Channel 4 is configured in PWM mode 1

Figure 225. Combined PWM mode on channels 1 and 3



19.3.12 Clearing the OCxREF signal on an external event

The OCxREF signal of a given channel can be cleared when a high level is applied on the `ocref_clr_int` input (OCxCE enable bit in the corresponding `TIMx_CCMRx` register set to 1). OCxREF remains low until the next update event (UEV) occurs. This function can only be used in Output compare and PWM modes. It does not work in Forced mode.

`OCREF_CLR_INPUT` can be selected between the `OCREF_CLR` input and `ETRF` (`ETR` after the filter) by configuring the `OCCS` bit in the `TIMx_SMCR` register.

The OCxREF signal for a given channel can be reset by applying a high level on the `ETRF` input (OCxCE enable bit set to 1 in the corresponding `TIMx_CCMRx` register). OCxREF remains low until the next update event (UEV) occurs.

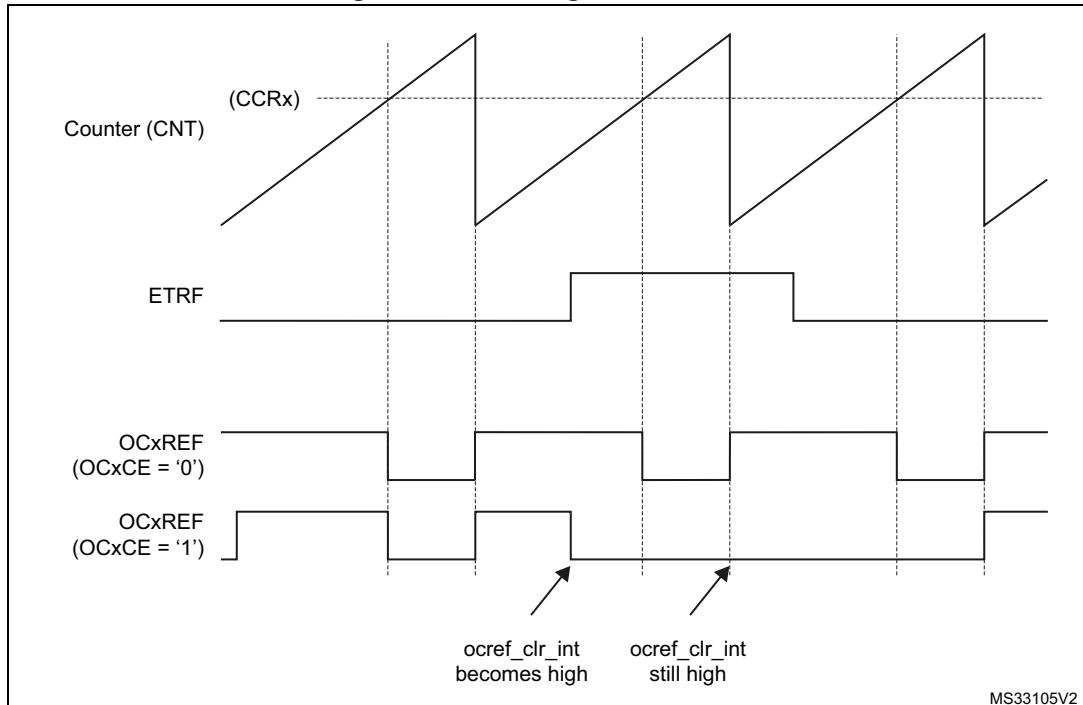
This function can be used only in the output compare and PWM modes. It does not work in forced mode.

For example, the OCxREF signal can be connected to the output of a comparator to be used for current handling. In this case, ETR must be configured as follows:

1. The external trigger prescaler should be kept off: bits `ETPS[1:0]` in the `TIMx_SMCR` register are cleared to 00.
2. The external clock mode 2 must be disabled: bit `ECE` in the `TIM1_SMCR` register is cleared to 0.
3. The external trigger polarity (ETP) and the external trigger filter (ETF) can be configured according to the application's needs.

Figure 226 shows the behavior of the OC_xREF signal when the ETRF input becomes high, for both values of the OC_xCE enable bit. In this example, the timer TIMx is programmed in PWM mode.

Figure 226. Clearing TIMx OC_xREF



Note: *In case of a PWM with a 100% duty cycle (if CCR_x>ARR), OC_xREF is enabled again at the next counter overflow.*

19.3.13 One-pulse mode

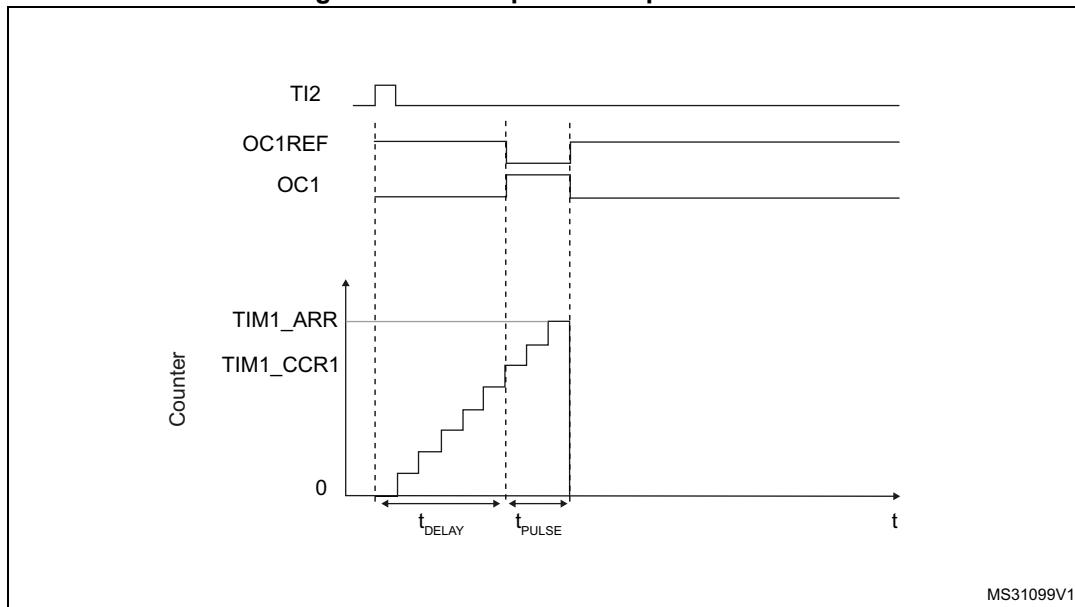
One-pulse mode (OPM) is a particular case of the previous modes. It allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length after a programmable delay.

Starting the counter can be controlled through the slave mode controller. Generating the waveform can be done in output compare mode or PWM mode. 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:

- CNT<CCR_x ≤ ARR (in particular, 0<CCR_x),

Figure 227. Example of one-pulse mode.



MS31099V1

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:

1. Map TI2FP2 on TI2 by writing CC2S=01 in the TIMx_CCMR1 register.
2. TI2FP2 must detect a rising edge, write CC2P=0 and CC2NP='0' in the TIMx_CCER register.
3. Configure TI2FP2 as trigger for the slave mode controller (TRGI) by writing TS=110 in the TIMx_SMCR register.
4. 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 (Single mode), 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). When OPM bit in the TIMx_CR1 register is set to '0', so the Repetitive Mode is selected.

Particular case: OCx fast enable:

In One-pulse mode, the edge detection on TIx input set the CEN bit which enables the counter. Then the comparison between the counter and the compare value makes the output toggle. But several clock cycles are needed for these operations and it limits the minimum delay t_{DELAY} min we can get.

If you want to output a waveform with the minimum delay, you can set the OCxFE bit in the TIMx_CCMRx register. Then OCxRef (and OCx) 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. OCxFE acts only if the channel is configured in PWM1 or PWM2 mode.

19.3.14 Retriggerable one pulse mode (OPM)

This mode allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length, but with the following differences with Non-retriggerable one pulse mode described in [Section 19.3.13](#):

- The pulse starts as soon as the trigger occurs (no programmable delay)
- The pulse is extended if a new trigger occurs before the previous one is completed

The timer must be in Slave mode, with the bits SMS[3:0] = '1000' (Combined Reset + trigger mode) in the TIMx_SMCR register, and the OCxM[3:0] bits set to '1000' or '1001' for Retriggerable OPM mode 1 or 2.

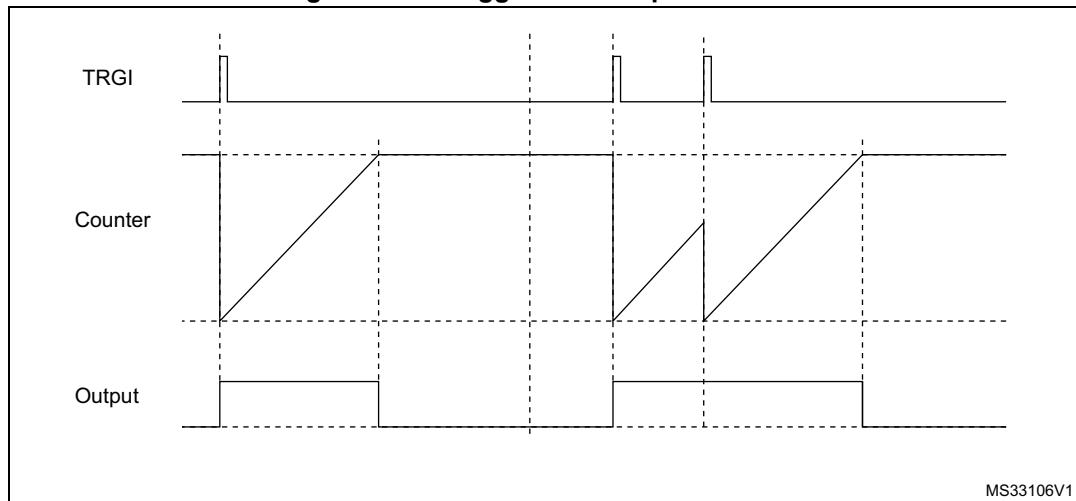
If the timer is configured in Up-counting mode, the corresponding CCRx must be set to 0 (the ARR register sets the pulse length). If the timer is configured in Down-counting mode CCRx must be above or equal to ARR.

Note: In retriggerable one pulse mode, the CCxIF flag is not significant.

The OCxM[3:0] and SMS[3:0] bit fields are split into two parts for compatibility reasons, the most significant bit is not contiguous with the 3 least significant ones.

This mode must not be used with center-aligned PWM modes. It is mandatory to have CMS[1:0] = 00 in TIMx_CR1.

Figure 228 Retriggerable one pulse mode



19.3.15 Encoder interface mode

To select Encoder Interface mode write SMS='001 in the TIMx_SMCR register if the counter is counting on TI2 edges only, SMS=010 if it is counting on TI1 edges only and SMS=011 if it is counting on both TI1 and TI2 edges.

Select the TI1 and TI2 polarity by programming the CC1P and CC2P bits in the TIMx_CCER register. CC1NP and CC2NP must be kept cleared. When needed, you can program the input filter as well. CC1NP and CC2NP must be kept low.

The two inputs TI1 and TI2 are used to interface to an incremental encoder. Refer to [Table 118](#). 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 quadrature 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 118. Counting direction versus encoder signals

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

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

Figure 229 gives an example of counter operation, showing count signal generation and direction control. It also shows how input jitter is compensated where both edges are selected. This might occur if the sensor is positioned near to one of the switching points. For this example we assume that the configuration is the following:

- CC1S= 01 (TIMx_CCMR1 register, TI1FP1 mapped on TI1)
- CC2S= 01 (TIMx_CCMR2 register, TI2FP2 mapped on TI2)
- CC1P and CC1NP = '0' (TIMx_CCER register, TI1FP1 noninverted, TI1FP1=TI1)
- CC2P and CC2NP = '0' (TIMx_CCER register, TI2FP2 noninverted, TI2FP2=TI2)
- SMS= 011 (TIMx_SMCR register, both inputs are active on both rising and falling edges)
- CEN= 1 (TIMx_CR1 register, Counter is enabled)

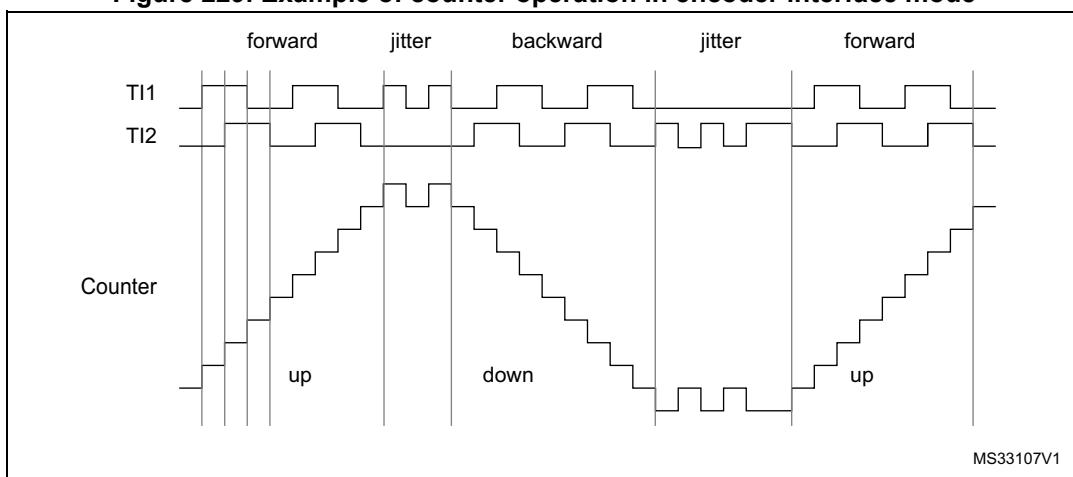
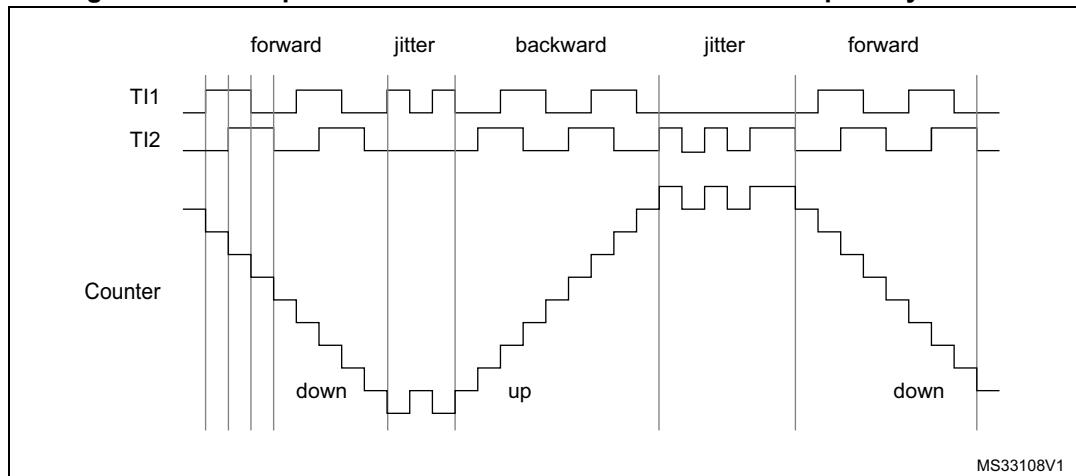
Figure 229. Example of counter operation in encoder interface mode

Figure 230 gives an example of counter behavior when TI1FP1 polarity is inverted (same configuration as above except CC1P=1).

Figure 230. 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.

19.3.16 UIF bit remapping

The IUFREMAP bit in the TIMx_CR1 register forces a continuous copy of the update interrupt flag (UIF) into bit 31 of the timer counter register's bit 31 (TIMxCNT[31]). This allows to atomically read both the counter value and a potential roll-over condition signaled by the UIFCPY flag. It eases the calculation of angular speed by avoiding race conditions caused, for instance, by a processing shared between a background task (counter reading) and an interrupt (update interrupt).

There is no latency between the UIF and UIFCPY flag assertions.

In 32-bit timer implementations, when the IUFREMAP bit is set, bit 31 of the counter is overwritten by the UIFCPY flag upon read access (the counter's most significant bit is only accessible in write mode).

19.3.17 Timer input XOR function

The TI1S bit in the TIM1xx_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 18.3.24: Interfacing with Hall sensors on page 560](#).

19.3.18 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.

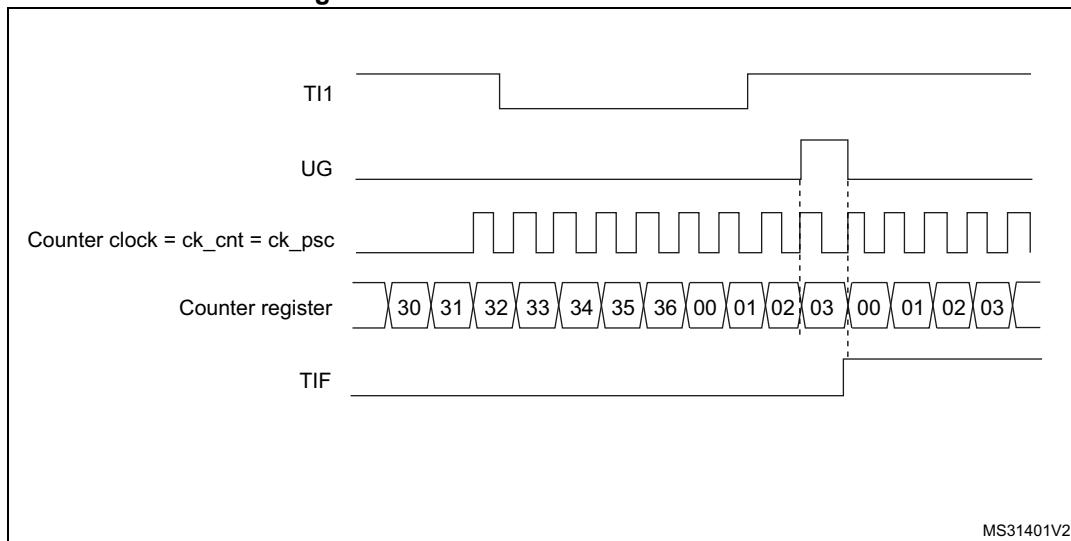
In the following example, the upcounter is cleared in response to a rising edge on TI1 input:

1. 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 and CC1NP=0 in TIMx_CCER register to validate the polarity (and detect rising edges only).
2. 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.
3. 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 231. Control circuit in reset mode



Slave mode: Gated mode

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

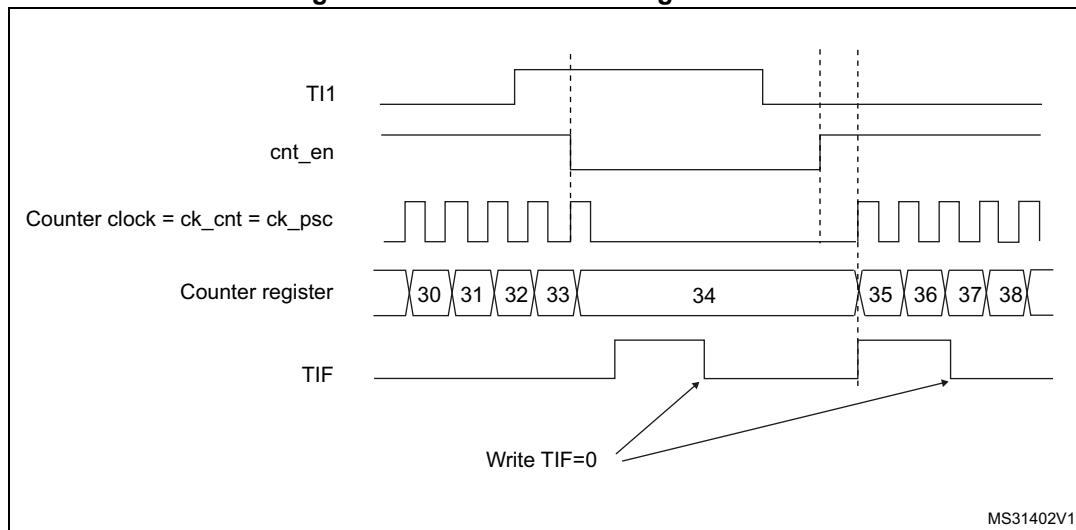
In the following example, the upcounter counts only when TI1 input is low:

1. Configure the channel 1 to detect low levels on TI1. Configure the input filter duration (in this example, we don't need any filter, so we keep IC1F=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 and CC1NP=0 in TIMx_CCER register to validate the polarity (and detect low level only).
2. 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.
3. 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 232. Control circuit in gated mode



1. The configuration “CCxP=CCxNP=1” (detection of both rising and falling edges) does not have any effect in gated mode because gated mode acts on a level and not on an edge.

Note:

The configuration “CCxP=CCxNP=1” (detection of both rising and falling edges) does not have any effect in gated mode because gated mode acts on a level and not on an edge.

Slave mode: Trigger mode

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

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

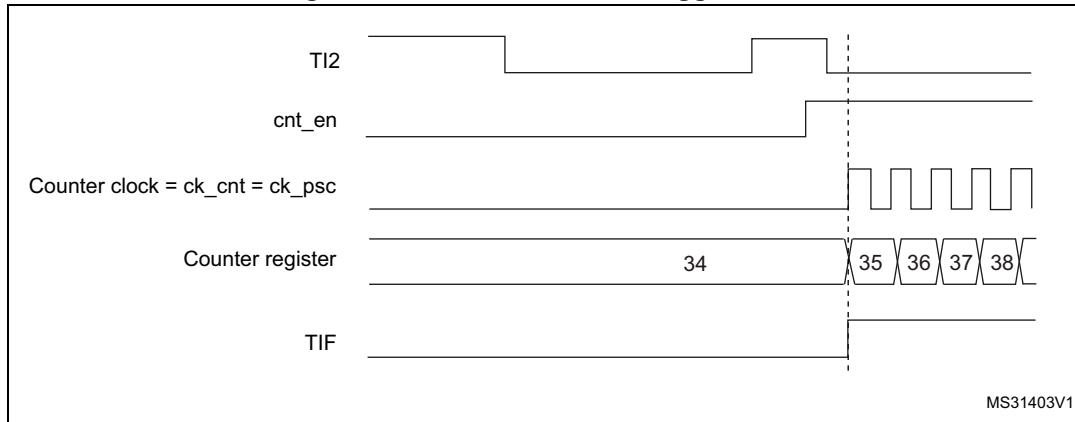
1. Configure the channel 2 to detect rising edges on TI2. Configure the input filter duration (in this example, we don't need any filter, so we keep IC2F=0000). The capture prescaler is not used for triggering, so you don't need to configure it. CC2S bits are selecting the input capture source only, CC2S=01 in TIMx_CCMR1 register. Write

- CC2P=1 and CC2NP=0 in TIMx_CCER register to validate the polarity (and detect low level only).
2. 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 233. 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.

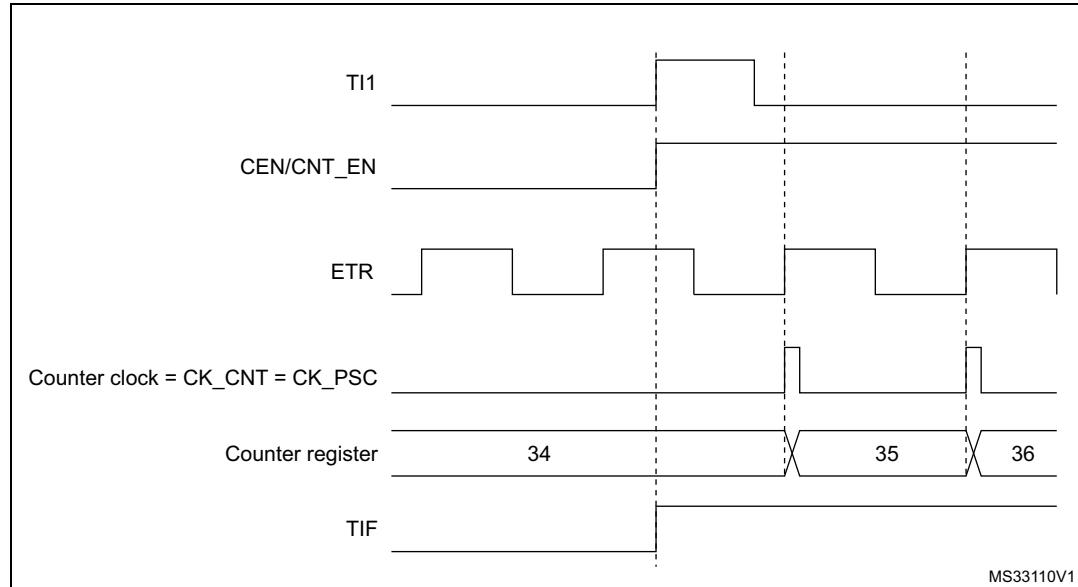
In the following example, the upcounter is incremented at each rising edge of the ETR signal as soon as a rising edge of TI1 occurs:

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 and CC1NP=0 in TIMx_CCER register to validate the polarity (and detect rising edge only).
3. Configure the timer in trigger mode by writing SMS=110 in TIMx_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx_SMCR register.

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 234. Control circuit in external clock mode 2 + trigger mode

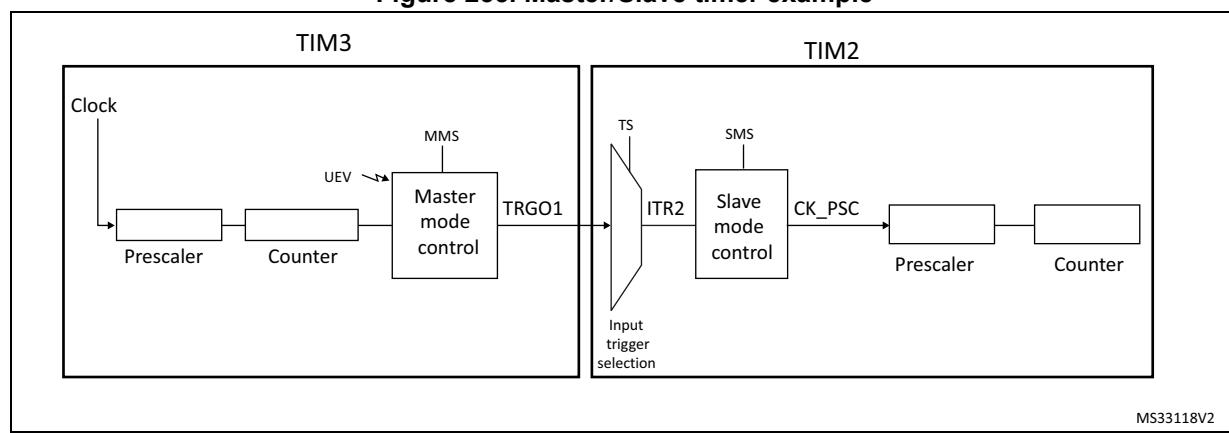


19.3.19 Timer synchronization

The TIMx timers are linked together internally for timer synchronization or chaining. When one Timer is configured in Master Mode, it can reset, start, stop or clock the counter of another Timer configured in Slave Mode.

[Figure 235: Master/Slave timer example](#) presents an overview of the trigger selection and the master mode selection blocks.

Figure 235. Master/Slave timer example



Using one timer as prescaler for another timer

For example, you can configure TIM3 to act as a prescaler for TIM. Refer to [Figure 235](#). To do this:

1. Configure TIM3 in master mode so that it outputs a periodic trigger signal on each update event UEV. If you write MMS=010 in the TIM3_CR2 register, a rising edge is output on TRGO each time an update event is generated.
2. To connect the TRGO output of TIM3 to TIM, TIM must be configured in slave mode using ITR2 as internal trigger. You select this through the TS bits in the TIM_SMCR register (writing TS=010).
3. Then you put the slave mode controller in external clock mode 1 (write SMS=111 in the TIM_SMCR register). This causes TIM to be clocked by the rising edge of the periodic TIM3 trigger signal (which correspond to the TIM3 counter overflow).
4. Finally both timers must be enabled by setting their respective CEN bits (TIMx_CR1 register).

Note: If OCx is selected on TIM3 as the trigger output (MMS=1xx), its rising edge is used to clock the counter of TIM.

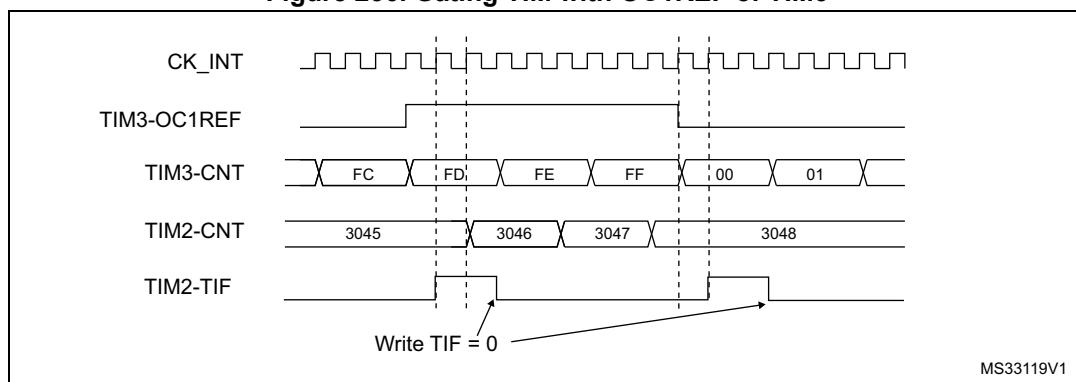
Using one timer to enable another timer

In this example, we control the enable of TIM with the output compare 1 of Timer 3. Refer to [Figure 235](#) for connections. TIM counts on the divided internal clock only when OC1REF of TIM3 is high. Both counter clock frequencies are divided by 3 by the prescaler compared to CK_INT ($f_{CK_CNT} = f_{CK_INT}/3$).

1. Configure TIM3 master mode to send its Output Compare 1 Reference (OC1REF) signal as trigger output (MMS=100 in the TIM3_CR2 register).
2. Configure the TIM3 OC1REF waveform (TIM3_CCMR1 register).
3. Configure TIM to get the input trigger from TIM3 (TS=010 in the TIM_SMCR register).
4. Configure TIM in gated mode (SMS=101 in TIM_SMCR register).
5. Enable TIM by writing '1 in the CEN bit (TIM_CR1 register).
6. Start TIM3 by writing '1 in the CEN bit (TIM3_CR1 register).

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

Figure 236. Gating TIM with OC1REF of TIM3



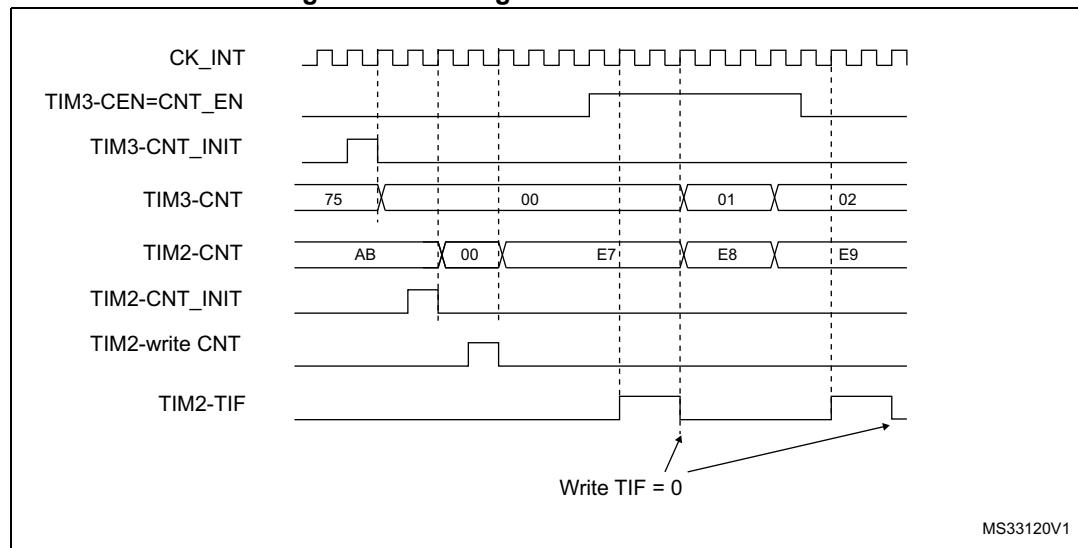
In the example in [Figure 236](#), the TIM counter and prescaler are not initialized before being started. So they start counting from their current value. It is possible to start from a given

value by resetting both timers before starting TIM3. 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 (refer to [Figure 237](#)), we synchronize TIM3 and TIM. TIM3 is the master and starts from 0. TIM is the slave and starts from 0xE7. The prescaler ratio is the same for both timers. TIM stops when TIM3 is disabled by writing '0 to the CEN bit in the TIM3_CR1 register:

1. Configure TIM3 master mode to send its Output Compare 1 Reference (OC1REF) signal as trigger output (MMS=100 in the TIM3_CR2 register).
2. Configure the TIM3 OC1REF waveform (TIM3_CCMR1 register).
3. Configure TIM to get the input trigger from TIM3 (TS=010 in the TIM_SMCR register).
4. Configure TIM in gated mode (SMS=101 in TIM_SMCR register).
5. Reset TIM3 by writing '1 in UG bit (TIM3_EGR register).
6. Reset TIM by writing '1 in UG bit (TIM_EGR register).
7. Initialize TIM to 0xE7 by writing '0xE7' in the TIM counter (TIM_CNTL).
8. Enable TIM by writing '1 in the CEN bit (TIM_CR1 register).
9. Start TIM3 by writing '1 in the CEN bit (TIM3_CR1 register).
10. Stop TIM3 by writing '0 in the CEN bit (TIM3_CR1 register).

Figure 237. Gating TIM with Enable of TIM3

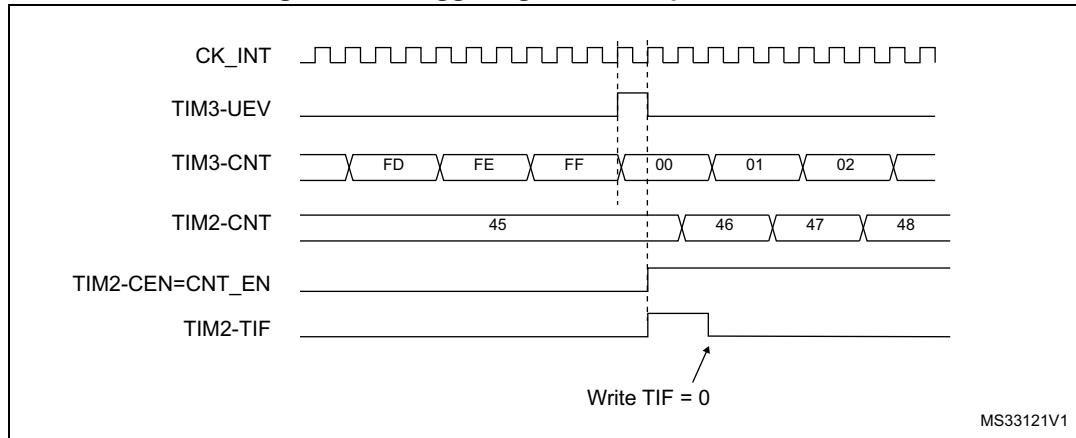


Using one timer to start another timer

In this example, we set the enable of Timer with the update event of Timer 3. Refer to [Figure 235](#) for connections. Timer 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 receives the trigger signal its CEN bit is automatically set and the counter counts until we write '0 to the CEN bit in the TIM_CR1 register. Both counter clock frequencies are divided by 3 by the prescaler compared to CK_INT ($f_{CK_CNT} = f_{CK_INT}/3$).

1. Configure TIM3 master mode to send its Update Event (UEV) as trigger output (MMS=010 in the TIM3_CR2 register).
2. Configure the TIM3 period (TIM3_ARR registers).
3. Configure TIM to get the input trigger from TIM3 (TS=010 in the TIM_SMCR register).
4. Configure TIM in trigger mode (SMS=110 in TIM_SMCR register).
5. Start TIM3 by writing '1 in the CEN bit (TIM3_CR1 register).

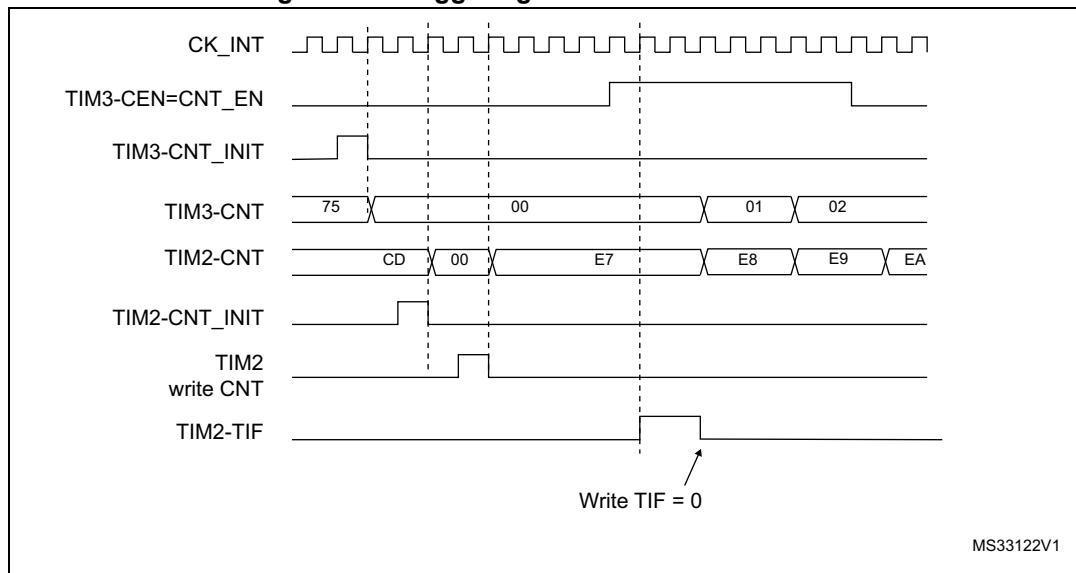
Figure 238. Triggering TIM with update of TIM3



As in the previous example, you can initialize both counters before starting counting.

Figure 239 shows the behavior with the same configuration as in *Figure 238* but in trigger mode instead of gated mode (SMS=110 in the TIM_SMCR register).

Figure 239. Triggering TIM with Enable of TIM3



Starting 2 timers synchronously in response to an external trigger

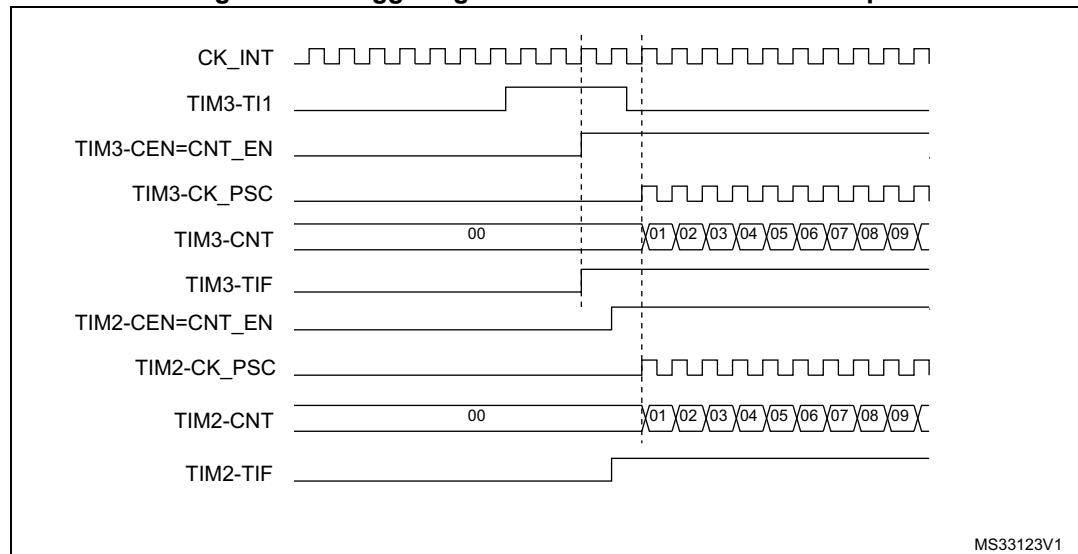
In this example, we set the enable of TIM3 when its TI1 input rises, and the enable of TIM2 with the enable of TIM3. Refer to *Figure 235* for connections. To ensure the counters are aligned, TIM3 must be configured in Master/Slave mode (slave with respect to TI1, master with respect to TIM2):

1. Configure TIM3 master mode to send its Enable as trigger output (MMS=001 in the TIM3_CR2 register).
2. Configure TIM3 slave mode to get the input trigger from TI1 (TS=100 in the TIM3_SMCR register).
3. Configure TIM3 in trigger mode (SMS=110 in the TIM3_SMCR register).
4. Configure the TIM3 in Master/Slave mode by writing MSM=1 (TIM3_SMCR register).
5. Configure TIM2 to get the input trigger from TIM3 (TS=000 in the TIM2_SMCR register).
6. Configure TIM2 in trigger mode (SMS=110 in the TIM2_SMCR register).

When a rising edge occurs on TI1 (TIM3), 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 TIM3.*

Figure 240. Triggering TIM3 and TIM2 with TIM3 TI1 input



Note: *The clock of the slave peripherals (timer, ADC, ...) receiving the TRGO or the TRGO2 signals must be enabled prior to receive events from the master timer, and the clock frequency (prescaler) must not be changed on-the-fly while triggers are received from the master timer.*

19.3.20 DMA burst mode

The TIMx timers have the capability to generate multiple DMA requests upon a single event. The main purpose is to be able to re-program part of the timer multiple times without software overhead, but it can also be used to read several registers in a row, at regular intervals.

The DMA controller destination is unique and must point to the virtual register TIMx_DMAR. On a given timer event, the timer launches a sequence of DMA requests (burst). Each write into the TIMx_DMAR register is actually redirected to one of the timer registers.

The DBL[4:0] bits in the TIMx_DCR register set the DMA burst length. The timer recognizes a burst transfer when a read or a write access is done to the TIMx_DMAR address), i.e. the number of transfers (either in half-words or in bytes).

The DBA[4:0] bits in the TIMx_DCR registers define the DMA base address for DMA transfers (when read/write access are done through the TIMx_DMAR address). DBA is defined as an offset starting from the address of the TIMx_CR1 register:

Example:

00000: TIMx_CR1

00001: TIMx_CR2

00010: TIMx_SMCR

As an example, the timer DMA burst feature is used to update the contents of the CCRx registers ($x = 2, 3, 4$) upon an update event, with the DMA transferring half words into the CCRx registers.

This is done in the following steps:

1. Configure the corresponding DMA channel as follows:
 - DMA channel peripheral address is the DMAR register address
 - DMA channel memory address is the address of the buffer in the RAM containing the data to be transferred by DMA into CCRx registers.
 - Number of data to transfer = 3 (See note below).
 - Circular mode disabled.
2. Configure the DCR register by configuring the DBA and DBL bit fields as follows:
DBL = 3 transfers, DBA = 0xE.
3. Enable the TIMx update DMA request (set the UDE bit in the DIER register).
4. Enable TIMx
5. Enable the DMA channel

This example is for the case where every CCRx register has to be updated once. If every CCRx register is to be updated twice for example, the number of data to transfer should be 6. Let's take the example of a buffer in the RAM containing data1, data2, data3, data4, data5 and data6. The data is transferred to the CCRx registers as follows: on the first update DMA request, data1 is transferred to CCR2, data2 is transferred to CCR3, data3 is transferred to CCR4 and on the second update DMA request, data4 is transferred to CCR2, data5 is transferred to CCR3 and data6 is transferred to CCR4.

Note: A null value can be written to the reserved registers.

19.3.21 Debug mode

When the microcontroller enters debug mode (Cortex®-M7 core - halted), the TIMx counter either continues to work normally or stops, depending on DBG_TIMx_STOP configuration bit in DBGMCU module. For more details, refer to [Section 40.16.2: Debug support for timers, watchdog, bxCAN and I2C](#).

19.4 TIM2/TIM3/TIM4/TIM5 registers

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

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

19.4.1 TIMx control register 1 (TIMx_CR1)(x = 2 to 5)

Address offset: 0x000

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	UIFRE MAP	Res.	CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN
				rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:12 Reserved, must be kept at reset value.

Bit 11 **UIFREMAP**: UIF status bit remapping

- 0: No remapping. UIF status bit is not copied to TIMx_CNT register bit 31.
- 1: Remapping enabled. UIF status bit is copied to TIMx_CNT register bit 31.

Bit 10 Reserved, must be kept at reset value.

Bits 9:8 **CKD[1:0]**: Clock division

This bit-field indicates the division ratio between the timer clock (CK_INT) frequency and sampling clock used by the digital filters (ETR, TIx),

- 00: $t_{DTS} = t_{CK_INT}$
- 01: $t_{DTS} = 2 \times t_{CK_INT}$
- 10: $t_{DTS} = 4 \times t_{CK_INT}$
- 11: Reserved

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

- 0: TIMx_ARR register is not buffered
- 1: TIMx_ARR register is buffered

Bits 6:5 **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.

19.4.2 TIMx control register 2 (TIMx_CR2)(x = 2 to 5)

Address offset: 0x04

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TI1S	MMS[2:0]	CCDS	Res.	Res.	Res.									

Bits 15:8 Reserved, must be kept at reset value.

Bit 7 **TI1S**: TI1 selection

- 0: The TIMx_CH1 pin is connected to TI1 input
 - 1: The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)
- See also [Section 18.3.24: Interfacing with Hall sensors on page 560](#)

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

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

000: **Reset** - the UG bit from the TIMx_EGR register is used as trigger output (TRGO). If the reset is generated by the trigger input (slave mode controller configured in reset mode) then the signal on TRGO is delayed compared to the actual reset.

001: **Enable** - the Counter enable signal, CNT_EN, is used as trigger output (TRGO). It is useful to start several timers at the same time or to control a window in which a slave timer is enabled. The Counter Enable signal is generated by a logic OR between CEN control bit and the trigger input when configured in gated mode.

When the Counter Enable signal is controlled by the trigger input, there is a delay on TRGO, except if the master/slave mode is selected (see the MSM bit description in TIMx_SMCR register).

010: **Update** - The update event is selected as trigger output (TRGO). For instance a master timer can then be used as a prescaler for a slave timer.

011: **Compare Pulse** - The trigger output 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)

Note: The clock of the slave timer or ADC must be enabled prior to receive events from the master timer, and must not be changed on-the-fly while triggers are received from the master timer.

Bit 3 **CCDS**: Capture/compare DMA selection

- 0: CCx DMA request sent when CCx event occurs
- 1: CCx DMA requests sent when update event occurs

Bits 2:0 Reserved, must be kept at reset value.

19.4.3 TIMx slave mode control register (TIMx_SMCR)(x = 2 to 5)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SMS[3]
															rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ETP	ECE	ETPS[1:0]		ETF[3:0]				MSM	TS[2:0]			OCCS	SMS[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bit 16 **SMS[3]:** Slave mode selection - bit 3

Refer to SMS description - bits 2:0.

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.

1: Setting the ECE bit has the same effect as selecting external clock mode 1 with TRGI connected to ETRF (SMS=111 and TS=111).

2: It is possible to simultaneously use external clock mode 2 with the following slave modes: reset mode, gated mode and trigger mode. Nevertheless, TRGI must not be connected to ETRF in this case (TS bits must not be 111).

3: If external clock mode 1 and external clock mode 2 are enabled at the same time, the external clock input is ETRF.

Bits 13:12 **ETPS[1:0]:** External trigger prescaler

External trigger signal ETRP frequency must be at most 1/4 of CK_INT frequency. A prescaler can be enabled to reduce ETRP frequency. It is useful when inputting fast external clocks.

00: Prescaler OFF

01: ETRP frequency divided by 2

10: ETRP frequency divided by 4

11: ETRP frequency divided by 8

Bits 11:8 ETF[3:0]: External trigger filter

This bit-field then defines the frequency used to sample ETRP signal and the length of the digital filter applied to ETRP. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output:

- 0000: No filter, sampling is done at f_{DTS}
- 0001: $f_{SAMPLING} = f_{CK_INT}$, N=2
- 0010: $f_{SAMPLING} = f_{CK_INT}$, N=4
- 0011: $f_{SAMPLING} = f_{CK_INT}$, N=8
- 0100: $f_{SAMPLING} = f_{DTS}/2$, N=6
- 0101: $f_{SAMPLING} = f_{DTS}/2$, N=8
- 0110: $f_{SAMPLING} = f_{DTS}/4$, N=6
- 0111: $f_{SAMPLING} = f_{DTS}/4$, N=8
- 1000: $f_{SAMPLING} = f_{DTS}/8$, N=6
- 1001: $f_{SAMPLING} = f_{DTS}/8$, N=8
- 1010: $f_{SAMPLING} = f_{DTS}/16$, N=5
- 1011: $f_{SAMPLING} = f_{DTS}/16$, N=6
- 1100: $f_{SAMPLING} = f_{DTS}/16$, N=8
- 1101: $f_{SAMPLING} = f_{DTS}/32$, N=5
- 1110: $f_{SAMPLING} = f_{DTS}/32$, N=6
- 1111: $f_{SAMPLING} = f_{DTS}/32$, N=8

Bit 7 MSM: Master/Slave mode

0: No action

1: The effect of an event on the trigger input (TRGI) is delayed to allow a perfect synchronization between the current timer and its slaves (through TRGO). It is useful if we want to synchronize several timers on a single external event.

Bits 6:4 **TS:** Trigger selection

This bit-field selects the trigger input to be used to synchronize the counter.

000: Internal Trigger 0 (ITR0).

001: Internal Trigger 1 (ITR1).

010: Internal Trigger 2 (ITR2).

011: Internal Trigger 3 (ITR3).

100: TI1 Edge Detector (TI1F_ED)

101: Filtered Timer Input 1 (TI1FP1)

110: Filtered Timer Input 2 (TI2FP2)

111: External Trigger input (ETRF)

See [Table 119: TIMx internal trigger connection on page 656](#) for more details on ITRx meaning for each Timer.

Note: These bits must be changed only when they are not used (e.g. when SMS=000) to avoid wrong edge detections at the transition.

Bit 3 **OCCS:** OCREF clear selection

This bit is used to select the OCREF clear source

0: OCREF_CLR_INT is connected to the OCREF_CLR input

1: OCREF_CLR_INT is connected to ETRF

Bits 16, 2, 1, 0 **SMS[3:0]:** Slave mode selection

When external signals are selected the active edge of the trigger signal (TRGI) is linked to the polarity selected on the external input (see Input Control register and Control Register description).

0000: Slave mode disabled - if CEN = '1 then the prescaler is clocked directly by the internal clock.

0001: Encoder mode 1 - Counter counts up/down on TI1FP1 edge depending on TI2FP2 level.

0010: Encoder mode 2 - Counter counts up/down on TI2FP2 edge depending on TI1FP1 level.

0011: Encoder mode 3 - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input.

0100: Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter and generates an update of the registers.

0101: 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.

0110: 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.

0111: External Clock Mode 1 - Rising edges of the selected trigger (TRGI) clock the counter.

1000: Combined reset + trigger mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter, generates an update of the registers and starts the counter.

Note: The gated mode must not be used if TI1F_ED is selected as the trigger input (TS=100). Indeed, TI1F_ED outputs 1 pulse for each transition on TI1F, whereas the gated mode checks the level of the trigger signal.

Note: The clock of the slave peripherals (timer, ADC, ...) receiving the TRGO or the TRGO2 signals must be enabled prior to receive events from the master timer, and the clock frequency (prescaler) must not be changed on-the-fly while triggers are received from the master timer.

Table 119. TIMx internal trigger connection

Slave TIM	ITR0 (TS = 000)	ITR1 (TS = 001)	ITR2 (TS = 010)	ITR3 (TS = 011)
TIM2	TIM1	TIM8/OTG_FS_SOF/OT_G_HS_SOF ⁽¹⁾	TIM3	TIM4
TIM3	TIM1	TIM2	TIM5	TIM4
TIM4	TIM1	TIM2	TIM3	TIM8
TIM5	TIM2	TIM3	TIM4	TIM8

1. Depends on the bit ITR1_RMP in TIM2_OR1 register.

19.4.4 TIMx DMA/Interrupt enable register (TIMx_DIER)(x = 2 to 5)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res.	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Res.	TIE	Res.	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw		rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw

Bit 15 Reserved, must be kept at reset value.

Bit 14 **TDE**: Trigger DMA request enable

- 0: Trigger DMA request disabled.
- 1: Trigger DMA request enabled.

Bit 13 Reserved, must be kept at reset value.

Bit 12 **CC4DE**: Capture/Compare 4 DMA request enable

- 0: CC4 DMA request disabled.
- 1: CC4 DMA request enabled.

Bit 11 **CC3DE**: Capture/Compare 3 DMA request enable

- 0: CC3 DMA request disabled.
- 1: CC3 DMA request enabled.

Bit 10 **CC2DE**: Capture/Compare 2 DMA request enable

- 0: CC2 DMA request disabled.
- 1: CC2 DMA request enabled.

Bit 9 **CC1DE**: Capture/Compare 1 DMA request enable

- 0: CC1 DMA request disabled.
- 1: CC1 DMA request enabled.

Bit 8 **UDE**: Update DMA request enable

- 0: Update DMA request disabled.
- 1: Update DMA request enabled.

Bit 7 Reserved, must be kept at reset value.

Bit 6 **TIE**: Trigger interrupt enable

- 0: Trigger interrupt disabled.
- 1: Trigger interrupt enabled.

Bit 5 Reserved, must be kept at reset value.

Bit 4 **CC4IE**: Capture/Compare 4 interrupt enable

- 0: CC4 interrupt disabled.
- 1: CC4 interrupt enabled.

Bit 3 **CC3IE**: Capture/Compare 3 interrupt enable

- 0: CC3 interrupt disabled.
- 1: CC3 interrupt enabled.

Bit 2 **CC2IE**: Capture/Compare 2 interrupt enable

- 0: CC2 interrupt disabled.
- 1: CC2 interrupt enabled.

Bit 1 **CC1IE**: Capture/Compare 1 interrupt enable

- 0: CC1 interrupt disabled.
- 1: CC1 interrupt enabled.

Bit 0 **UIE**: Update interrupt enable

- 0: Update interrupt disabled.
- 1: Update interrupt enabled.

19.4.5 TIMx status register (TIMx_SR)(x = 2 to 5)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	CC4OF	CC3OF	CC2OF	CC1OF	Res.	Res.	TIF	Res.	CC4IF	CC3IF	CC2IF	CC1IF	UIF

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

Bit 12 **CC4OF**: Capture/Compare 4 overcapture flag
refer to CC1OF description

Bit 11 **CC3OF**: Capture/Compare 3 overcapture flag
refer to CC1OF description

Bit 10 **CC2OF**: Capture/compare 2 overcapture flag
refer to CC1OF description

Bit 9 **CC1OF**: Capture/Compare 1 overcapture flag

This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to '0'.

0: No overcapture has been detected.

1: The counter value has been captured in TIMx_CCR1 register while CC1IF flag was already set

Bits 8:7 Reserved, must be kept at reset value.

Bit 6 **TIF**: Trigger interrupt flag

This flag is set by hardware on trigger event (active edge detected on TRGI input when the slave mode controller is enabled in all modes but gated mode. It is set when the counter starts or stops when gated mode is selected. It is cleared by software.

0: No trigger event occurred.

1: Trigger interrupt pending.

Bit 5 Reserved, must be kept at reset value.

Bit 4 **CC4IF**: Capture/Compare 4 interrupt flag

Refer to CC1IF description

Bit 3 **CC3IF**: Capture/Compare 3 interrupt flag

Refer to CC1IF description

Bit 2 **CC2IF**: Capture/Compare 2 interrupt flag

Refer to CC1IF description

Bit 1 **CC1IF**: Capture/compare 1 interrupt flag

If channel CC1 is configured as output: This flag is set by hardware when the counter matches the compare value, with some exception in center-aligned mode (refer to the CMS bits in the TIMx_CR1 register description) and in retriggerable one pulse mode. 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 (for TIM2 to TIM4) and if UDIS=0 in the TIMx_CR1 register.

When CNT is reinitialized by software using the UG bit in TIMx_EGR register, if URS=0 and UDIS=0 in the TIMx_CR1 register.

When CNT is reinitialized by a trigger event (refer to the synchro control register description), if URS=0 and UDIS=0 in the TIMx_CR1 register.

19.4.6 TIMx event generation register (TIMx_EGR)(x = 2 to 5)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TG	Res.	CC4G	CC3G	CC2G	CC1G	UG								

Bits 15:7 Reserved, must be kept at reset value.

Bit 6 **TG**: Trigger generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: The TIF flag is set in TIMx_SR register. Related interrupt or DMA transfer can occur if enabled.

Bit 5 Reserved, must be kept at reset value.

Bit 4 **CC4G**: Capture/compare 4 generation

Refer to CC1G description

- Bit 3 **CC3G**: Capture/compare 3 generation
Refer to CC1G description
- Bit 2 **CC2G**: Capture/compare 2 generation
Refer to CC1G description
- Bit 1 **CC1G**: Capture/compare 1 generation
This bit is set by software in order to generate an event, it is automatically cleared by hardware.
0: No action
1: A capture/compare event is generated on channel 1:
If channel CC1 is configured as output:
CC1IF flag is set, Corresponding interrupt or DMA request is sent if enabled.
If channel CC1 is configured as input:
The current value of the counter is captured in TIMx_CCR1 register. The CC1IF flag is set, the corresponding interrupt or DMA request is sent if enabled. The CC1OF flag is set if the CC1IF flag was already high.
- Bit 0 **UG**: Update generation
This bit can be set by software, it is automatically cleared by hardware.
0: No action
1: Re-initialize the counter and generates an update of the registers. Note that the prescaler counter is cleared too (anyway the prescaler ratio is not affected). The counter is cleared if the center-aligned mode is selected or if DIR=0 (upcounting), else it takes the auto-reload value (TIMx_ARR) if DIR=1 (downcounting).

19.4.7 TIMx capture/compare mode register 1 (TIMx_CCMR1)(x = 2 to 5)

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.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC2M [3]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC1M [3]
							Res.								Res.
							rw								rw
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	rw

Output compare mode

Bits 31:25 Reserved, always read as 0.

Bit 24 **OC2M[3]**: Output Compare 2 mode - bit 3

Bits 23:17 Reserved, always read as 0.

Bit 16 **OC1M[3]**: Output Compare 1 mode - bit 3

Bit 15 **OC2CE**: Output compare 2 clear enable

Bits 14:12 **OC2M[2:0]**: Output compare 2 mode

refer to OC1M description on bits 6:4

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

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.

0000: Frozen - The comparison between the output compare register TIMx_CCR1 and the counter TIMx_CNT has no effect on the outputs.(this mode is used to generate a timing base).

0001: 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).

0010: 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).

0011: Toggle - OC1REF toggles when TIMx_CNT=TIMx_CCR1.

0100: Force inactive level - OC1REF is forced low.

0101: Force active level - OC1REF is forced high.

0110: 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).

0111: 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.

1000: Retriggerable OPM mode 1 - In up-counting mode, the channel is active until a trigger event is detected (on TRGI signal). Then, a comparison is performed as in PWM mode 1 and the channels becomes inactive again at the next update. In down-counting mode, the channel is inactive until a trigger event is detected (on TRGI signal). Then, a comparison is performed as in PWM mode 1 and the channels becomes inactive again at the next update.

1001: Retriggerable OPM mode 2 - In up-counting mode, the channel is inactive until a trigger event is detected (on TRGI signal). Then, a comparison is performed as in PWM mode 2 and the channels becomes inactive again at the next update. In down-counting mode, the channel is active until a trigger event is detected (on TRGI signal). Then, a comparison is performed as in PWM mode 1 and the channels becomes active again at the next update.

1010: Reserved,

1011: Reserved,

1100: Combined PWM mode 1 - OC1REF has the same behavior as in PWM mode 1. OC1REFC is the logical OR between OC1REF and OC2REF.

1101: Combined PWM mode 2 - OC1REF has the same behavior as in PWM mode 2. OC1REFC is the logical AND between OC1REF and OC2REF.

1110: Asymmetric PWM mode 1 - OC1REF has the same behavior as in PWM mode 1. OC1REFC outputs OC1REF when the counter is counting up, OC2REF when it is counting down.

1111: Asymmetric PWM mode 2 - OC1REF has the same behavior as in PWM mode 2. OC1REFC outputs OC1REF when the counter is counting up, OC2REF when it is counting down.

Note: **1:** These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx_BDTR register) and CC1S=00 (the channel is configured in output).

2: In PWM mode, 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).

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. OCFE acts only if the channel is configured in PWM1 or PWM2 mode.

Bits 1:0 **CC1S**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output.

01: CC1 channel is configured as input, IC1 is mapped on TI1.

10: CC1 channel is configured as input, IC1 is mapped on TI2.

11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC1S bits are writable only when the channel is OFF (CC1E = 0 in TIMx_CCER).

Input capture mode

Bits 31:16 Reserved, always read as 0.

Bits 15:12 **IC2F**: Input capture 2 filter

Bits 11:10 **IC2PSC[1:0]**: Input capture 2 prescaler

Bits 9:8 **CC2S**: Capture/compare 2 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output.

01: CC2 channel is configured as input, IC2 is mapped on TI2.

10: CC2 channel is configured as input, IC2 is mapped on TI1.

11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC2S bits are writable only when the channel is OFF (CC2E = 0 in TIMx_CCER).

Bits 7:4 **IC1F**: Input capture 1 filter

This bit-field defines the frequency used to sample TI1 input and the length of the digital filter applied to TI1. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output:

- 0000: No filter, sampling is done at f_{DTS}
- 0001: $f_{SAMPLING} = f_{CK_INT}$, N=2
- 0010: $f_{SAMPLING} = f_{CK_INT}$, N=4
- 0011: $f_{SAMPLING} = f_{CK_INT}$, N=8
- 0100: $f_{SAMPLING} = f_{DTS}/2$, N=6
- 0101: $f_{SAMPLING} = f_{DTS}/2$, N=8
- 0110: $f_{SAMPLING} = f_{DTS}/4$, N=6
- 0111: $f_{SAMPLING} = f_{DTS}/4$, N=8
- 1000: $f_{SAMPLING} = f_{DTS}/8$, N=6
- 1001: $f_{SAMPLING} = f_{DTS}/8$, N=8
- 1010: $f_{SAMPLING} = f_{DTS}/16$, N=5
- 1011: $f_{SAMPLING} = f_{DTS}/16$, N=6
- 1100: $f_{SAMPLING} = f_{DTS}/16$, N=8
- 1101: $f_{SAMPLING} = f_{DTS}/32$, N=5
- 1110: $f_{SAMPLING} = f_{DTS}/32$, N=6
- 1111: $f_{SAMPLING} = f_{DTS}/32$, N=8

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).

19.4.8 TIMx capture/compare mode register 2 (TIMx_CCMR2)(x = 2 to 5)

Address offset: 0x1C

Reset value: 0x0000

Refer to the above CCMR1 register description.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC4M [3]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC3M [3]
							Res.								Res.
							rw								rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC4CE	OC4M[2:0]			OC4PE	OC4FE	CC4S[1:0]	OC3CE	OC3M[2:0]			OC3PE	OC3FE	CC3S[1:0]		
IC4F[3:0]				IC4PSC[1:0]			IC3F[3:0]			IC3PSC[1:0]					
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Output compare mode

Bits 31:25 Reserved, always read as 0.

Bit 24 **OC4M[3]**: Output Compare 2 mode - bit 3

Bits 23:17 Reserved, always read as 0.

Bit 16 **OC3M[3]**: Output Compare 1 mode - bit 3

Bit 15 **OC4CE**: Output compare 4 clear enable

Bits 14:12 **OC4M**: Output compare 4 mode

Refer to OC1M description (bits 6:4 in TIMx_CCMR1 register)

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

Refer to OC1M description (bits 6:4 in TIMx_CCMR1 register)

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 31:16 Reserved, always read as 0.

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).

19.4.9 TIMx capture/compare enable register (TIMx_CCER)(x = 2 to 5)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC4NP	Res.	CC4P	CC4E	CC3NP	Res.	CC3P	CC3E	CC2NP	Res.	CC2P	CC2E	CC1NP	Res.	CC1P	CC1E
rw		rw	rw												

Bit 15 **CC4NP**: Capture/Compare 4 output Polarity.

Refer to CC1NP description

Bit 14 Reserved, must be kept at reset value.

Bit 13 **CC4P**: Capture/Compare 4 output Polarity.

Refer to CC1P description

Bit 12 **CC4E**: Capture/Compare 4 output enable.

refer to CC1E description

Bit 11 **CC3NP**: Capture/Compare 3 output Polarity.

Refer to CC1NP description

Bit 10 Reserved, must be kept at reset value.

Bit 9 **CC3P**: Capture/Compare 3 output Polarity.

Refer to CC1P description

Bit 8 **CC3E**: Capture/Compare 3 output enable.

Refer to CC1E description

Bit 7 **CC2NP**: Capture/Compare 2 output Polarity.

Refer to CC1NP description

Bit 6 Reserved, must be kept at reset value.

Bit 5 **CC2P**: Capture/Compare 2 output Polarity.

refer to CC1P description

Bit 4 **CC2E**: Capture/Compare 2 output enable.

Refer to CC1E description

Bit 3 **CC1NP**: Capture/Compare 1 output Polarity.

CC1 channel configured as output: CC1NP must be kept cleared in this case.

CC1 channel configured as input: This bit is used in conjunction with CC1P to define TI1FP1/TI2FP1 polarity. refer to CC1P description.

Bit 2 Reserved, must be kept at reset value.

Bit 1 **CC1P: Capture/Compare 1 output Polarity.**

CC1 channel configured as output:

0: OC1 active high

1: OC1 active low

CC1 channel configured as input: CC1NP/CC1P bits select TI1FP1 and TI2FP1 polarity for trigger or capture operations.

00: noninverted/rising edge

Circuit is sensitive to TIxFP1 rising edge (capture, trigger in reset, external clock or trigger mode), TIxFP1 is not inverted (trigger in gated mode, encoder mode).

01: inverted/falling edge

Circuit is sensitive to TIxFP1 falling edge (capture, trigger in reset, external clock or trigger mode), TIxFP1 is inverted (trigger in gated mode, encoder mode).

10: reserved, do not use this configuration.

11: noninverted/both edges

Circuit is sensitive to both TIxFP1 rising and falling edges (capture, trigger in reset, external clock or trigger mode), TIxFP1 is not inverted (trigger in gated mode). This configuration must not be used for encoder mode.

Bit 0 **CC1E: Capture/Compare 1 output enable.**

CC1 channel configured as output:

0: Off - OC1 is not active

1: On - OC1 signal is output on the corresponding output pin

CC1 channel configured as input: This bit determines if a capture of the counter value can actually be done into the input capture/compare register 1 (TIMx_CCR1) or not.

0: Capture disabled

1: Capture enabled

Table 120. Output control bit for standard OCx channels

CCxE bit	OCx output state
0	Output Disabled (OCx=0, OCx_EN=0)
1	OCx=OCxREF + Polarity, OCx_EN=1

Note: The state of the external IO pins connected to the standard OCx channels depends on the OCx channel state and the GPIO and AFIO registers.

19.4.10 TIMx counter (TIMx_CNT)(x = 2 to 5)

Address offset: 0x24

Reset value: 0x0000 0000

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

Bit 31 Value depends on UIFREMAP in TIMx_CR1.

If UIFREMAP = 0

CNT[31]: Most significant bit of counter value (on TIM2 and TIM5)

Reserved on other timers

If UIFREMAP = 1

UIFCPY: UIF Copy

This bit is a read-only copy of the UIF bit of the TIMx_ISR register

Bits 30:16 **CNT[30:16]:** Most significant part counter value (on TIM2 and TIM5)

Bits 15:0 **CNT[15:0]:** Least significant part of counter value

19.4.11 TIMx prescaler (TIMx_PSC)(x = 2 to 5)

Address offset: 0x28

Reset value: 0x0000

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

Bits 15:0 **PSC[15:0]:** Prescaler value

The counter clock frequency CK_CNT is equal to $f_{CK_PSC} / (PSC[15:0] + 1)$.

PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx_EGR register or through trigger controller when configured in “reset mode”).

19.4.12 TIMx auto-reload register (TIMx_ARR)(x = 2 to 5)

Address offset: 0x2C

Reset value: 0xFFFF FFFF

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

Bits 31:16 **ARR[31:16]:** High auto-reload value (on TIM2 and TIM5)

Bits 15:0 **ARR[15:0]:** Low Auto-reload value

ARR is the value to be loaded in the actual auto-reload register.

Refer to the [Section 19.3.1: Time-base unit on page 607](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

19.4.13 TIMx capture/compare register 1 (TIMx_CCR1)(x = 2 to 5)

Address offset: 0x34

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR1[31:16] (depending on timers)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **CCR1[31:16]**: High Capture/Compare 1 value (on TIM2 and TIM5)

Bits 15:0 **CCR1[15:0]**: Low Capture/Compare 1 value

If channel CC1 is configured as output:

CCR1 is the value to be loaded in the actual capture/compare 1 register (preload value). It is loaded permanently if the preload feature is not selected in the TIMx_CCMR1 register (bit OC1PE). 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). The TIMx_CCR1 register is read-only and cannot be programmed.

19.4.14 TIMx capture/compare register 2 (TIMx_CCR2)(x = 2 to 5)

Address offset: 0x38

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR2[31:16] (depending on timers)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR2[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **CCR2[31:16]**: High Capture/Compare 2 value (on TIM2 and TIM5)

Bits 15:0 **CCR2[15:0]**: Low Capture/Compare 2 value

If channel CC2 is configured as output:

CCR2 is the value to be loaded in the actual capture/compare 2 register (preload value). It is loaded permanently if the preload feature is not selected in the TIMx_CCMR1 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). The TIMx_CCR2 register is read-only and cannot be programmed.

19.4.15 TIMx capture/compare register 3 (TIMx_CCR3)(x = 2 to 5)

Address offset: 0x3C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR3[31:16] (depending on timers)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR3[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **CCR3[31:16]**: High Capture/Compare 3 value (on TIM2 and TIM5)

Bits 15:0 **CCR3[15:0]**: Low 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_CCMR2 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). The TIMx_CCR3 register is read-only and cannot be programmed.

19.4.16 TIMx capture/compare register 4 (TIMx_CCR4)(x = 2 to 5)

Address offset: 0x40

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR4[31:16] (depending on timers)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR4[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **CCR4[31:16]**: High Capture/Compare 4 value (on TIM2 and TIM5)

Bits 15:0 **CCR4[15:0]**: Low 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_CCMR2 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). The TIMx_CCR4 register is read-only and cannot be programmed.

19.4.17 TIMx DMA control register (TIMx_DCR)(x = 2 to 5)

Address offset: 0x48

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	DBL[4:0]								Res.	Res.	Res.	DBA[4:0]	
			RW	RW	RW	RW	RW				RW	RW	RW	RW	RW

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

Bits 12:8 **DBL[4:0]**: DMA burst length

This 5-bit vector defines the number of DMA transfers (the timer recognizes a burst transfer when a read or a write access is done to the TIMx_DMAR address).

00000: 1 transfer,
00001: 2 transfers,
00010: 3 transfers,

...
10001: 18 transfers.

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

Bits 4:0 **DBA[4:0]**: DMA base address

This 5-bit vector defines the base-address for DMA transfers (when read/write access are done through the TIMx_DMAR address). DBA is defined as an offset starting from the address of the TIMx_CR1 register.

Example:

00000: TIMx_CR1
00001: TIMx_CR2
00010: TIMx_SMCR

...
Example: Let us consider the following transfer: DBL = 7 transfers & DBA = TIMx_CR1. In this case the transfer is done to/from 7 registers starting from the TIMx_CR1 address.

19.4.18 TIMx DMA address for full transfer (TIMx_DMAR)(x = 2 to 5)

Address offset: 0x4C

Reset value: 0x0000

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

Bits 15:0 **DMAB[15:0]**: DMA register for burst accesses

A read or write operation to the DMAR register accesses the register located at the address
(TIMx_CR1 address) + (DBA + DMA index) × 4

where TIMx_CR1 address is the address of the control register 1, DBA is the DMA base address configured in TIMx_DCR register, DMA index is automatically controlled by the DMA transfer, and ranges from 0 to DBL (DBL configured in TIMx_DCR).

19.4.19 TIM2 option register (TIM2_OR)

Address offset: 0x50

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	ITR1_RMP[1:0]	Res.										

Bits 15:12 Reserved, must be kept at reset value.

Bits 11:10 **ITR1_RMP[1:0]**: Internal trigger 1 remap

Set and cleared by software.

00: TIM8_TRGOUT

01: Reserved

10: OTG_FS_SOF is connected to the TIM2_ITR1 input

11: OTG_HS_SOF is connected to the TIM2_ITR1 input

Bits 9:0 Reserved, must be kept at reset value.

19.4.20 TIM5 option register (TIM5_OR)

Address offset: 0x50

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TI4_RMP[1:0]	Res.													

Bits 15:8 Reserved, must be kept at reset value.

Bits 7:6 **TI4_RMP[1:0]**: Timer Input 4 remap

Set and cleared by software.

00: TIM5 channel4 is connected to the GPIO: Refer to the alternate function mapping table in the datasheets.

01: The LSI internal clock is connected to the TIM5_CH4 input for calibration purposes

10: The LSE internal clock is connected to the TIM5_CH4 input for calibration purposes

11: The RTC wakeup interrupt is connected to the TIM5_CH4 input for calibration purposes.

Wakeup interrupt should be enabled.

Bits 5:0 Reserved, must be kept at reset value.

19.4.21 TIMx register map

TIMx registers are mapped as described in the table below:

Table 121. TIM2/TIM3/TIM4/TIM5 register map and reset values

Table 121. TIM2/TIM3/TIM4/TIM5 register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x24	TIMx_CNT	CNT[31] or UIFCPY	CNT[30:16] (TIM2 and TIM5 only, reserved on the other timers)															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			
0x28	TIMx_PSC	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	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			
0x2C	TIMx_ARR	ARR[31:16] (TIM2 and TIM5 only, reserved on the other timers)	ARR[31:16] (TIM2 and TIM5 only, reserved on the other timers)															ARR[15:0]															
			Reset value	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1			
0x30		Reserved																															
0x34	TIMx_CCR1	CCR1[31:16] (TIM2 and TIM5 only, reserved on the other timers)	CCR1[31:16] (TIM2 and TIM5 only, reserved on the other timers)															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			
0x38	TIMx_CCR2	CCR2[31:16] (TIM2 and TIM5 only, reserved on the other timers)	CCR2[31:16] (TIM2 and TIM5 only, reserved on the other timers)															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			
0x3C	TIMx_CCR3	CCR3[31:16] (TIM2 and TIM5 only, reserved on the other timers)	CCR3[31:16] (TIM2 and TIM5 only, reserved on the other timers)															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			
0x40	TIMx_CCR4	CCR4[31:16] (TIM2 and TIM5 only, reserved on the other timers)	CCR4[31:16] (TIM2 and TIM5 only, reserved on the other timers)															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			
0x44		Reserved																															
0x48	TIMx_DCR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DBL[4:0]				Res.	Res.	Res.	Res.	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			
0x4C	TIMx_DMAR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	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			
0x50	TIM2_OR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
			Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

Table 121. TIM2/TIM3/TIM4/TIM5 register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x50	TIM5_OR	Res.																															
	Reset value																																

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

20 General-purpose timers (TIM9/TIM10/TIM11/TIM12/TIM13/TIM14)

20.1 TIM9/TIM10/TIM11/TIM12/TIM13/TIM14 introduction

The TIM9/TIM10/TIM11/TIM12/TIM13/TIM14 general-purpose timers consist in a 16-bit auto-reload counter driven by a programmable prescaler.

They may be used for a variety of purposes, including measuring the pulse lengths of input signals (input capture) or generating output waveforms (output compare, PWM).

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 TIM9/TIM10/TIM11/TIM12/TIM13/TIM14 timers are completely independent, and do not share any resources. They can be synchronized together as described in [Section 20.3.16: Timer synchronization \(TIM9/TIM12\)](#).

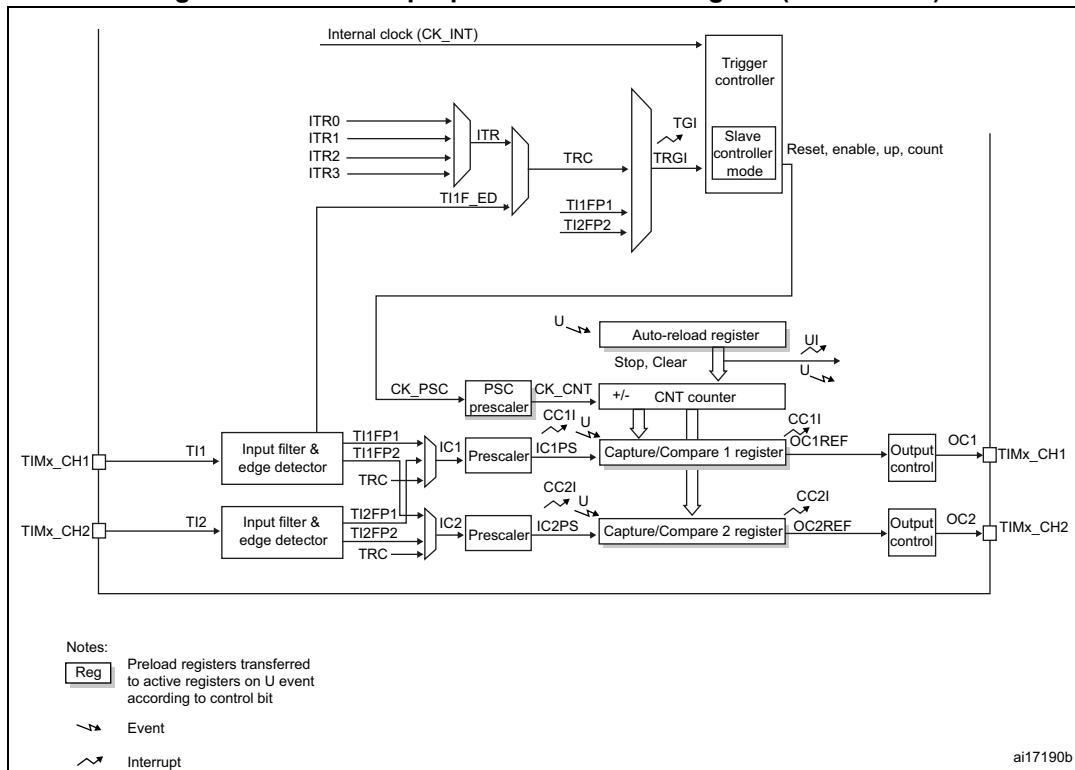
20.2 TIM9/TIM10/TIM11/TIM12/TIM13/TIM14 main features

20.2.1 TIM9/TIM12 main features

The features of the TIM9/TIM12 general-purpose timers include:

- 16-bit auto-reload upcounter
- 16-bit programmable prescaler used to divide the counter clock frequency by any factor between 1 and 65536 (can be changed “on the fly”)
- Up to 2 independent channels for:
 - Input capture
 - Output compare
 - PWM generation (edge-aligned mode)
 - One-pulse mode output
- Synchronization circuit to control the timer with external signals and to interconnect several timers together
- Interrupt generation on the following events:
 - Update: counter overflow, counter initialization (by software or internal trigger)
 - Trigger event (counter start, stop, initialization or count by internal trigger)
 - Input capture
 - Output compare

Figure 241. General-purpose timer block diagram (TIM9/TIM12)

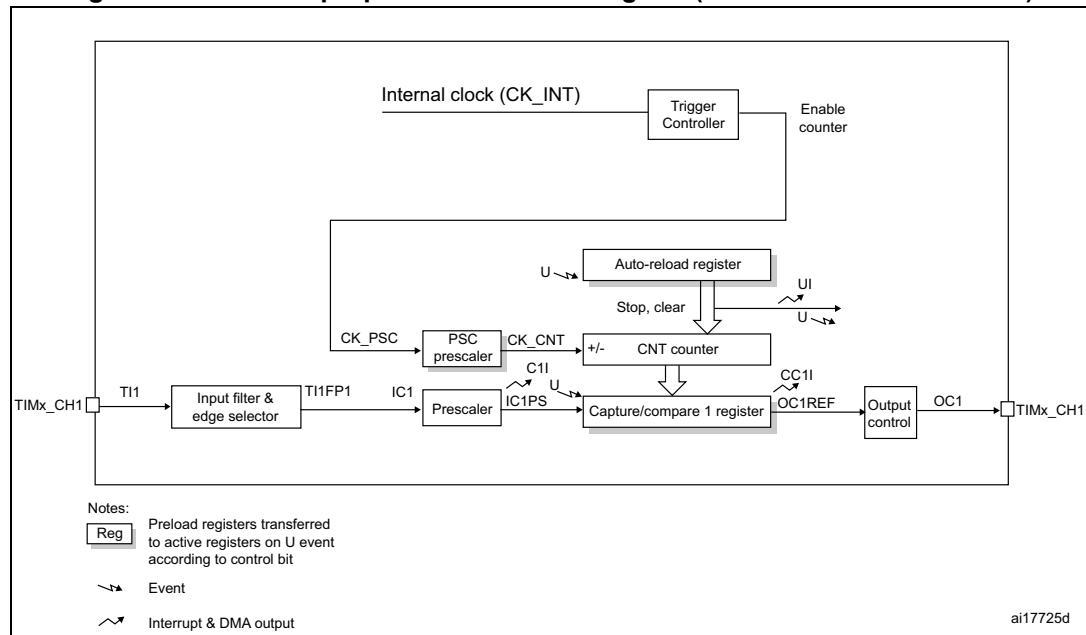


20.2.2 TIM10/TIM11/TIM13/TIM14 main features

The features of general-purpose timers TIM10/TIM11/TIM13/TIM14 include:

- 16-bit auto-reload upcounter
- 16-bit programmable prescaler used to divide the counter clock frequency by any factor between 1 and 65536 (can be changed “on the fly”)
- independent channel for:
 - Input capture
 - Output compare
 - PWM generation (edge-aligned mode)
 - One-pulse mode output
- Interrupt generation on the following events:
 - Update: counter overflow, counter initialization (by software)
 - Input capture
 - Output compare

Figure 242. General-purpose timer block diagram (TIM10/TIM11/TIM13/TIM14)



20.3 TIM9/TIM10/TIM11/TIM12/TIM13/TIM14 functional description

20.3.1 Time-base unit

The main block of the timer is a 16-bit up-counter with its related auto-reload register. The counter clock can be divided by a prescaler.

The counter, the auto-reload register and the prescaler register can be written or read by software. This is true even when the counter is running.

The time-base unit includes:

- Counter register (TIMx_CNT)
- Prescaler register (TIMx_PSC)
- Auto-reload register (TIMx_ARR)

The auto-reload register is preloaded. Writing to or reading from the auto-reload register accesses the preload register. The content of the preload register are transferred into the shadow register permanently or at each update event (UEV), depending on the auto-reload preload enable bit (ARPE) in TIMx_CR1 register. The update event is sent when the counter reaches the overflow and if the UDIS bit equals 0 in the TIMx_CR1 register. It can also be generated by software. The generation of the update event is described in details for each configuration.

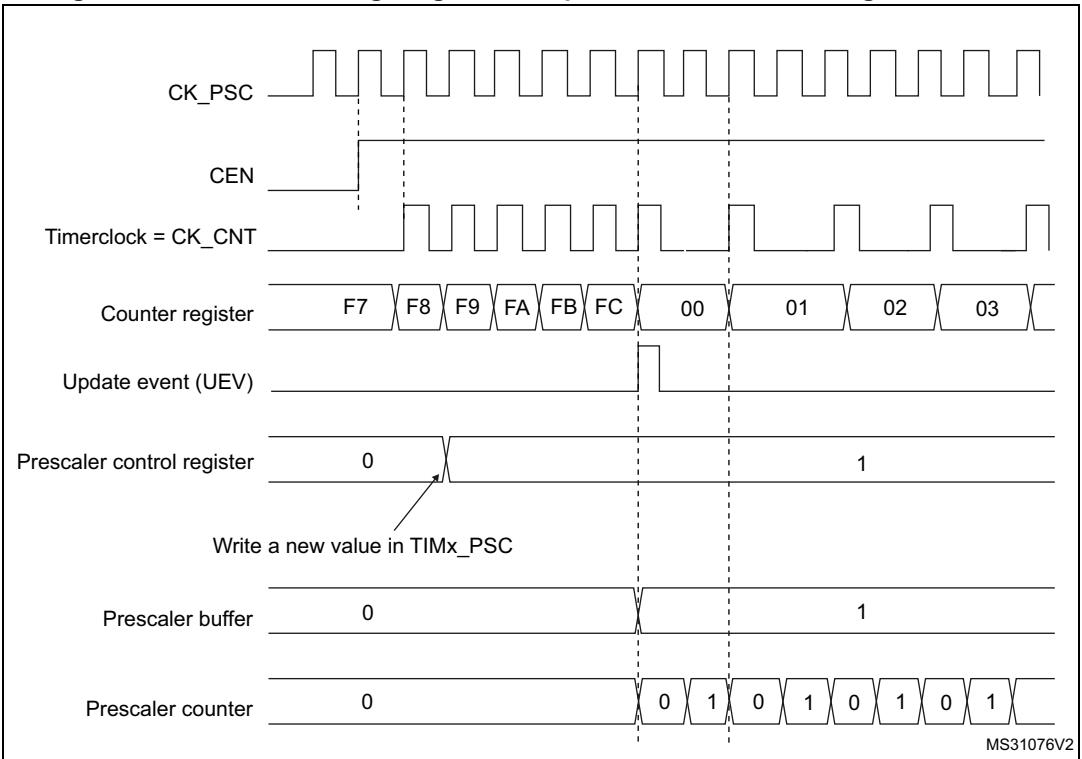
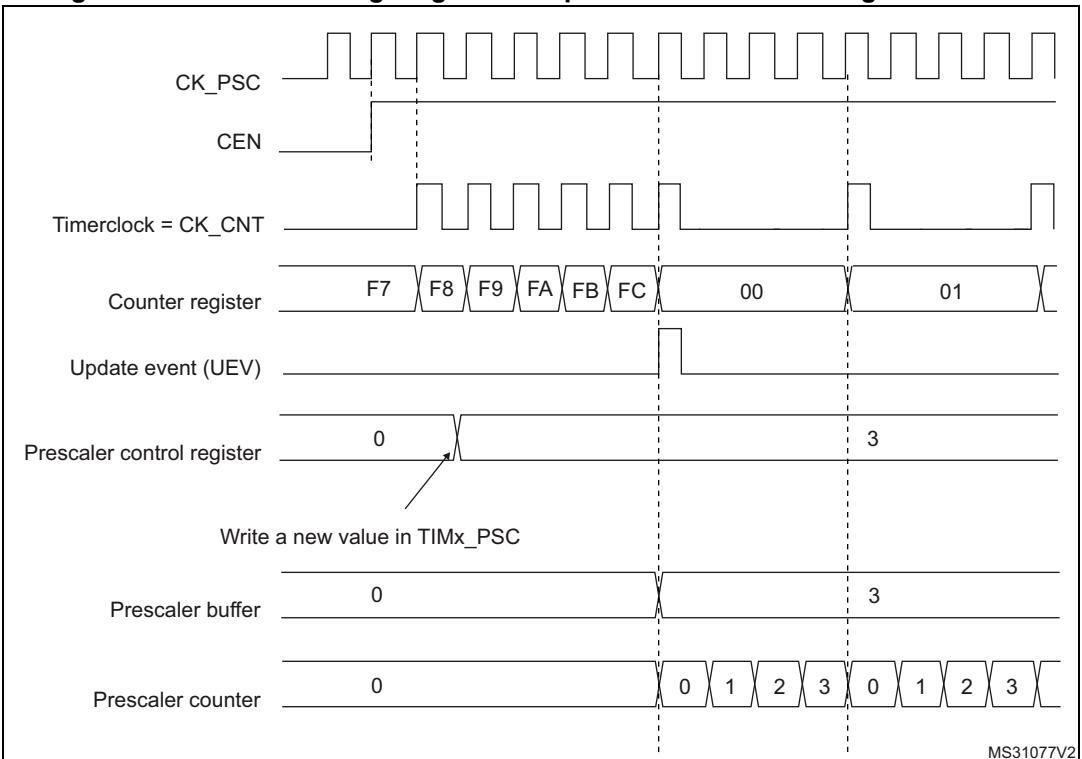
The counter is clocked by the prescaler output CK_CNT, which is enabled only when the counter enable bit (CEN) in TIMx_CR1 register is set (refer also to the slave mode controller description to get more details on counter enabling).

Note that the counter starts counting 1 clock cycle after setting the CEN bit in the TIMx_CR1 register.

Prescaler description

The prescaler can divide the counter clock frequency by any factor between 1 and 65536. It is based on a 16-bit counter controlled through a 16-bit register (in the TIMx_PSC register). It can be changed on the fly as this control register is buffered. The new prescaler ratio is taken into account at the next update event.

Figure 243 and *Figure 244* give some examples of the counter behavior when the prescaler ratio is changed on the fly.

Figure 243. Counter timing diagram with prescaler division change from 1 to 2**Figure 244. Counter timing diagram with prescaler division change from 1 to 4**

20.3.2 Counter modes

Upcounting mode

In upcounting mode, the counter counts from 0 to the auto-reload value (content of the TIMx_ARR register), then restarts from 0 and generates a counter overflow event.

Setting the UG bit in the TIMx_EGR register (by software or by using the slave mode controller on TIM9/TIM12) also generates an update event.

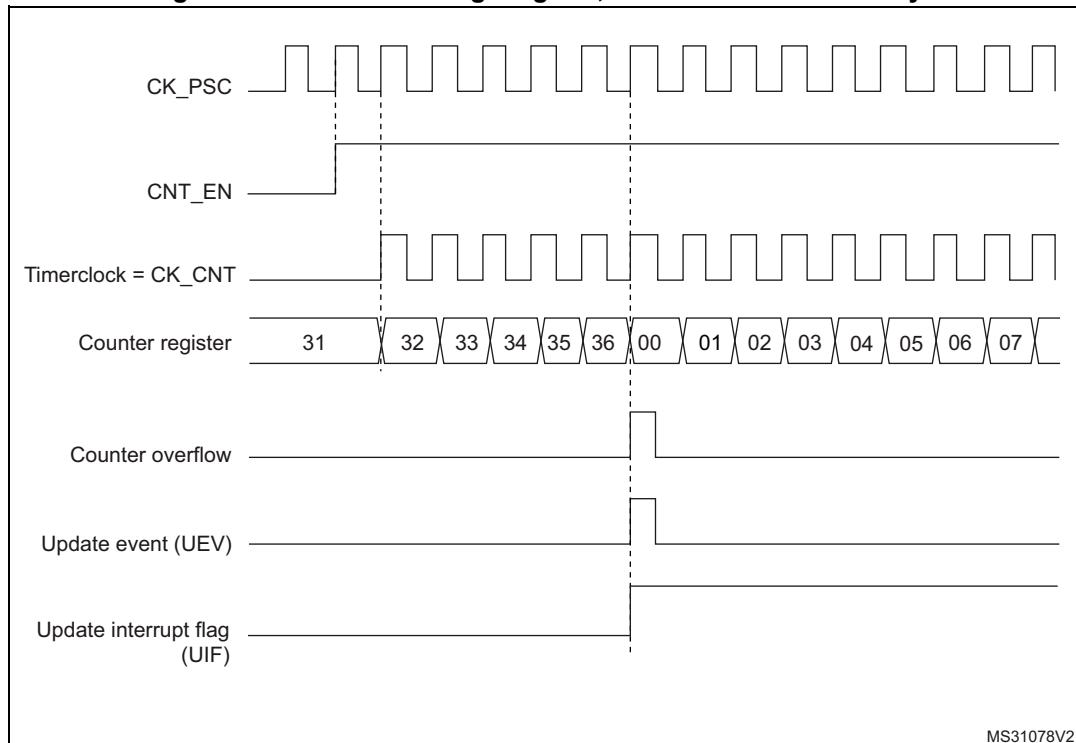
The UEV event can be disabled by software by setting the UDIS bit in the TIMx_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until the UDIS bit has been written to 0. However, the counter restarts from 0, as well as the counter of the prescaler (but the prescale rate does not change). In addition, if the URS bit (update request selection) in TIMx_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

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

- The auto-reload shadow register is updated with the preload value (TIMx_ARR),
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx_PSC register).

The following figures show some examples of the counter behavior for different clock frequencies when TIMx_ARR=0x36.

Figure 245. Counter timing diagram, internal clock divided by 1



MS31078V2

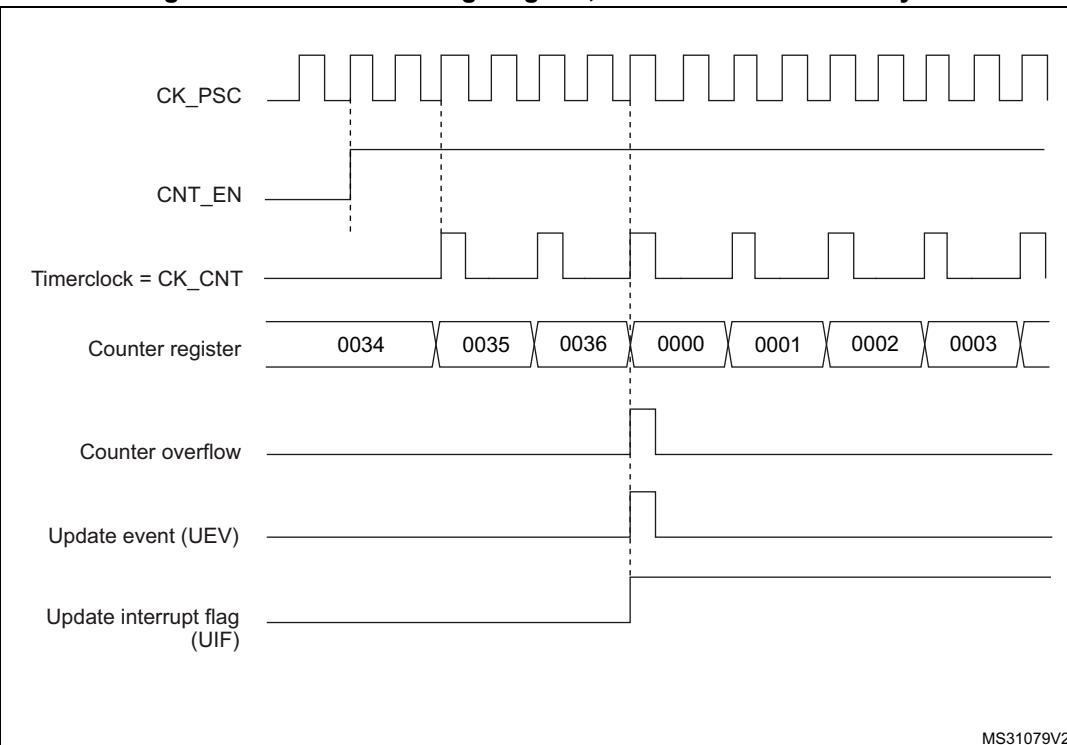
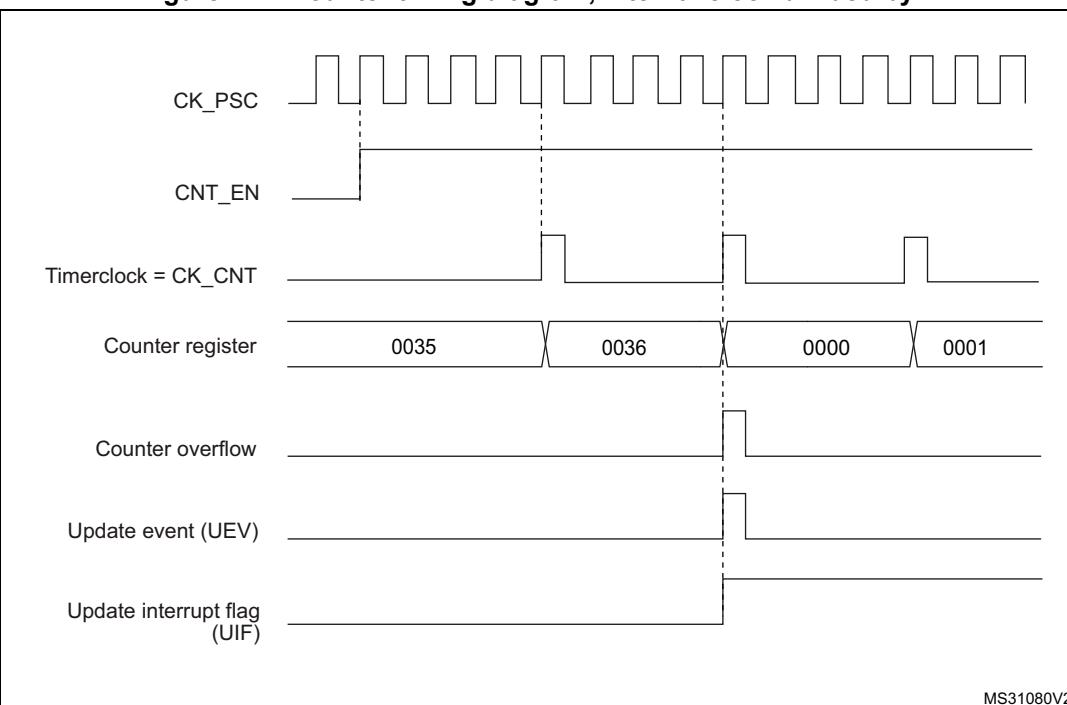
Figure 246. Counter timing diagram, internal clock divided by 2**Figure 247. Counter timing diagram, internal clock divided by 4**

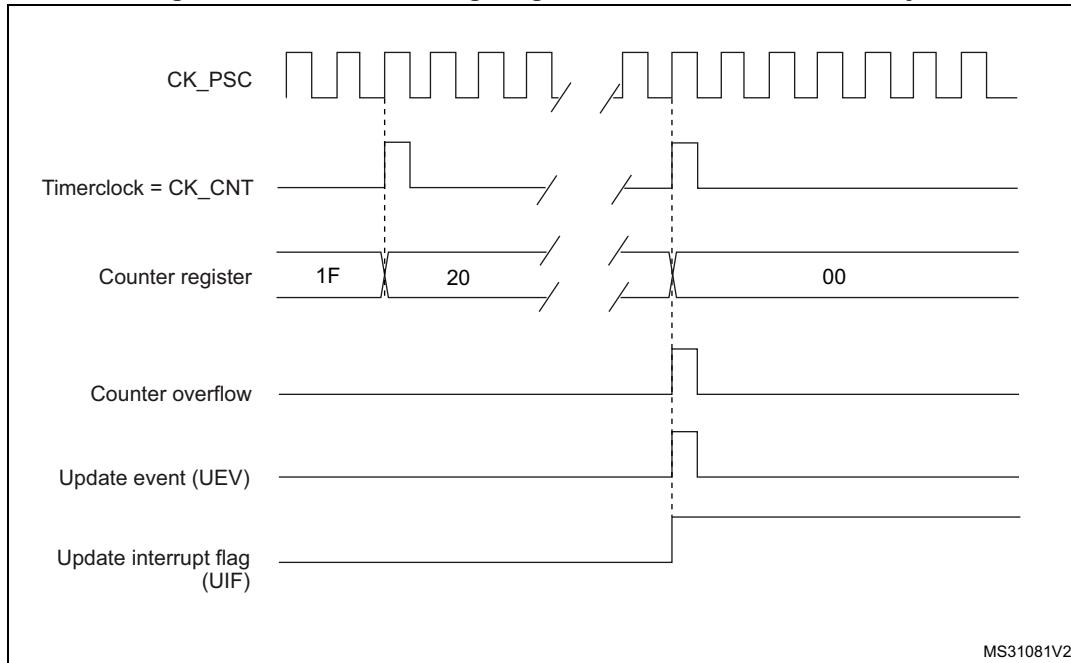
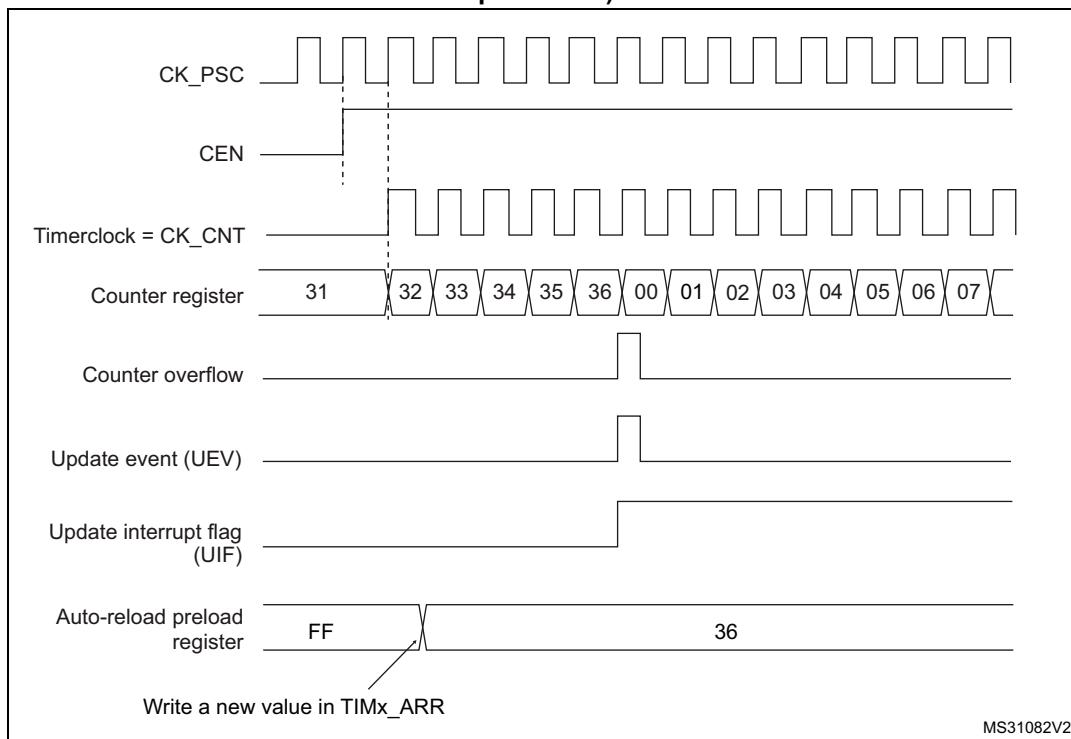
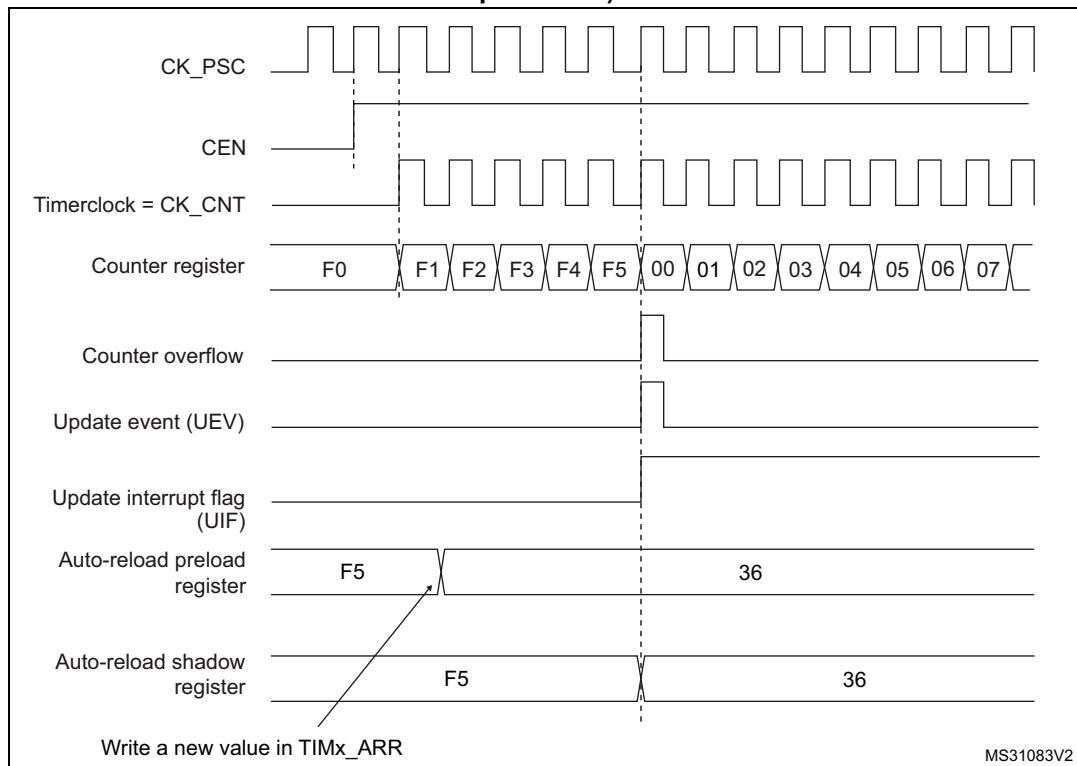
Figure 248. Counter timing diagram, internal clock divided by N**Figure 249. Counter timing diagram, update event when ARPE=0 (TIMx_ARR not preloaded)**

Figure 250. Counter timing diagram, update event when ARPE=1 (TIMx_ARR preloaded)



20.3.3 Clock selection

The counter clock can be provided by the following clock sources:

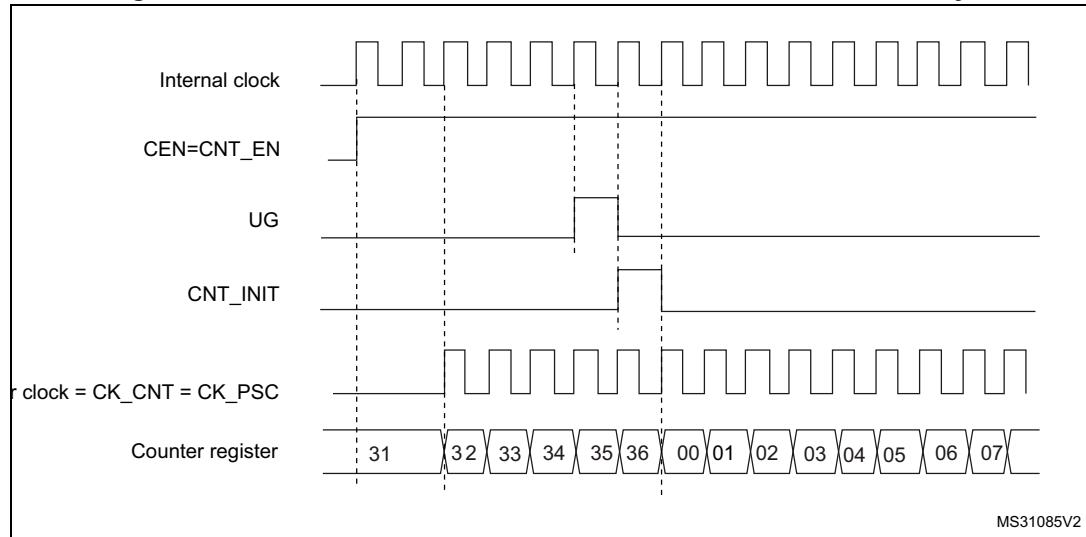
- Internal clock (CK_INT)
- External clock mode1 (for TIM9/TIM12): external input pin (TIx)
- Internal trigger inputs (ITRx) (for TIM9/TIM12): connecting the trigger output from another timer. For instance, another timer can be configured as a prescaler for TIM12. Refer to [Section : Using one timer as prescaler for another timer](#) for more details.

Internal clock source (CK_INT)

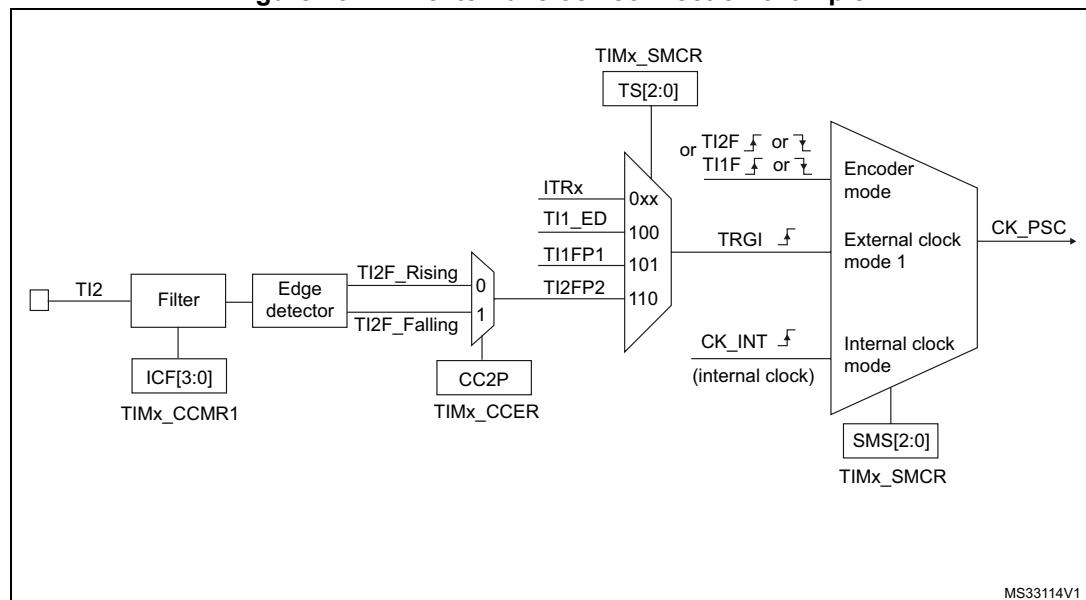
The internal clock source is the default clock source for TIM10/TIM11/TIM13/TIM14.

For TIM9/TIM12, the internal clock source is selected when the slave mode controller is disabled (SMS='000'). The CEN bit in the TIMx_CR1 register and the UG bit in the TIMx_EGR register are then used as control bits and can be changed only by software (except for UG which remains cleared). As soon as the CEN bit is programmed to 1, the prescaler is clocked by the internal clock CK_INT.

[Figure 251](#) shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

Figure 251. Control circuit in normal mode, internal clock divided by 1**External clock source mode 1 (TIM9/TIM12)**

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 252. TI2 external clock connection example

For example, to configure the upcounter to count in response to a rising edge on the TI2 input, use the following procedure:

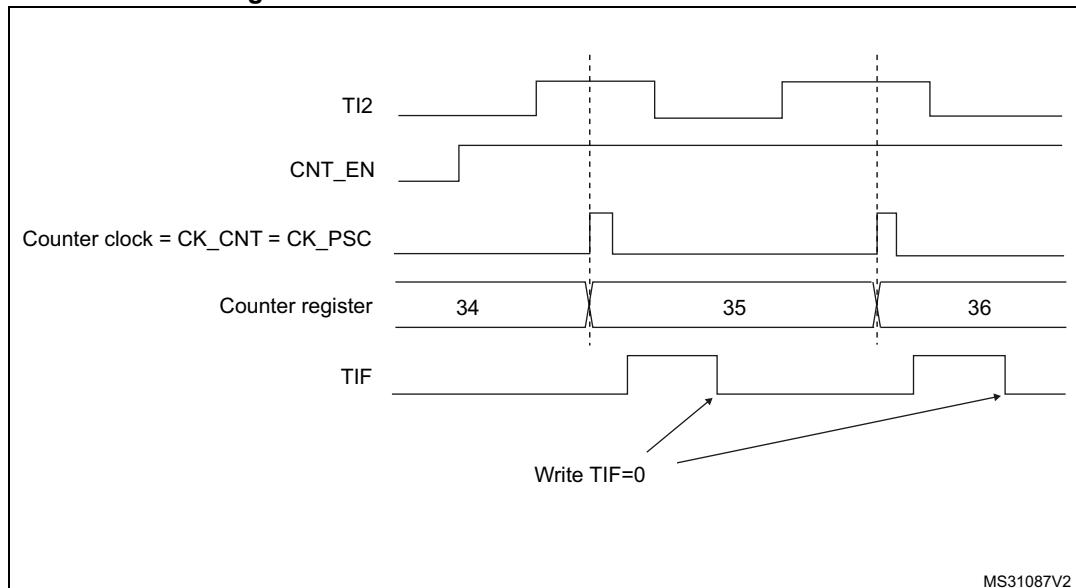
1. Configure channel 2 to detect rising edges on the TI2 input by writing CC2S = '01' in the TIMx_CCMR1 register.
2. Configure the input filter duration by writing the IC2F[3:0] bits in the TIMx_CCMR1 register (if no filter is needed, keep IC2F='0000').
3. Select the rising edge polarity by writing CC2P='0' and CC2NP='0' in the TIMx_CCER register.
4. Configure the timer in external clock mode 1 by writing SMS='111' in the TIMx_SMCR register.
5. Select TI2 as the trigger input source by writing TS='110' in the TIMx_SMCR register.
6. Enable the counter by writing CEN='1' in the TIMx_CR1 register.

Note: The capture prescaler is not used for triggering, so 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 253. Control circuit in external clock mode 1



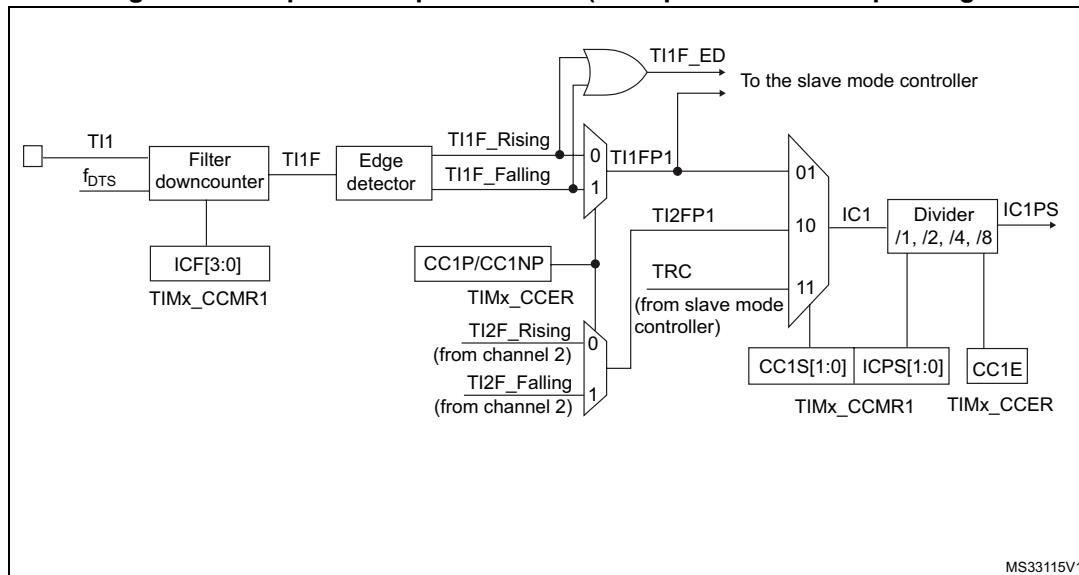
20.3.4 Capture/compare channels

Each Capture/Compare channel is built around a capture/compare register (including a shadow register), a input stage for capture (with digital filter, multiplexing and prescaler) and an output stage (with comparator and output control).

[Figure 254](#) to [Figure 256](#) 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 254. 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 255. Capture/compare channel 1 main circuit

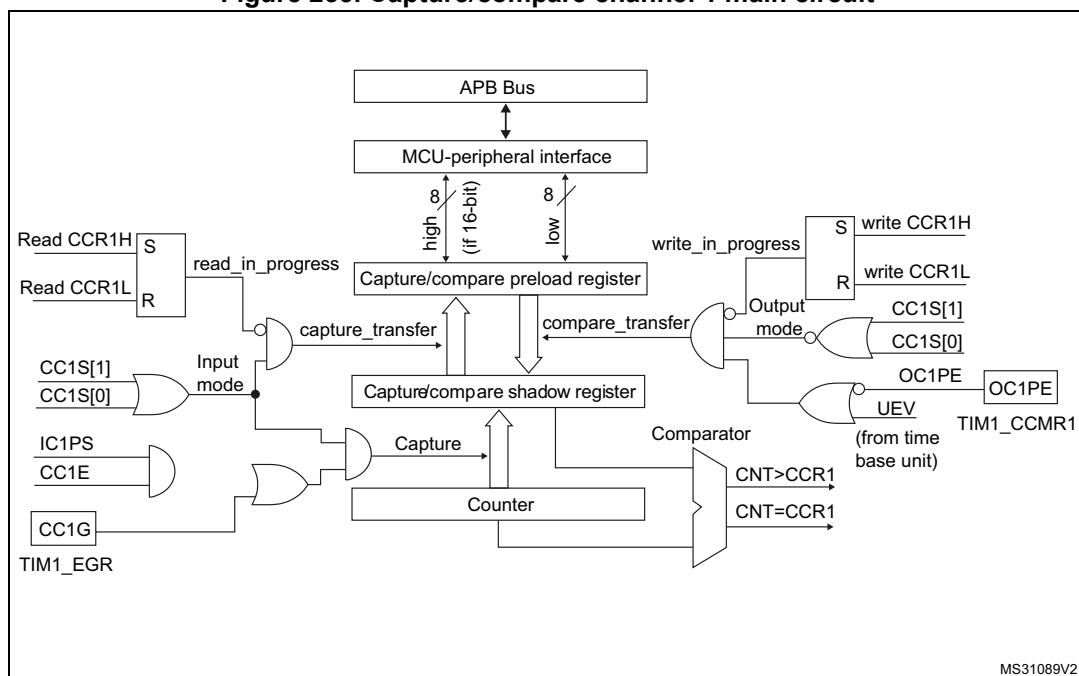
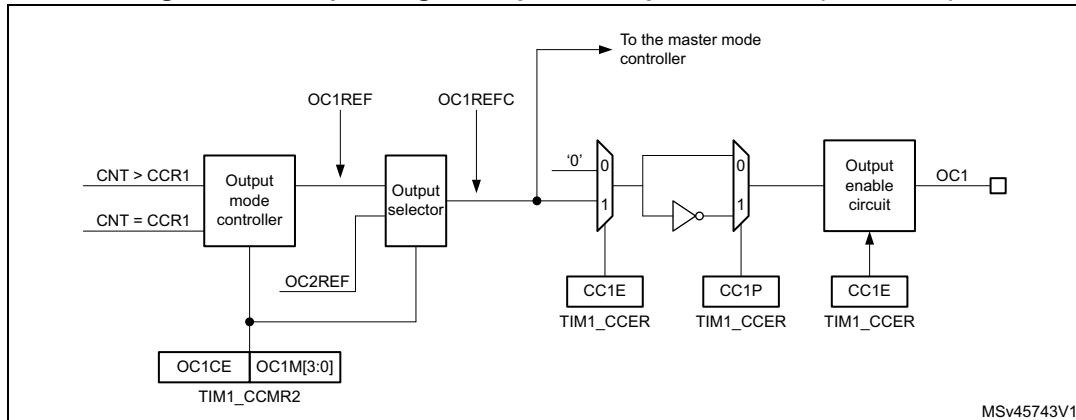


Figure 256. Output stage of capture/compare channel (channel 1)



MSv45743V1

The capture/compare block is made of one preload register and one shadow register. Write and read always access the preload register.

In capture mode, captures are actually done in the shadow register, which is copied into the preload register.

In compare mode, the content of the preload register is copied into the shadow register which is compared to the counter.

20.3.5 Input capture mode

In Input capture mode, the Capture/Compare Registers (TIMx_CCRx) are used to latch the value of the counter after a transition detected by the corresponding ICx signal. When a capture occurs, the corresponding CCxIF flag (TIMx_SR register) is set and an interrupt or a DMA request can be sent if they are enabled. If a capture occurs while the CCxIF flag was already high, then the over-capture flag CCxOF (TIMx_SR register) is set. CCxIF can be cleared by software by writing it to '0' or by reading the captured data stored in the TIMx_CCRx register. CCxOF is cleared when you write it to '0'.

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

1. Select the active input: TIMx_CCR1 must be linked to the TI1 input, so write the CC1S bits to '01' in the TIMx_CCMR1 register. As soon as CC1S becomes different from '00', the channel is configured in input mode and the TIMx_CCR1 register becomes read-only.
2. Program the input filter duration you need with respect to the signal you connect to the timer (by programming the ICxF bits in the TIMx_CCMRx register if the input is one of the TIx inputs). 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.
3. Select the edge of the active transition on the TI1 channel by programming CC1P and CC1NP bits to '00' in the TIMx_CCER register (rising edge in this case).
 4. Program the input prescaler. In our example, we wish the capture to be performed at each valid transition, so the prescaler is disabled (write IC1PS bits to '00' in the TIMx_CCMR1 register).
 5. Enable capture from the counter into the capture register by setting the CC1E bit in the TIMx_CCER register.
 6. If needed, enable the related interrupt request by setting the CC1IE bit in the TIMx_DIER register.

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.

In order to handle the overcapture, it is recommended to read the data before the overcapture flag. This is to avoid missing an overcapture which could happen after reading the flag and before reading the data.

Note: *IC interrupt requests can be generated by software by setting the corresponding CCxG bit in the TIMx_EGR register.*

20.3.6 PWM input mode (only for TIM9/TIM12)

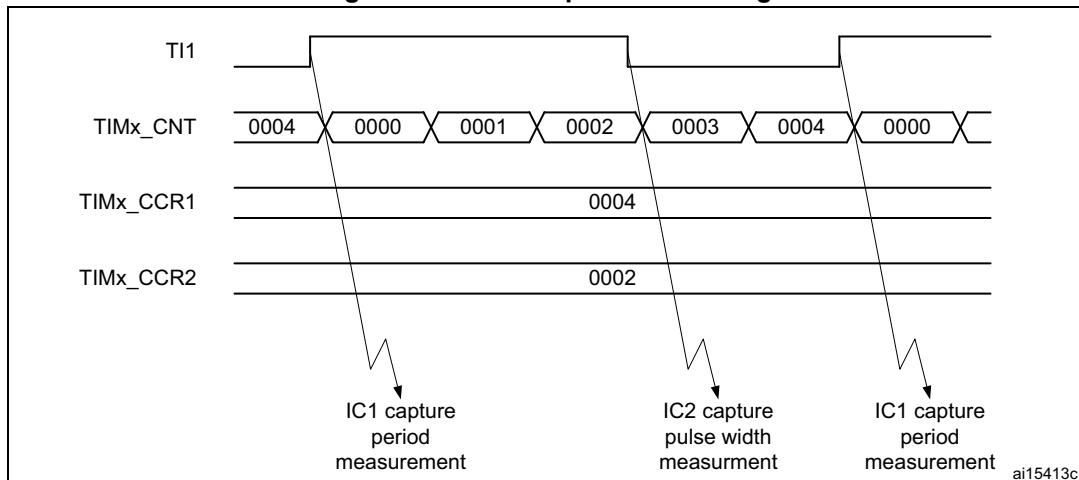
This mode is a particular case of input capture mode. The procedure is the same except:

- 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 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):

1. Select the active input for TIMx_CCR1: write the CC1S bits to '01' in the TIMx_CCMR1 register (TI1 selected).
2. Select the active polarity for TI1FP1 (used both for capture in TIMx_CCR1 and counter clear): program the CC1P and CC1NP bits to '00' (active on rising edge).
3. Select the active input for TIMx_CCR2: write the CC2S bits to '10' in the TIMx_CCMR1 register (TI1 selected).
4. Select the active polarity for TI1FP2 (used for capture in TIMx_CCR2): program the CC2P and CC2NP bits to '11' (active on falling edge).
5. Select the valid trigger input: write the TS bits to '101' in the TIMx_SMCR register (TI1FP1 selected).
6. Configure the slave mode controller in reset mode: write the SMS bits to '100' in the TIMx_SMCR register.
7. Enable the captures: write the CC1E and CC2E bits to '1' in the TIMx_CCER register.

Figure 257. PWM input mode timing



1. The PWM input mode can be used only with the TIMx_CH1/TIMx_CH2 signals due to the fact that only TI1FP1 and TI2FP2 are connected to the slave mode controller.

20.3.7 Forced output mode

In output mode (CCxS bits = '00' in the TIMx_CCMRx register), each output compare signal (OCxREF and then OCx) can be forced to active or inactive level directly by software, independently of any comparison between the output compare register and the counter.

To force an output compare signal (OCXREF/OCx) to its active level, you just need to write '0101' in the OCxM bits in the corresponding TIMx_CCMRx register. Thus OCXREF is forced high (OCxREF is always active high) and OCx get opposite value to CCxP polarity bit.

For example: CCxP='0' (OCx active high) => OCx is forced to high level.

The OCxREF signal can be forced low by writing the OCxM bits to '0100' 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 requests can be sent accordingly. This is described in the output compare mode section below.

20.3.8 Output compare mode

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

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

1. Assigns the corresponding output pin to a programmable value defined by the output compare mode (OCxM bits in the TIMx_CCMRx register) and the output polarity (CCxP bit in the TIMx_CCER register). The output pin can keep its level (OCxM='0000'), be set active (OCxM='0001'), be set inactive (OCxM='0010') or can toggle (OCxM='0011') on match.
2. Sets a flag in the interrupt status register (CCxIF bit in the TIMx_SR register).
3. Generates an interrupt if the corresponding interrupt mask is set (CCxIE bit in the TIMx_DIER register).

The TIMx_CCRx registers can be programmed with or without preload registers using the OCxPE bit in the TIMx_CCMRx register.

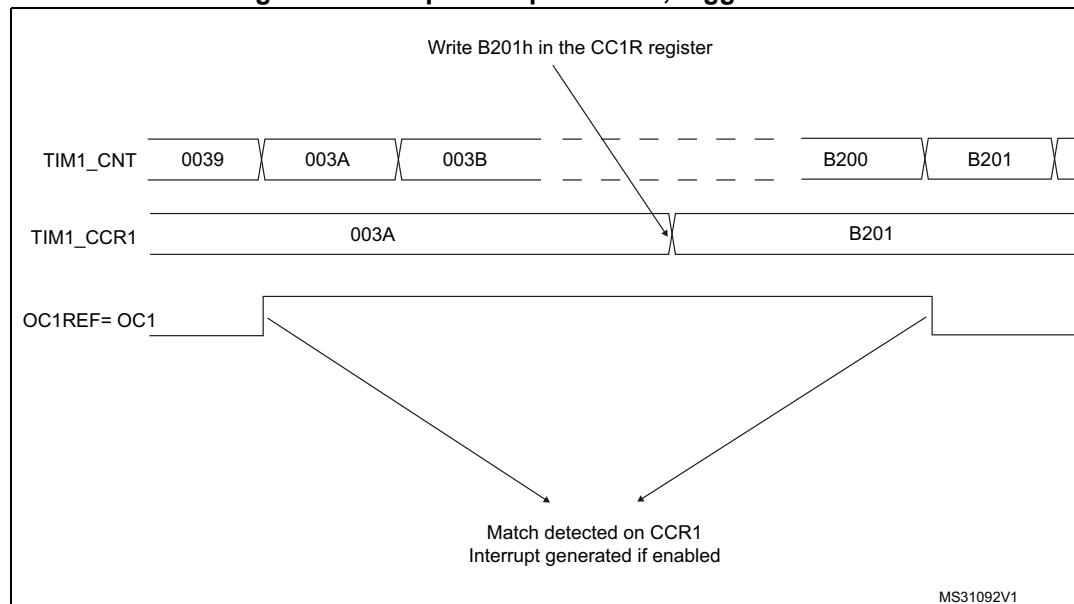
In output compare mode, the update event UEV has no effect on OCxREF and OCx output. The timing resolution is one count of the counter. Output compare mode can also be used to output a single pulse (in One-pulse mode).

Procedure:

1. Select the counter clock (internal, external, prescaler).
2. Write the desired data in the TIMx_ARR and TIMx_CCRx registers.
3. Set the CCxIE bit if an interrupt request is to be generated.
4. Select the output mode. For example:
 - Write OCxM = '0011' to toggle OCx output pin when CNT matches CCRx
 - Write OCxPE = '0' to disable preload register
 - Write CCxP = '0' to select active high polarity
 - Write CCxE = '1' to enable the output
5. Enable the counter by setting the CEN bit in the TIMx_CR1 register.

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 258](#).

Figure 258. Output compare mode, toggle on OC1.



20.3.9 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_CCRx register.

The PWM mode can be selected independently on each channel (one PWM per OCx output) by writing '0110' (PWM mode 1) or '0111' (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.

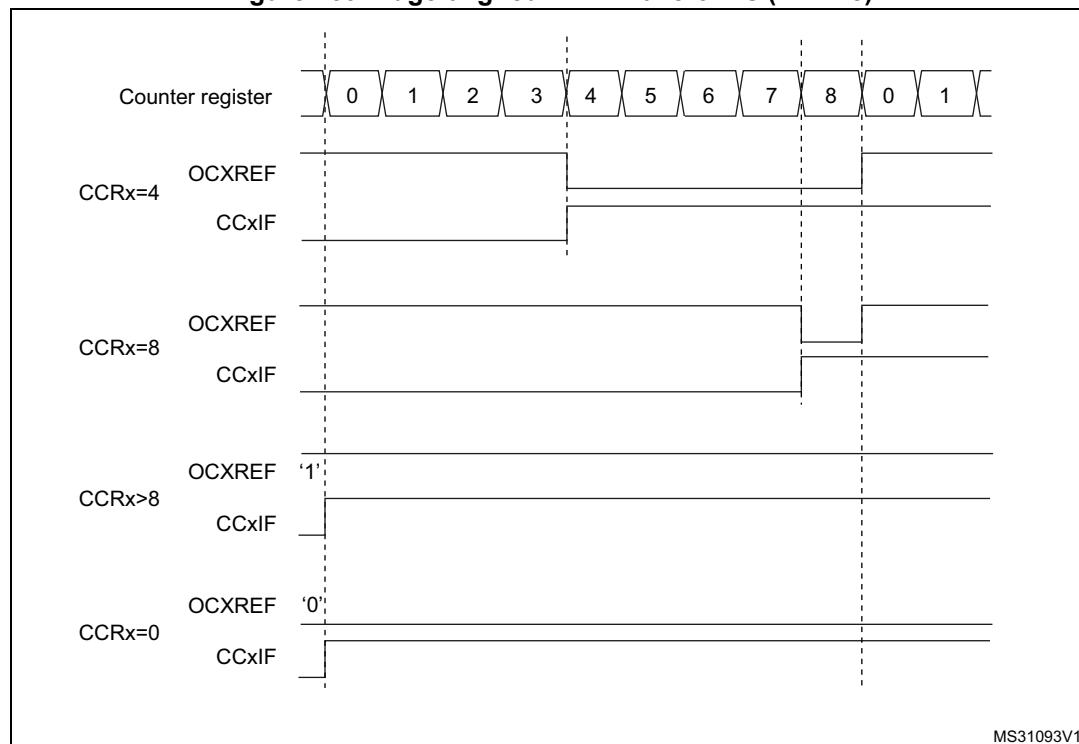
The `OCx` polarity is software programmable using the `CCxP` bit in the `TIMx_CCER` register. It can be programmed as active high or active low. The `OCx` output is enabled by the `CCxE` bit in the `TIMx_CCER` register. Refer to the `TIMx_CCRx` register description for more details.

In PWM mode (1 or 2), `TIMx_CNT` and `TIMx_CCRx` are always compared to determine whether $\text{TIMx_CNT} \leq \text{TIMx_CCRx}$.

The timer is able to generate PWM in edge-aligned mode only since the counter is upcounting.

In the following example, we consider PWM mode 1. The reference PWM signal `OCxREF` is high as long as $\text{TIMx_CNT} < \text{TIMx_CCRx}$ else it becomes low. If the compare value in `TIMx_CCRx` is greater than the auto-reload value (in `TIMx_ARR`) then `OCxREF` is held at '1'. If the compare value is 0 then `OCxRef` is held at '0'. [Figure 259](#) shows some edge-aligned PWM waveforms in an example where `TIMx_ARR=8`.

Figure 259. Edge-aligned PWM waveforms (ARR=8)



20.3.10 Combined PWM mode (TIM9/TIM12 only)

Combined PWM mode allows two edge or center-aligned PWM signals to be generated with programmable delay and phase shift between respective pulses. While the frequency is determined by the value of the `TIMx_ARR` register, the duty cycle and delay are determined

by the two $\text{TIMx_CCR}x$ registers. The resulting signals, $\text{OC}x\text{REFC}$, are made of an OR or AND logical combination of two reference PWMs:

- $\text{OC}1\text{REFC}$ (or $\text{OC}2\text{REFC}$) is controlled by the $\text{TIMx_CCR}1$ and $\text{TIMx_CCR}2$ registers

Combined PWM mode can be selected independently on two channels (one $\text{OC}x$ output per pair of CCR registers) by writing '1100' (Combined PWM mode 1) or '1101' (Combined PWM mode 2) in the $\text{OC}x\text{M}$ bits in the $\text{TIMx_CCMR}x$ register.

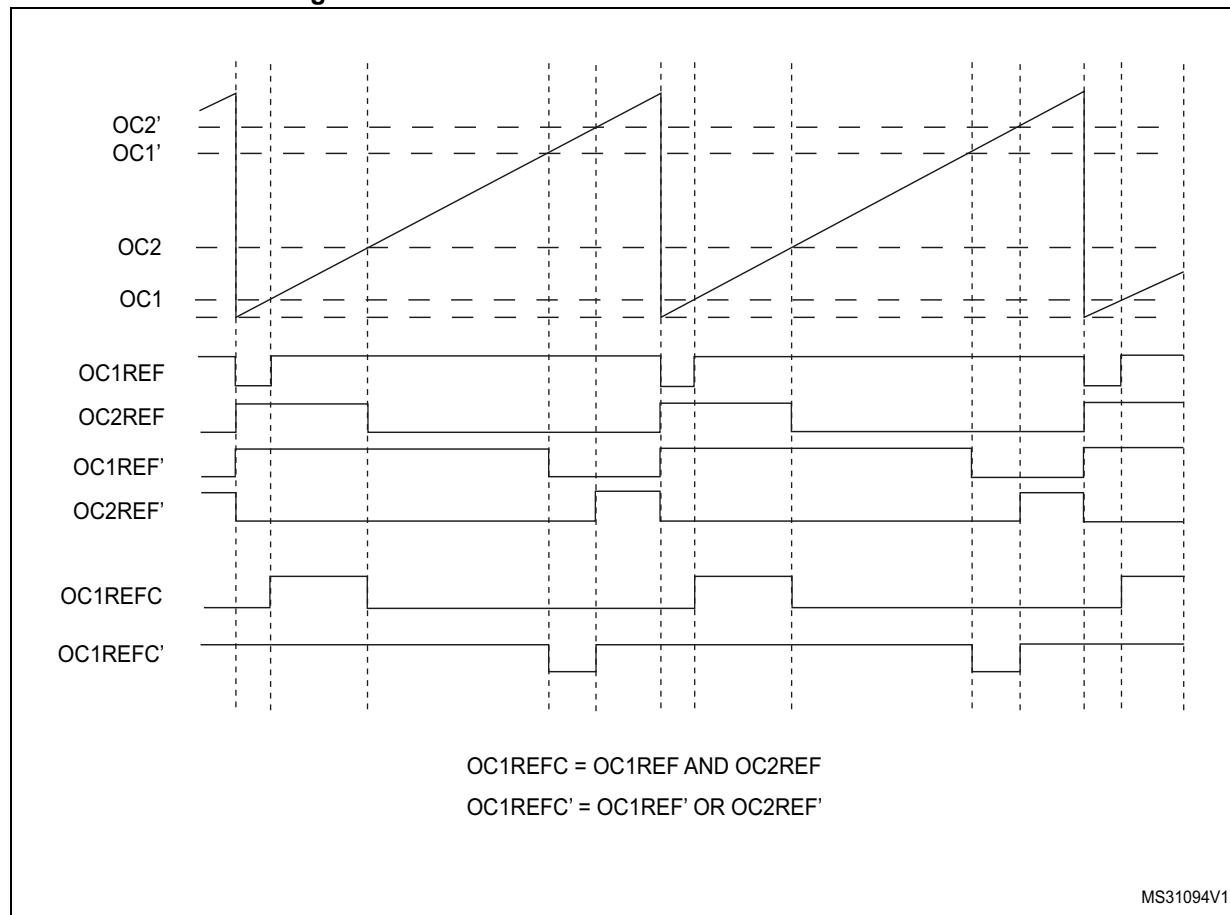
When a given channel is used as a combined PWM channel, its complementary channel must be configured in the opposite PWM mode (for instance, one in Combined PWM mode 1 and the other in Combined PWM mode 2).

Note: *The $\text{OC}x\text{M}[3:0]$ bit field is split into two parts for compatibility reasons, the most significant bit is not contiguous with the 3 least significant ones.*

Figure 260 represents an example of signals that can be generated using combined PWM mode, obtained with the following configuration:

- Channel 1 is configured in Combined PWM mode 2,
- Channel 2 is configured in PWM mode 1,

Figure 260. Combined PWM mode on channel 1 and 2



20.3.11 One-pulse mode

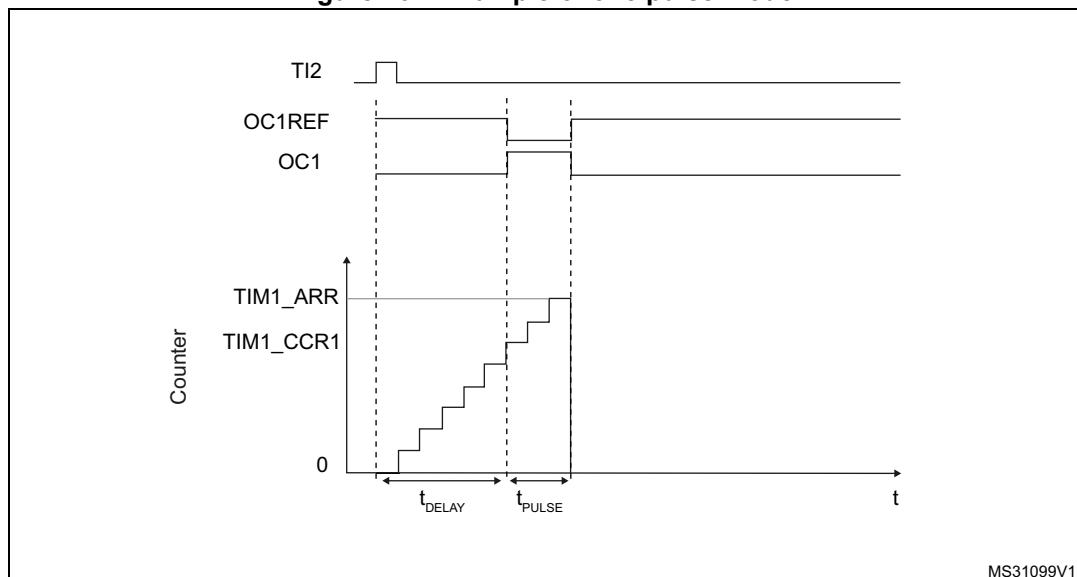
One-pulse mode (OPM) is a particular case of the previous modes. It allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length after a programmable delay.

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 as follows:

$$\text{CNT} < \text{CCRx} \leq \text{ARR} \text{ (in particular, } 0 < \text{CCRx})$$

Figure 261. 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.

Use TI2FP2 as trigger 1:

1. Map TI2FP2 to TI2 by writing CC2S='01' in the TIMx_CCMR1 register.
2. TI2FP2 must detect a rising edge, write CC2P='0' and CC2NP='0' in the TIMx_CCER register.
3. Configure TI2FP2 as trigger for the slave mode controller (TRGI) by writing TS='110' in the TIMx_SMCR register.
4. 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='0111' 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.

You only want 1 pulse (Single mode), 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). When OPM bit in the TIMx_CR1 register is set to '0', so the Repetitive Mode is selected.

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 TIMx_CCMRx register. Then OC_xRef (and OC_x) 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. OC_xFE acts only if the channel is configured in PWM1 or PWM2 mode.

20.3.12 Retriggerable one pulse mode (OPM) (TIM12 only)

This mode allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length, but with the following differences with non-retriggerable one pulse mode described in [Section 20.3.11: One-pulse mode](#):

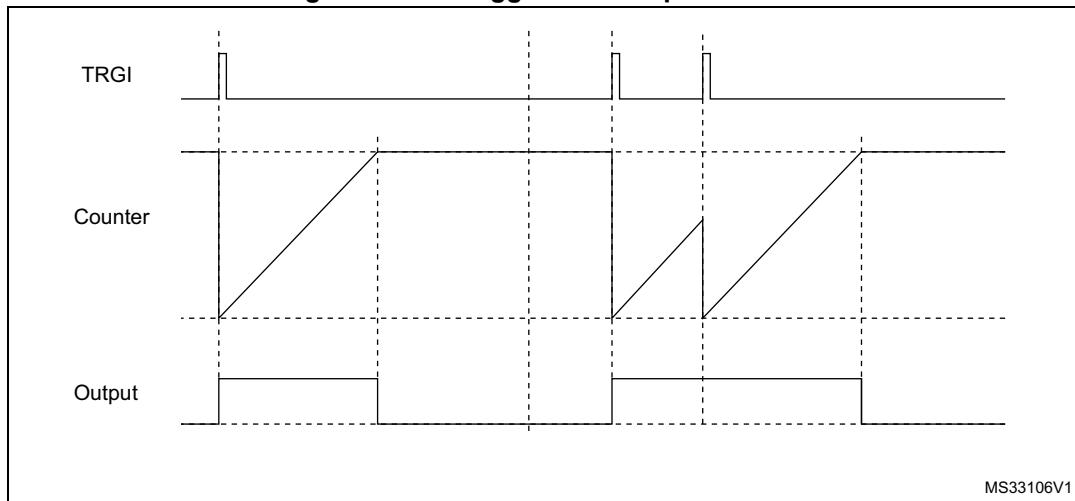
- The pulse starts as soon as the trigger occurs (no programmable delay)
- The pulse is extended if a new trigger occurs before the previous one is completed

The timer must be in Slave mode, with the bits SMS[3:0] = '1000' (Combined Reset + trigger mode) in the TIMx_SMCR register, and the OCxM[3:0] bits set to '1000' or '1001' for retrigerrable OPM mode 1 or 2.

If the timer is configured in up-counting mode, the corresponding CCR_x must be set to 0 (the ARR register sets the pulse length). If the timer is configured in down-counting mode, CCR_x must be above or equal to ARR.

Note: *The OCxM[3:0] and SMS[3:0] bit fields are split into two parts for compatibility reasons, the most significant bit are not contiguous with the 3 least significant ones.*

This mode must not be used with center-aligned PWM modes. It is mandatory to have CMS[1:0] = 00 in TIMx_CR1.

Figure 262. Retriggerable one pulse mode

20.3.13 UIF bit remapping

The IUFREMAP bit in the TIMx_CR1 register forces a continuous copy of the Update Interrupt Flag UIF into bit 31 of the timer counter register (TIMxCNT[31]). This allows to atomically read both the counter value and a potential roll-over condition signaled by the UIFCPY flag. In particular cases, it can ease the calculations by avoiding race conditions caused for instance by a processing shared between a background task (counter reading) and an interrupt (Update Interrupt).

There is no latency between the assertions of the UIF and UIFCPY flags.

20.3.14 TIM9/TIM12 external trigger synchronization

The TIM9/TIM12 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_CC Rx) are updated.

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

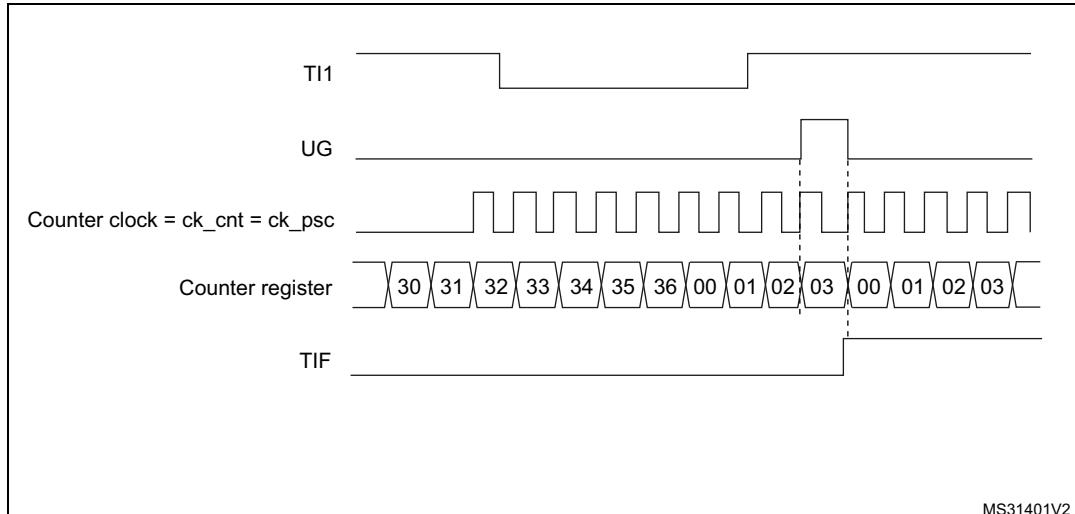
1. Configure the channel 1 to detect rising edges on TI1. Configure the input filter duration (in this example, we don't need any filter, so we keep IC1F='0000'). The capture prescaler is not used for triggering, so you don't need to configure it. The CC1S bits select the input capture source only, CC1S = '01' in the TIMx_CCMR1 register. Program CC1P and CC1NP to '00' in TIMx_CCER register to validate the polarity (and detect rising edges only).
2. 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.
3. 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 can be sent if enabled (depending on the TIE bit 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 263. Control circuit in reset mode



Slave mode: Gated mode

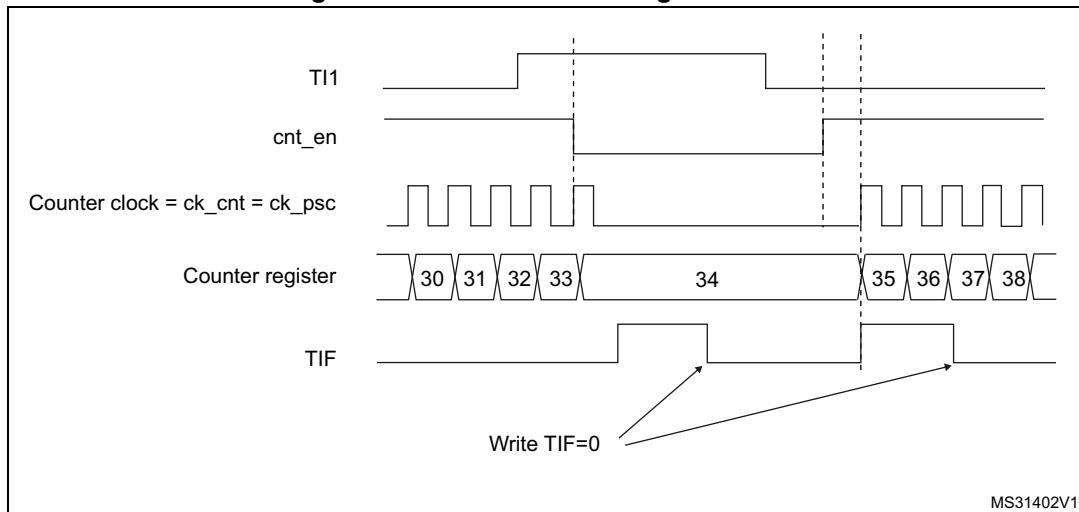
The counter can be enabled depending on the level of a selected input.

In the following example, the upcounter counts only when TI1 input is low:

1. Configure the channel 1 to detect low levels on TI1. Configure the input filter duration (in this example, we don't need any filter, so we keep IC1F='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. Program CC1P='1' and CC1NP='0' in TIMx_CCER register to validate the polarity (and detect low level only).
2. 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.
3. 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 264. Control circuit in gated mode**Slave mode: Trigger mode**

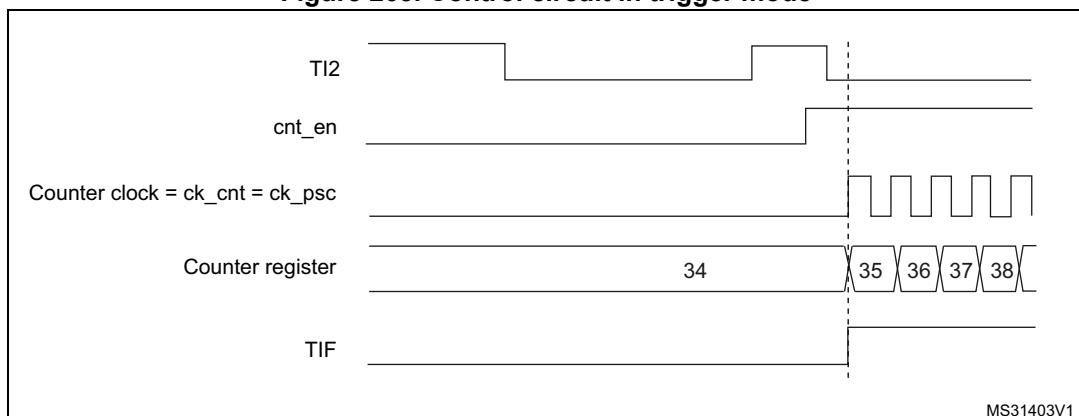
The counter can start in response to an event on a selected input.

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

1. 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 TIMx_CCMR1 register. Program CC2P='1' and CC2NP='0' in TIMx_CCER register to validate the polarity (and detect low level only).
2. 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 265. Control circuit in trigger mode

20.3.15 Slave mode – combined reset + trigger mode

In this case, a rising edge of the selected trigger input (TRGI) reinitializes the counter, generates an update of the registers, and starts the counter.

This mode is used for one-pulse mode.

20.3.16 Timer synchronization (TIM9/TIM12)

The TIM timers are linked together internally for timer synchronization or chaining. Refer to [Section 19.3.19: Timer synchronization](#) for details.

Note: *The clock of the slave timer must be enabled prior to receive events from the master timer, and must not be changed on-the-fly while triggers are received from the master timer.*

20.3.17 Debug mode

When the microcontroller enters debug mode (Cortex®-M7 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 40.16.2: Debug support for timers, watchdog, bxCAN and I2C](#).

20.4 TIM9/TIM12 registers

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

The peripheral registers have to be written by half-words (16 bits) or words (32 bits). Read accesses can be done by bytes (8 bits), half-words (16 bits) or words (32 bits).

20.4.1 TIMx control register 1 (TIMx_CR1)(x = 9, 12)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	UIFRE MAP	Res.	CKD[1:0]		ARPE	Res.	Res.	Res.	OPM	URS	UDIS	CEN
				rw		rw	rw	rw				rw	rw	rw	rw

Bits 15:12 Reserved, must be kept at reset value.

Bit 11 **UIFREMAP**: UIF status bit remapping

- 0: No remapping. UIF status bit is not copied to TIMx_CNT register bit 31.
- 1: Remapping enabled. UIF status bit is copied to TIMx_CNT register bit 31.

Bit 10 Reserved, must be kept at reset value.

Bits 9:8 **CKD[1:0]**: Clock division

This bit-field indicates the division ratio between the timer clock (CK_INT) frequency and sampling clock used by the digital filters (Tlx).

- 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 6:4 Reserved, must be kept at reset value.

Bit 3 **OPM**: One-pulse mode

- 0: Counter is not stopped on the update event
 1: Counter stops counting on the next update event (clearing the CEN bit).

Bit 2 **URS**: Update request source

This bit is set and cleared by software to select the UEV event sources.

- 0: Any of the following events generates an update interrupt if enabled. These events can be:
- Counter overflow
 - Setting the UG bit
 - Update generation through the slave mode controller

- 1: Only counter overflow generates an update interrupt if enabled.

Bit 1 **UDIS**: Update disable

This bit is set and cleared by software to enable/disable update event (UEV) generation.

- 0: UEV enabled. An UEV is generated by one of the following events:

- Counter overflow
- Setting the UG bit

Buffered registers are then loaded with their preload values.

- 1: UEV disabled. No UEV is generated, shadow registers keep their value (ARR, PSC, CCRx). The counter and the prescaler are reinitialized if the UG bit is set.

Bit 0 **CEN**: Counter enable

- 0: Counter disabled

- 1: Counter enabled

CEN is cleared automatically in one-pulse mode, when an update event occurs.

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

20.4.2 TIMx slave mode control register (TIMx_SMCR)(x = 9, 12)

Address offset: 0x08

Reset value: 0x0000 0000

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

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

Bit 16 **SMS[3]**: Slave mode selection - bit 3

Refer to SMS description - bits 2:0

Bits 15:8 Reserved, must be kept at reset value.

Bit 7 **MSM**: Master/Slave mode

- 0: No action

- 1: The effect of an event on the trigger input (TRGI) is delayed to allow a perfect synchronization between the current timer and its slaves (through TRGO). It is useful in order to synchronize several timers on a single external event.

Bits 6:4 **TS[2:0]**: Trigger selection

This bitfield selects the trigger input to be used to synchronize the counter.

- 000: Internal Trigger 0 (ITR0)
- 001: Internal Trigger 1 (ITR1)
- 010: Internal Trigger 2 (ITR2)
- 011: Internal Trigger 3 (ITR3)
- 100: TI1 Edge Detector (TI1F_ED)
- 101: Filtered Timer Input 1 (TI1FP1)
- 110: Filtered Timer Input 2 (TI2FP2)
- 111: Reserved.

See [Table 122: TIMx internal trigger connection on page 701](#) for more details on the meaning of ITRx for each timer.

Note: These bits must be changed only when they are not used (e.g. when SMS='000') to avoid wrong edge detections at the transition.

Bit 3 Reserved, must be kept at reset value.

Bits 16, 2, 1, 0 **SMS[3:0]**: Slave mode selection

When external signals are selected the active edge of the trigger signal (TRGI) is linked to the polarity selected on the external input (see Input Control register and Control Register description).

0000: Slave mode disabled - if CEN = '1' then the prescaler is clocked directly by the internal clock.

0001: Reserved

0010: Reserved

0011: Reserved

0100: Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter and generates an update of the registers.

0101: 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.

0110: 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.

0111: External Clock Mode 1 - Rising edges of the selected trigger (TRGI) clock the counter.

1000: Combined reset + trigger mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter, generates an update of the registers and starts the counter.

Other codes: reserved.

Note: The gated mode must not be used if TI1F_ED is selected as the trigger input (TS='100'). Indeed, TI1F_ED outputs 1 pulse for each transition on TI1F, whereas the gated mode checks the level of the trigger signal.

Note: The clock of the slave timer must be enabled prior to receive events from the master timer, and must not be changed on-the-fly while triggers are received from the master timer.

Table 122. TIMx internal trigger connection

Slave TIM	ITR0 (TS = '000')	ITR1 (TS = '001')	ITR2 (TS = '010')	ITR3 (TS = '011')
TIM9	TIM2	TIM3	TIM10_OC	TIM11_OC
TIM12	TIM4	TIM5	TIM13_OC	TIM14_OC

20.4.3 TIMx Interrupt enable register (TIMx_DIER)(x = 9, 12)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TIE	Res.	Res.	Res.	CC2IE	CC1IE	UIE								
									rw				rw	rw	rw

Bits 15:7 Reserved, must be kept at reset value.

Bit 6 **TIE**: Trigger interrupt enable

- 0: Trigger interrupt disabled.
- 1: Trigger interrupt enabled.

Bits 5:3 Reserved, must be kept at reset value.

Bit 2 **CC2IE**: Capture/Compare 2 interrupt enable

- 0: CC2 interrupt disabled.
- 1: CC2 interrupt enabled.

Bit 1 **CC1IE**: Capture/Compare 1 interrupt enable

- 0: CC1 interrupt disabled.
- 1: CC1 interrupt enabled.

Bit 0 **UIE**: Update interrupt enable

- 0: Update interrupt disabled.
- 1: Update interrupt enabled.

20.4.4 TIMx status register (TIMx_SR)(x = 9, 12)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	CC2OF	CC1OF	Res.	Res.	TIF	Res.	Res.	Res.	CC2IF	CC1IF	UIF
					rc_w0	rc_w0			rc_w0				rc_w0	rc_w0	rc_w0

Bits 15:11 Reserved, must be kept at reset value.

Bit 10 **CC2OF**: Capture/compare 2 overcapture flag

refer to CC1OF description

Bit 9 **CC1OF**: Capture/Compare 1 overcapture flag

This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to '0'.

0: No overcapture has been detected.

1: The counter value has been captured in TIMx_CCR1 register while CC1IF flag was already set

Bits 8:7 Reserved, must be kept at reset value.

Bit 6 **TIF**: Trigger interrupt flag

This flag is set by hardware on trigger event (active edge detected on TRGI input when the slave mode controller is enabled in all modes but gated mode. It is set when the counter starts or stops when gated mode is selected. It is cleared by software.

0: No trigger event occurred.

1: Trigger interrupt pending.

Bits 5:3 Reserved, must be kept at reset value.

Bit 2 **CC2IF**: Capture/Compare 2 interrupt flag

refer to CC1IF description

Bit 1 **CC1IF**: Capture/compare 1 interrupt flag**If channel CC1 is configured as output:**

This flag is set by hardware when the counter matches the compare value. It is cleared by software.

0: No match.

1: The content of the counter TIMx_CNT matches the content of the TIMx_CCR1 register. When the contents of TIMx_CCR1 are greater than the contents of TIMx_ARR, the CC1IF bit goes high on the counter overflow.

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 and if UDIS='0' in the TIMx_CR1 register.
- When CNT is reinitialized by software using the UG bit in TIMx_EGR register, if URS='0' and UDIS='0' in the TIMx_CR1 register.
- When CNT is reinitialized by a trigger event (refer to the synchro control register description), if URS='0' and UDIS='0' in the TIMx_CR1 register.

20.4.5 TIMx event generation register (TIMx_EGR)(x = 9, 12)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TG	Res.	Res.	Res.	CC2G	CC1G	UG								

Bits 15:7 Reserved, must be kept at reset value.

Bit 6 **TG**: Trigger generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: The TIF flag is set in the TIMx_SR register. Related interrupt can occur if enabled

Bits 5:3 Reserved, must be kept at reset value.

Bit 2 **CC2G**: Capture/compare 2 generation
refer to CC1G description

Bit 1 **CC1G**: Capture/compare 1 generation

This bit is set by software to generate an event, it is automatically cleared by hardware.

0: No action

1: A capture/compare event is generated on channel 1:

If channel CC1 is configured as output:

the CC1IF flag is set, the corresponding interrupt is sent if enabled.

If channel CC1 is configured as input:

The current counter value is captured in the TIMx_CCR1 register. The CC1IF flag is set, the corresponding interrupt is sent if enabled. The CC1OF flag is set if the CC1IF flag was already high.

Bit 0 **UG**: Update generation

This bit can be set by software, it is automatically cleared by hardware.

0: No action

1: Re-initializes the counter and generates an update of the registers. The prescaler counter is also cleared and the prescaler ratio is not affected. The counter is cleared.

20.4.6 TIMx capture/compare mode register 1 (TIMx_CCMR1)(x = 9, 12)

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 in this register have different functions in input and output modes. For a given bit, OCxx describes its function when the channel is configured in output mode, ICxx describes its function when the channel is configured in input mode. So you must take care that the same bit can have different meanings for the input stage and the output stage.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC2M [3]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC1M [3]
							Res.								Res.
							rw								rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]	Res.	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	rw

Output compare mode

Bits 31:25 Reserved, always read as 0

Bit 24 **OC2M[3]**: Output Compare 2 mode - bit 3

Refer to OC2M description on bits 14:12

Bits 23:17 Reserved, always read as 0

Bit 16 **OC1M[3]**: Output Compare 1 mode - bit 3

Refer to OC1M description on bits 6:4

Bit 15 Reserved, must be kept at reset value.

Bits 14:12 **OC2M[2:0]**: Output compare 2 mode

Refer to OC1M[3:0] for bit description.

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 bitfield defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output

01: CC2 channel is configured as input, IC2 is mapped on 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 works only if an internal trigger input is selected through the TS bit (TIMx_SMCR register)

Note: The CC2S bits are writable only when the channel is OFF (CC2E = 0 in TIMx_CCER).

Bit 7 Reserved, must be kept at reset value.

Bits 6:4 **OC1M[3:0]**: Output compare 1 mode (refer to bit 16 for OC1M[3])

These bits define the behavior of the output reference signal OC1REF from which OC1 is derived. OC1REF is active high whereas the active level of OC1 depends on the CC1P.

0000: Frozen - The comparison between the output compare register TIMx_CCR1 and the counter TIMx_CNT has no effect on the outputs.(this mode is used to generate a timing base).

0001: Set channel 1 to active level on match. The OC1REF signal is forced high when the TIMx_CNT counter matches the capture/compare register 1 (TIMx_CCR1).

0010: Set channel 1 to inactive level on match. The OC1REF signal is forced low when the TIMx_CNT counter matches the capture/compare register 1 (TIMx_CCR1).

0011: Toggle - OC1REF toggles when TIMx_CNT=TIMx_CCR1

0100: Force inactive level - OC1REF is forced low

0101: Force active level - OC1REF is forced high

0110: PWM mode 1 - channel 1 is active as long as TIMx_CNT<TIMx_CCR1 else it is inactive

0111: PWM mode 2 - channel 1 is inactive as long as TIMx_CNT<TIMx_CCR1 else it is active

1000: Retriggerable OPM mode 1 - The channel is active until a trigger event is detected (on TRGI signal). Then, a comparison is performed as in PWM mode 1 and the channels becomes active again at the next update.

1001: Retriggerable OPM mode 2 - The channel is inactive until a trigger event is detected (on TRGI signal). Then, a comparison is performed as in PWM mode 2 and the channels becomes inactive again at the next update.

1010: Reserved,

1011: Reserved,

1100: Combined PWM mode 1 - OC1REF has the same behavior as in PWM mode 1. OC1REFC is the logical OR between OC1REF and OC2REF.

1101: Combined PWM mode 2 - OC1REF has the same behavior as in PWM mode 2. OC1REFC is the logical AND between OC1REF and OC2REF.

1110: Reserved,

1111: Reserved

Note: In PWM mode 1 or 2, the OCREF level changes only when the result of the comparison changes or when the output compare mode switches from “frozen” mode to “PWM” mode.

Bit 3 OC1PE: Output compare 1 preload enable

0: Preload register on TIMx_CCR1 disabled. TIMx_CCR1 can be written at anytime, the new value is taken into account immediately

1: Preload register on TIMx_CCR1 enabled. Read/Write operations access the preload register. TIMx_CCR1 preload value is loaded into the active register at each update event

Note: The PWM mode can be used without validating the preload register only in one-pulse mode (OPM bit set in the 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 the counter and CCR1 values even when the trigger is ON. The minimum delay to activate the CC1 output when an edge occurs on the trigger input is 5 clock cycles

1: An active edge on the trigger input acts like a compare match on the CC1 output. Then, OC is set to the compare level independently of the result of the comparison. Delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. OC1FE acts only if the channel is configured in PWM1 or PWM2 mode.

Bits 1:0 CC1S[1:0]: Capture/Compare 1 selection

This bitfield defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output

01: CC1 channel is configured as input, IC1 is mapped on 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 works only if an internal trigger input is selected through the TS bit (TIMx_SMCR register)

Note: The 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 bitfield defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output

01: CC2 channel is configured as input, IC2 is mapped on 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 works only if an internal trigger input is selected through the TS bit (TIMx_SMCR register)

Note: The CC2S bits are writable only when the channel is OFF (CC2E = 0 in TIMx_CCER).

Bits 7:4 **IC1F**: Input capture 1 filter

This bitfield defines the frequency used to sample the TI1 input and the length of the digital filter applied to TI1. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output:

0000: No filter, sampling is done at f_{DTS}

0001: $f_{SAMPLING} = f_{CK_INT}$, N=2

0010: $f_{SAMPLING} = f_{CK_INT}$, N=4

0011: $f_{SAMPLING} = f_{CK_INT}$, N=8

0100: $f_{SAMPLING} = f_{DTS}/2$, N=6

0101: $f_{SAMPLING} = f_{DTS}/2$, N=8

0110: $f_{SAMPLING} = f_{DTS}/4$, N=6

0111: $f_{SAMPLING} = f_{DTS}/4$, N=8

1000: $f_{SAMPLING} = f_{DTS}/8$, N=6

1001: $f_{SAMPLING} = f_{DTS}/8$, N=8

1010: $f_{SAMPLING} = f_{DTS}/16$, N=5

1011: $f_{SAMPLING} = f_{DTS}/16$, N=6

1100: $f_{SAMPLING} = f_{DTS}/16$, N=8

1101: $f_{SAMPLING} = f_{DTS}/32$, N=5

1110: $f_{SAMPLING} = f_{DTS}/32$, N=6

1111: $f_{SAMPLING} = f_{DTS}/32$, N=8

Bits 3:2 **IC1PSC**: Input capture 1 prescaler

This bitfield defines the ratio of the prescaler acting on the CC1 input (IC1).

The prescaler is reset as soon as CC1E='0' (TIMx_CCER register).

00: no prescaler, capture is done each time an edge is detected on the capture input

01: capture is done once every 2 events

10: capture is done once every 4 events

11: capture is done once every 8 events

Bits 1:0 **CC1S[1:0]**: Capture/Compare 1 selection

This bitfield defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output

01: CC1 channel is configured as input, IC1 is mapped on 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: The CC1S bits are writable only when the channel is OFF (CC1E = 0 in TIMx_CCER).

20.4.7 TIMx capture/compare enable register (TIMx_CCER)(x = 9, 12)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CC2NP	Res.	CC2P	CC2E	CC1NP	Res.	CC1P	CC1E							
								rw		rw	rw	rw		rw	rw

Bits 15:8 Reserved, must be kept at reset value.

Bit 7 **CC2NP**: Capture/Compare 2 output Polarity

Refer to CC1NP description

Bit 6 Reserved, must be kept at reset value.

Bit 5 **CC2P**: Capture/Compare 2 output Polarity

Refer to CC1P description

Bit 4 **CC2E**: Capture/Compare 2 output enable

Refer to CC1E description

Bit 3 **CC1NP**: Capture/Compare 1 complementary output Polarity

CC1 channel configured as output: CC1NP must be kept cleared

CC1 channel configured as input: CC1NP is used in conjunction with CC1P to define TI1FP1/TI2FP1 polarity (refer to CC1P description).

Bit 2 Reserved, must be kept at reset value.

Bit 1 **CC1P**: Capture/Compare 1 output Polarity.

CC1 channel configured as output:

0: OC1 active high.

1: OC1 active low.

CC1 channel configured as input:

The CC1P and CC1NP bits select TI1FP1 polarity for capture operations.

00: non-inverted/rising edge

Circuit is sensitive to TIxFP1 rising edge (capture, trigger in reset, external clock or trigger mode), TIxFP1 is not inverted (trigger in gated mode, encoder mode).

01: inverted/falling edge

Circuit is sensitive to TIxFP1 falling edge (capture, trigger in reset, external clock or trigger mode), TIxFP1 is inverted (trigger in gated mode, encoder mode).

10: reserved, do not use this configuration.

11: noninverted/both edges

Circuit is sensitive to both TIxFP1 rising and falling edges (capture, trigger in reset, external clock or trigger mode), TIxFP1 is not inverted (trigger in gated mode). This configuration must not be used for encoder mode.

Bit 0 **CC1E**: Capture/Compare 1 output enable.

CC1 channel configured as output:

0: Off - OC1 is not active.

1: On - OC1 signal is output on the corresponding output pin.

CC1 channel configured as input:

This bit determines if a capture of the counter value can actually be done into the input capture/compare register 1 (TIMx_CCR1) or not.

0: Capture disabled.

1: Capture enabled.

Table 123. 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 states of the external I/O pins connected to the standard OCx channels depend on the state of the OCx channel and on the GPIO registers.

20.4.8 TIMx counter (TIMx_CNT)(x = 9, 12)

Address offset: 0x24

Reset value: 0x0000 0000

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

Bit 31 **UIFCPY**: UIF Copy

This bit is a read-only copy of the UIF bit in the TIMx_ISR register.

Bits 30:16 Reserved, must be kept at reset value.

Bits 15:0 **CNT[15:0]**: Counter value

20.4.9 TIMx prescaler (TIMx_PSC)(x = 9, 12)

Address offset: 0x28

Reset value: 0x0000

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

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK_CNT is equal to $f_{CK_PSC} / (PSC[15:0] + 1)$.

PSC contains the value to be loaded into the active prescaler register at each update event. (including when the counter is cleared through UG bit of TIMx_EGR register or through trigger controller when configured in “reset mode”).

20.4.10 TIMx auto-reload register (TIMx_ARR)(x = 9, 12)

Address offset: 0x2C

Reset value: 0xFFFF

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

Bits 15:0 **ARR[15:0]**: Auto-reload value

ARR is the value to be loaded into the actual auto-reload register.

Refer to the [Section 20.3.1: Time-base unit on page 678](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

20.4.11 TIMx capture/compare register 1 (TIMx_CCR1)(x = 9, 12)

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 into the actual capture/compare 1 register (preload value). It is loaded permanently if the preload feature is not selected in the TIMx_CCMR1 register (OC1PE bit). Else the preload value is copied into the active capture/compare 1 register when an update event occurs.

The active capture/compare register contains the value to be compared to the TIMx_CNT counter and signaled on the OC1 output.

If channel CC1 is configured as input:

CCR1 is the counter value transferred by the last input capture 1 event (IC1).

20.4.12 TIMx capture/compare register 2 (TIMx_CCR2)(x = 9, 12)

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 into the actual capture/compare 2 register (preload value). It is loaded permanently if the preload feature is not selected in the TIMx_CCMR2 register (OC2PE bit). Else the preload value is copied into the active capture/compare 2 register when an update event occurs.

The active capture/compare register contains the value to be compared to the TIMx_CNT counter and signalled on the OC2 output.

If channel CC2 is configured as input:

CCR2 is the counter value transferred by the last input capture 2 event (IC2).

20.4.13 TIM9/TIM12 register map

TIM9/TIM12 registers are mapped as 16-bit addressable registers as described below:

Table 124. TIM9/TIM12 register map and reset values

Offset	Register name	Reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	TIMx_CR1	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value																																	
0x08	TIMx_SMCR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																	
0x0C	TIMx_DIER	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																	
0x10	TIMx_SR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																	
0x14	TIMx_EGR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																	
0x18	TIMx_CCMR1 Output Compare mode	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																	
	TIMx_CCMR1 Input Capture mode	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																	
0x1C	Reserved																																	
0x20	TIMx_CCER	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																	
0x24	TIMx_CNT	UIFCPY	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x28	TIMx_PSC	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
	Reset value																																	
0x2C	TIMx_ARR	Reserved	ARR[15:0]																	CC2F[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	0		

Table 124. TIM9/TIM12 register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x30	Reserved																																
0x34	TIMx_CCR1	Res.																															
	Reset value																																
0x38	TIMx_CCR2	Res.																															
	Reset value																																

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

20.5 TIM10/TIM11/TIM13/TIM14 registers

The peripheral registers have to be written by half-words (16 bits) or words (32 bits). Read accesses can be done by bytes (8 bits), half-words (16 bits) or words (32 bits).

20.5.1 TIMx control register 1 (TIMx_CR1)(x = 10, 11, 13, 14)

Address offset: 0x000

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	UIFRE MAP	Res.	CKD[1:0]		ARPE	Res.	Res.	Res.	OPM	URS	UDIS	CEN
				rw		rw	rw	rw				rw	rw	rw	rw

Bits 15:12 Reserved, must be kept at reset value.

Bit 11 **UIFREMAP**: UIF status bit remapping

- 0: No remapping. UIF status bit is not copied to TIMx_CNT register bit 31.
- 1: Remapping enabled. UIF status bit is copied to TIMx_CNT register bit 31.

Bit 10 Reserved, must be kept at reset value.

Bits 9:8 **CKD[1:0]**: Clock division

This bit-field indicates the division ratio between the timer clock (CK_INT) frequency and sampling clock used by the digital filters (Tlx),

- 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 6:4 Reserved, must be kept at reset value.

Bit 3 **OPM**: One-pulse mode

- 0: Counter is not stopped on the update event
- 1: Counter stops counting on the next update event (clearing the CEN bit).

Bit 2 URS: Update request source

This bit is set and cleared by software to select the update interrupt (UEV) sources.

0: Any of the following events generate an UEV if enabled:

- Counter overflow
- Setting the UG bit

1: Only counter overflow generates an UEV if enabled.

Bit 1 UDIS: Update disable

This bit is set and cleared by software to enable/disable update interrupt (UEV) event generation.

0: UEV enabled. An UEV is generated by one of the following events:

- Counter overflow
- Setting the UG bit

Buffered registers are then loaded with their preload values.

1: UEV disabled. No UEV is generated, shadow registers keep their value (ARR, PSC, CCRx). The counter and the prescaler are reinitialized if the UG bit is set.

Bit 0 CEN: Counter enable

0: Counter disabled

1: Counter enabled

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

20.5.2 TIMx Interrupt enable register (TIMx_DIER)(x = 10, 11, 13, 14)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CC1IE	UIE													

Bits 15:2 Reserved, must be kept at reset value.

Bit 1 CC1IE: Capture/Compare 1 interrupt enable

0: CC1 interrupt disabled

1: CC1 interrupt enabled

Bit 0 UIE: Update interrupt enable

0: Update interrupt disabled

1: Update interrupt enabled

20.5.3 TIMx status register (TIMx_SR)(x = 10, 11, 13, 14)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	CC1OF	Res.	CC1IF	UIF						

Bits 15:10 Reserved, must be kept at reset value.

Bit 9 **CC1OF**: Capture/Compare 1 overcapture flag

This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to '0'.

0: No overcapture has been detected.

1: The counter value has been captured in TIMx_CCR1 register while CC1IF flag was already set

Bits 8:2 Reserved, must be kept at reset value.

Bit 1 **CC1IF**: Capture/compare 1 interrupt flag

If channel CC1 is configured as output:

This flag is set by hardware when the counter matches the compare value. It is cleared by software.

0: No match.

1: The content of the counter TIMx_CNT matches the content of the TIMx_CCR1 register. When the contents of TIMx_CCR1 are greater than the contents of TIMx_ARR, the CC1IF bit goes high on the counter overflow.

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 and if UDIS='0' in the TIMx_CR1 register.
- When CNT is reinitialized by software using the UG bit in TIMx_EGR register, if URS='0' and UDIS='0' in the TIMx_CR1 register.

20.5.4 TIMx event generation register (TIMx_EGR)(x = 10, 11, 13, 14)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CC1G	UG													

Bits 15:2 Reserved, must be kept at reset value.

Bit 1 CC1G: Capture/compare 1 generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: A capture/compare event is generated on channel 1:

If channel CC1 is configured as output:

CC1IF flag is set, Corresponding interrupt or 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 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.

20.5.5 TIMx capture/compare mode register 1 (TIMx_CCMR1)(x = 10, 11, 13, 14)

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.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Res.	Res.	Res.	Res.	Res.	Res.	OC1M [3]	Res.									
																rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]										
Res.	IC1F[3:0]			IC1PSC[1:0]												
									rw	rw	rw	rw	rw	rw	rw	rw

Output compare mode

Bits 31:17 Reserved, always read as 0

Bit 16 **OC1M[3]**: Output Compare 1 mode - bit 3

Refer to OC1M description on bits 6:4

Bits 15:7 Reserved, must be kept at reset value.

Bits 6:4 **OC1M[3:0]**: Output compare 1 mode (refer to bit 16 for OC1M[3])

These bits define the behavior of the output reference signal OC1REF from which OC1 is derived. OC1REF is active high whereas OC1 active level depends on CC1P bit.

0000: Frozen. The comparison between the output compare register TIMx_CCR1 and the counter TIMx_CNT has no effect on the outputs.

0001: 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).

0010: 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).

0011: Toggle - OC1REF toggles when TIMx_CNT = TIMx_CCR1.

0100: Force inactive level - OC1REF is forced low.

0101: Force active level - OC1REF is forced high.

0110: PWM mode 1 - Channel 1 is active as long as TIMx_CNT < TIMx_CCR1 else inactive.

0111: PWM mode 2 - Channel 1 is inactive as long as TIMx_CNT < TIMx_CCR1 else active

Others: Reserved

Note: In PWM mode 1 or 2, the OCREF level changes when the result of the comparison changes or when the output compare mode switches from frozen to PWM mode.

Bit 3 **OC1PE**: Output compare 1 preload enable

0: Preload register on TIMx_CCR1 disabled. TIMx_CCR1 can be written at anytime, the new value is taken in account immediately.

1: Preload register on TIMx_CCR1 enabled. Read/Write operations access the preload register. TIMx_CCR1 preload value is loaded in the active register at each update event.

Note: The PWM mode can be used without validating the preload register only in one pulse mode (OPM bit set in TIMx_CR1 register). Else the behavior is not guaranteed.

Bit 2 **OC1FE**: Output compare 1 fast enable

This bit is used to accelerate the effect of an event on the trigger input on the CC output.

0: CC1 behaves normally depending on counter and CCR1 values even when the trigger is ON. The minimum delay to activate CC1 output when an edge occurs on the trigger input is 5 clock cycles.

1: An active edge on the trigger input acts like a compare match on CC1 output. OC is then set to the compare level independently of the result of the comparison. Delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. OC1FE acts only if the channel is configured in PWM1 or PWM2 mode.

Bits 1:0 **CC1S[1:0]**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output.

01: CC1 channel is configured as input, IC1 is mapped on TI1.

10: 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 15:8 Reserved, must be kept at reset value.

Bits 7:4 **IC1F**: Input capture 1 filter

This bit-field defines the frequency used to sample TI1 input and the length of the digital filter applied to TI1. The digital filter is made of an event counter in which N consecutive events are needed to validate a transition on the output:

- 0000: No filter, sampling is done at f_{DTS}
- 1000: $f_{SAMPLING} = f_{DTS}/8$, N=6
- 0001: $f_{SAMPLING} = f_{CK_INT}$, N=21
- 0001: $f_{SAMPLING} = f_{DTS}/8$, N=8
- 0010: $f_{SAMPLING} = f_{CK_INT}$, N=41
- 0010: $f_{SAMPLING} = f_{DTS}/16$, N=5
- 0011: $f_{SAMPLING} = f_{CK_INT}$, N=81
- 0011: $f_{SAMPLING} = f_{DTS}/16$, N=6
- 0100: $f_{SAMPLING} = f_{DTS}/2$, N=61
- 1100: $f_{SAMPLING} = f_{DTS}/16$, N=8
- 0101: $f_{SAMPLING} = f_{DTS}/2$, N=81
- 1101: $f_{SAMPLING} = f_{DTS}/32$, N=5
- 0110: $f_{SAMPLING} = f_{DTS}/4$, N=61
- 1110: $f_{SAMPLING} = f_{DTS}/32$, N=6
- 0111: $f_{SAMPLING} = f_{DTS}/4$, N=81
- 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' (TIMx_CCER register).

00: no prescaler, capture is done each time an edge is detected on the capture input

01: capture is done once every 2 events

10: capture is done once every 4 events

11: capture is done once every 8 events

Bits 1:0 **CC1S[1:0]**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output

01: CC1 channel is configured as input, IC1 is mapped on TI1

10: Reserved

11: Reserved

Note: CC1S bits are writable only when the channel is OFF (CC1E = 0 in TIMx_CCER).

20.5.6 TIMx capture/compare enable register (TIMx_CCER)(x = 10, 11, 13, 14)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CC1NP	Res.	CC1P	CC1E											

Bits 15:4 Reserved, must be kept at reset value.

Bit 3 **CC1NP**: Capture/Compare 1 complementary output Polarity.

CC1 channel configured as output: CC1NP must be kept cleared.

CC1 channel configured as input: CC1NP bit is used in conjunction with CC1P to define TI1FP1 polarity (refer to CC1P description).

Bit 2 Reserved, must be kept at reset value.

Bit 1 **CC1P**: Capture/Compare 1 output Polarity.

CC1 channel configured as output:

0: OC1 active high

1: OC1 active low

CC1 channel configured as input:

The CC1P and CC1NP bits select TI1FP1 polarity for capture operations.

00: noninverted/rising edge

Circuit is sensitive to TI1FP1 rising edge (capture mode), TI1FP1 is not inverted.

01: inverted/falling edge

Circuit is sensitive to TI1FP1 falling edge (capture mode), TI1FP1 is inverted.

10: reserved, do not use this configuration.

11: noninverted/both edges

Circuit is sensitive to both TI1FP1 rising and falling edges (capture mode), TI1FP1 is not inverted.

Bit 0 **CC1E**: Capture/Compare 1 output enable.

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 125. Output control bit for standard OCx channels

CCxE bit	OCx output state
0	Output Disabled (OCx='0', OCx_EN='0')
1	OCx=OCxREF + Polarity, OCx_EN='1'

Note: The state of the external I/O pins connected to the standard OCx channels depends on the OCx channel state and the GPIO registers.

20.5.7 TIMx counter (TIMx_CNT)(x = 10, 11, 13, 14)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UIF CPY	Res.														
rw															
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **UIFCPY**: UIF Copy

This bit is a read-only copy of the UIF bit in the TIMx_ISR register.

Bits 30:16 Reserved, must be kept at reset value.

Bits 15:0 **CNT[15:0]**: Counter value

20.5.8 TIMx prescaler (TIMx_PSC)(x = 10, 11, 13, 14)

Address offset: 0x28

Reset value: 0x0000

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

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK_CNT is equal to $f_{CK_PSC} / (PSC[15:0] + 1)$.

PSC contains the value to be loaded in the active prescaler register at each update event.

(including when the counter is cleared through UG bit of TIMx_EGR register or through trigger controller when configured in “reset mode”).

20.5.9 TIMx auto-reload register (TIMx_ARR)(x = 10, 11, 13, 14)

Address offset: 0x2C

Reset value: 0xFFFF

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

Bits 15:0 **ARR[15:0]**: Auto-reload value

ARR is the value to be loaded in the actual auto-reload register.

Refer to [Section 20.3.1: Time-base unit on page 678](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

20.5.10 TIMx capture/compare register 1 (TIMx_CCR1)(x = 10, 11, 13, 14)

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).

20.5.11 TIM11 option register 1 (TIM11_OR)

Address offset: 0x50

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	TI1_RMP[1:0]															

Bits 15:2 Reserved, must be kept at reset value.

Bits 1:0 **TI1_RMP[1:0]**: TIM11 Input 1 remapping capability

Set and cleared by software.

00: TIM11 Channel1 is connected to GPIO (refer to the Alternate function mapping)

01: Reserved

10: HSE internal clock (1MHz for RTC) is connected to TIM11_CH1 input for measurement purposes

11: MCO1 is connected to TIM11_CH1 input

20.5.12 TIM10/TIM11/TIM13/TIM14 register map

TIMx registers are mapped as 16-bit addressable registers as described in the tables below:

Table 126. TIM10/TIM11/TIM13/TIM14 register map and reset values

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	TIMx_CR1	Res.	CKD [1:0]	ARPE	Res.	Res.	Res.	OPM	URS	UDIS	CEN																						
	Reset value																				0	0	0	0		0	0	0	0	0	0	0	
0x04 to 0x08	Reserved																																
0x0C	TIMx_DIER	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CC1IE	UIE																							
	Reset value																												0	0	0	0	

Table 126. TIM10/TIM11/TIM13/TIM14 register map and reset values (continued)

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

21 Basic timers (TIM6/TIM7)

21.1 TIM6/TIM7 introduction

The basic timers TIM6 and TIM7 consist of a 16-bit auto-reload counter driven by a programmable prescaler.

They may be used as generic timers for time-base generation but they are also specifically used to drive the digital-to-analog converter (DAC). In fact, the timers are internally connected to the DAC and are able to drive it through their trigger outputs.

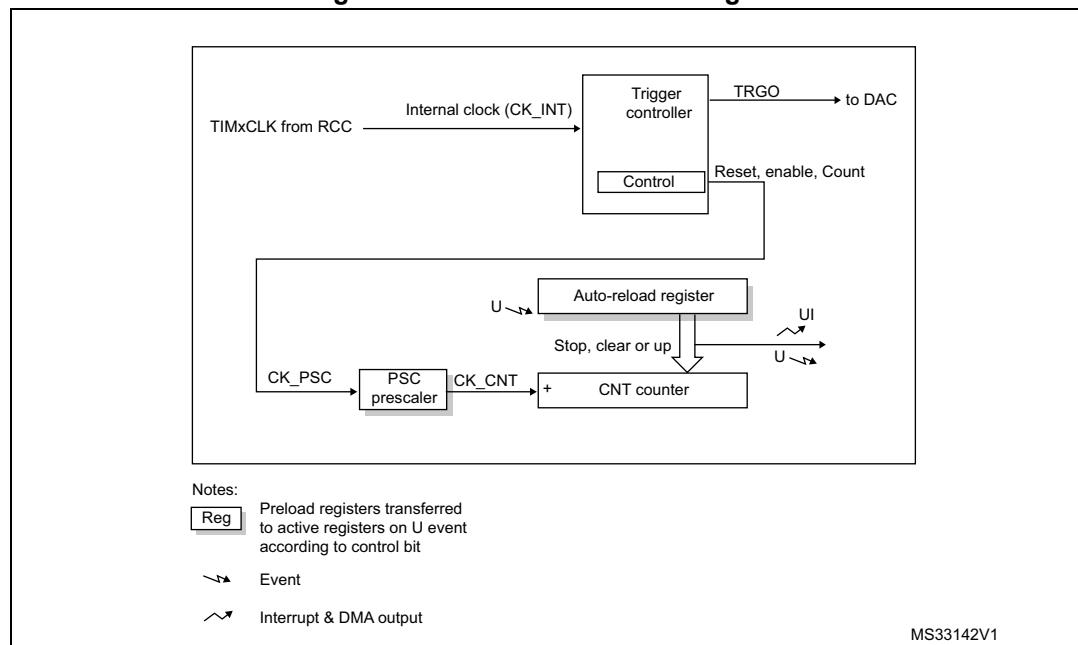
The timers are completely independent, and do not share any resources.

21.2 TIM6/TIM7 main features

Basic timer (TIM6/TIM7) features include:

- 16-bit auto-reload upcounter
- 16-bit programmable prescaler used to divide (also “on the fly”) the counter clock frequency by any factor between 1 and 65535
- Synchronization circuit to trigger the DAC
- Interrupt/DMA generation on the update event: counter overflow

Figure 266. Basic timer block diagram



21.3 TIM6/TIM7 functional description

21.3.1 Time-base unit

The main block of the programmable timer is a 16-bit upcounter with its related auto-reload register. The counter clock can be divided by a prescaler.

The counter, the auto-reload register and the prescaler register can be written or read by software. This is true even when the counter is running.

The time-base unit includes:

- Counter Register (TIMx_CNT)
- Prescaler Register (TIMx_PSC)
- Auto-Reload Register (TIMx_ARR)

The auto-reload register is preloaded. The preload register is accessed each time an attempt is made to write or read the auto-reload register. The contents of the preload register are transferred into the shadow register permanently or at each update event UEV, depending on the auto-reload preload enable bit (ARPE) in the TIMx_CR1 register. The update event is sent when the counter reaches the overflow value and if the UDIS bit equals 0 in the TIMx_CR1 register. It can also be generated by software. The generation of the update event is described in detail for each configuration.

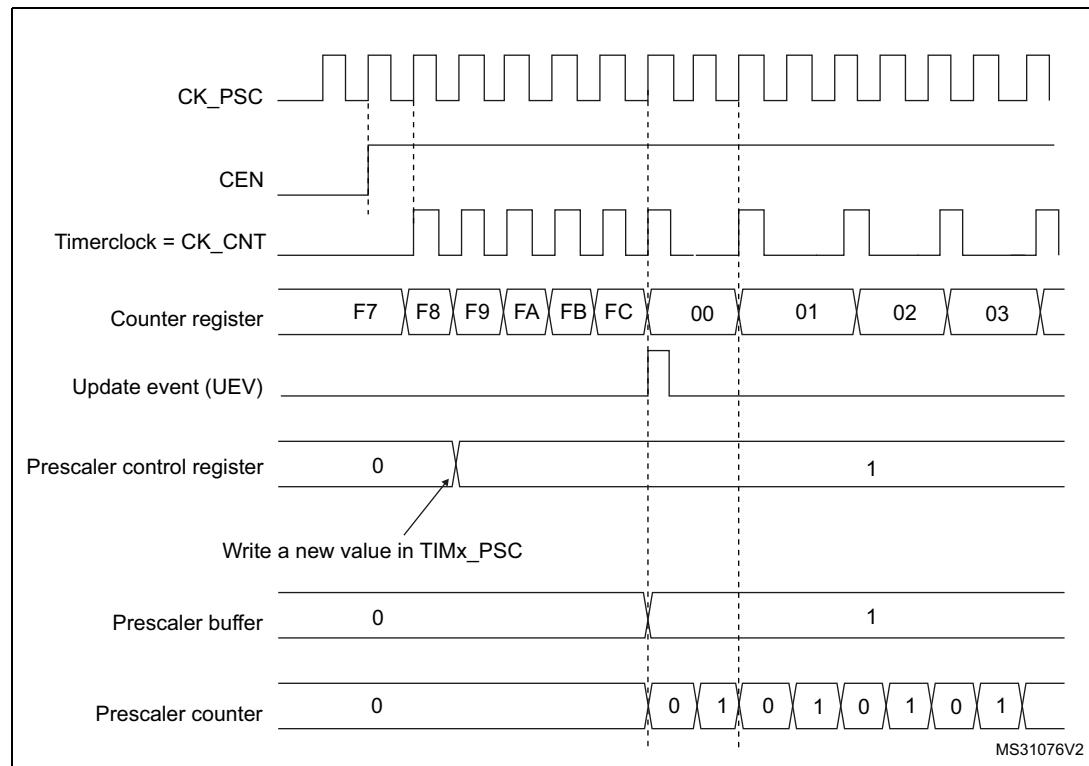
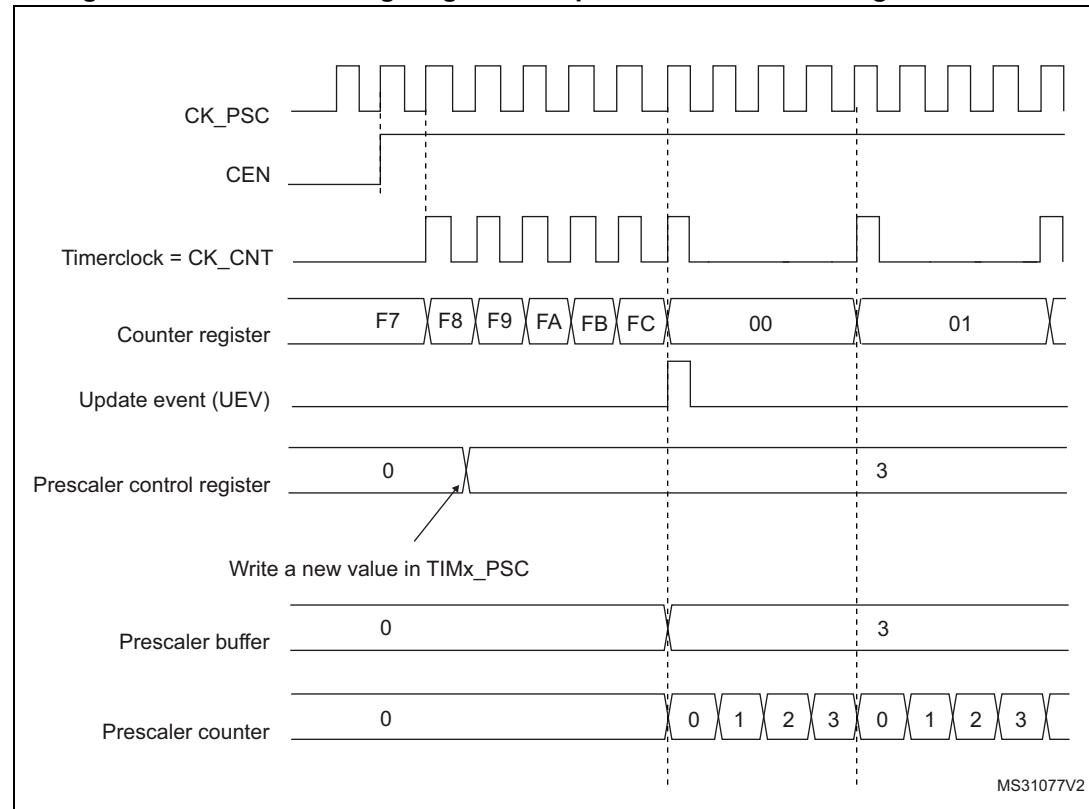
The counter is clocked by the prescaler output CK_CNT, which is enabled only when the counter enable bit (CEN) in the TIMx_CR1 register is set.

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

Prescaler description

The prescaler can divide the counter clock frequency by any factor between 1 and 65536. It is based on a 16-bit counter controlled through a 16-bit register (in the TIMx_PSC register). It can be changed on the fly as the TIMx_PSC control register is buffered. The new prescaler ratio is taken into account at the next update event.

Figure 267 and *Figure 268* give some examples of the counter behavior when the prescaler ratio is changed on the fly.

Figure 267. Counter timing diagram with prescaler division change from 1 to 2**Figure 268. Counter timing diagram with prescaler division change from 1 to 4**

21.3.2 Counting mode

The counter counts from 0 to the auto-reload value (contents of the TIMx_ARR register), then restarts from 0 and generates a counter overflow event.

An update event can be generated at each counter overflow or by setting the UG bit in the TIMx_EGR register (by software or by using the slave mode controller).

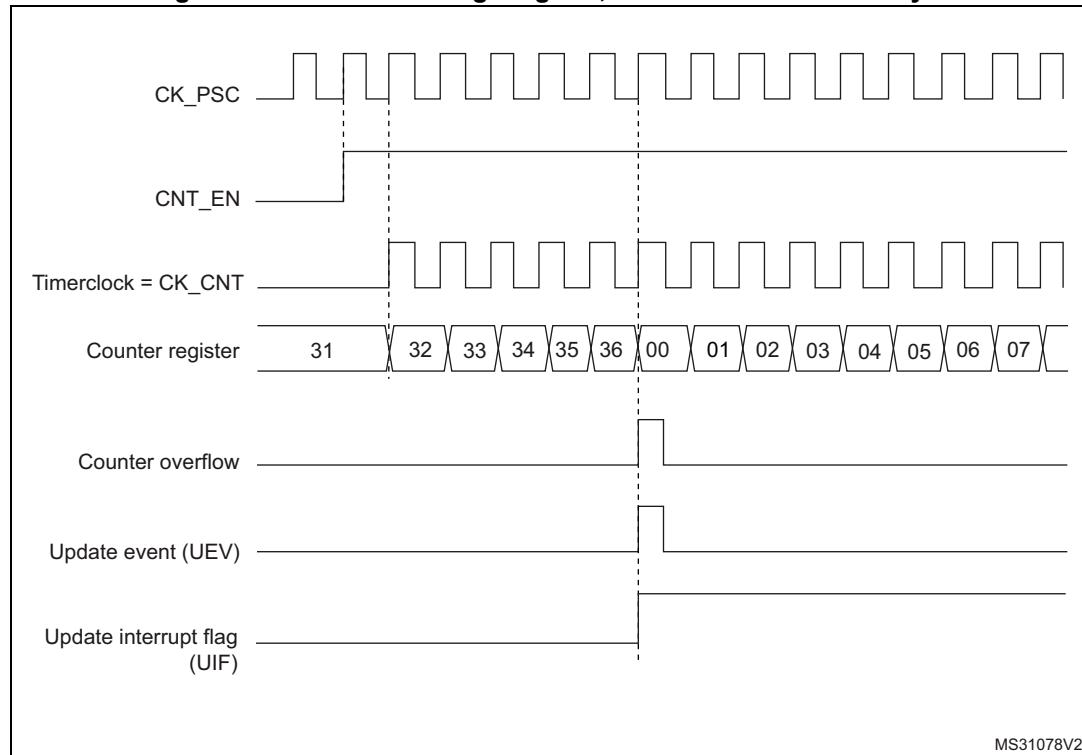
The UEV event can be disabled by software by setting the UDIS bit in the TIMx_CR1 register. This avoids updating the shadow registers while writing new values into the preload registers. In this way, no update event occurs until the UDIS bit has been written to 0, however, the counter and the prescaler counter both restart from 0 (but the prescale rate does not change). In addition, if the URS (update request selection) bit in the TIMx_CR1 register is set, setting the UG bit generates an update event UEV, but the UIF flag is not set (so no interrupt or DMA request is sent).

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

- The buffer of the prescaler is reloaded with the preload value (contents of the TIMx_PSC register)
- The auto-reload shadow register is updated with the preload value (TIMx_ARR)

The following figures show some examples of the counter behavior for different clock frequencies when TIMx_ARR = 0x36.

Figure 269. Counter timing diagram, internal clock divided by 1



MS31078V2

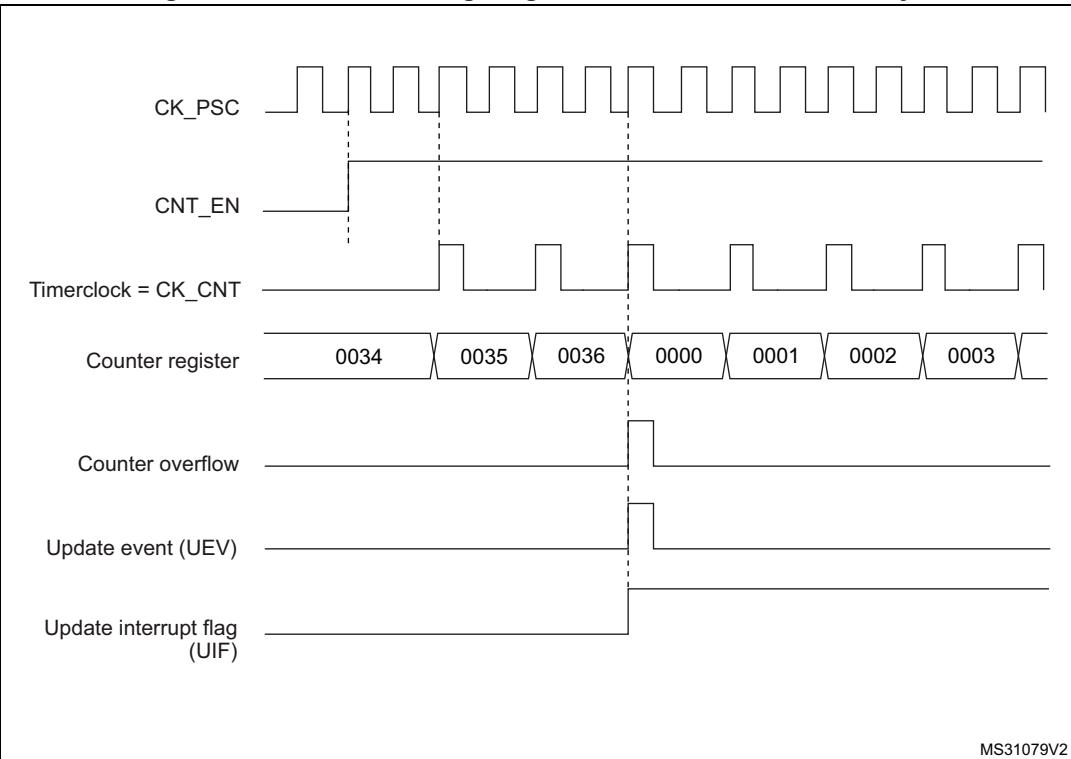
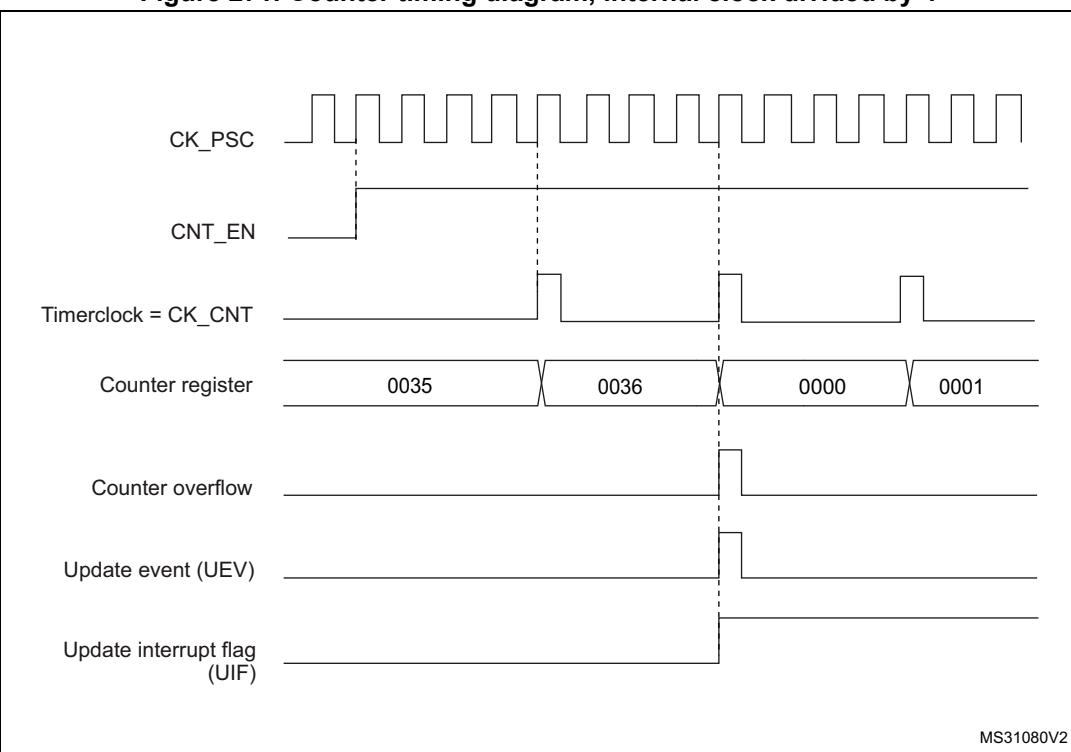
Figure 270. Counter timing diagram, internal clock divided by 2**Figure 271. Counter timing diagram, internal clock divided by 4**

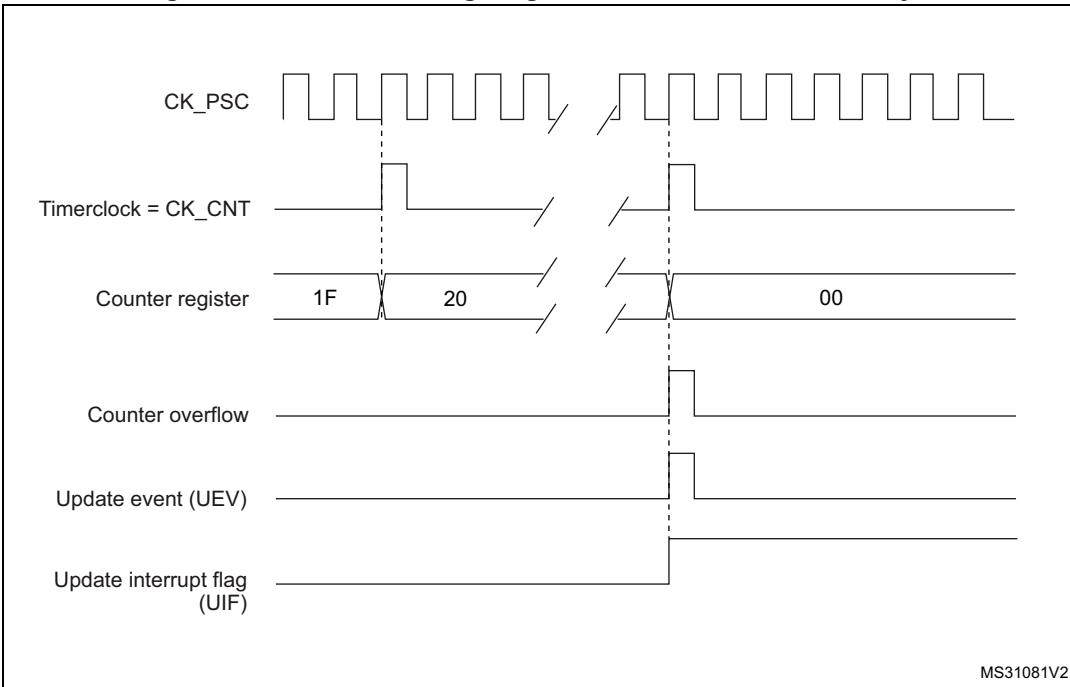
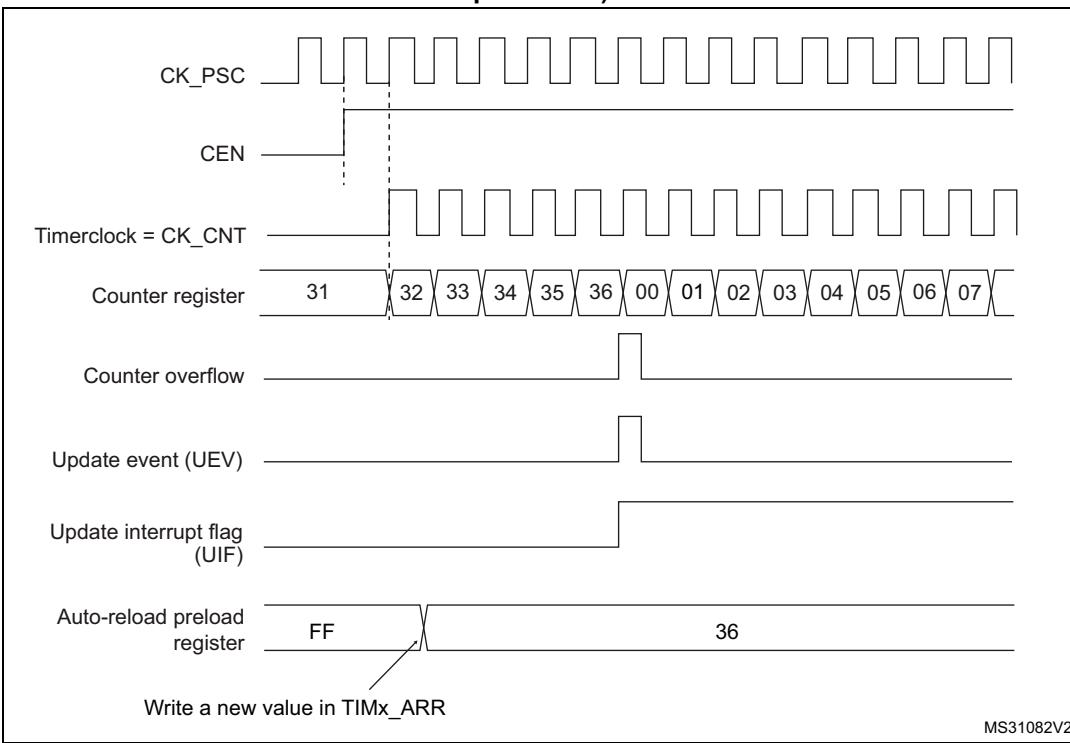
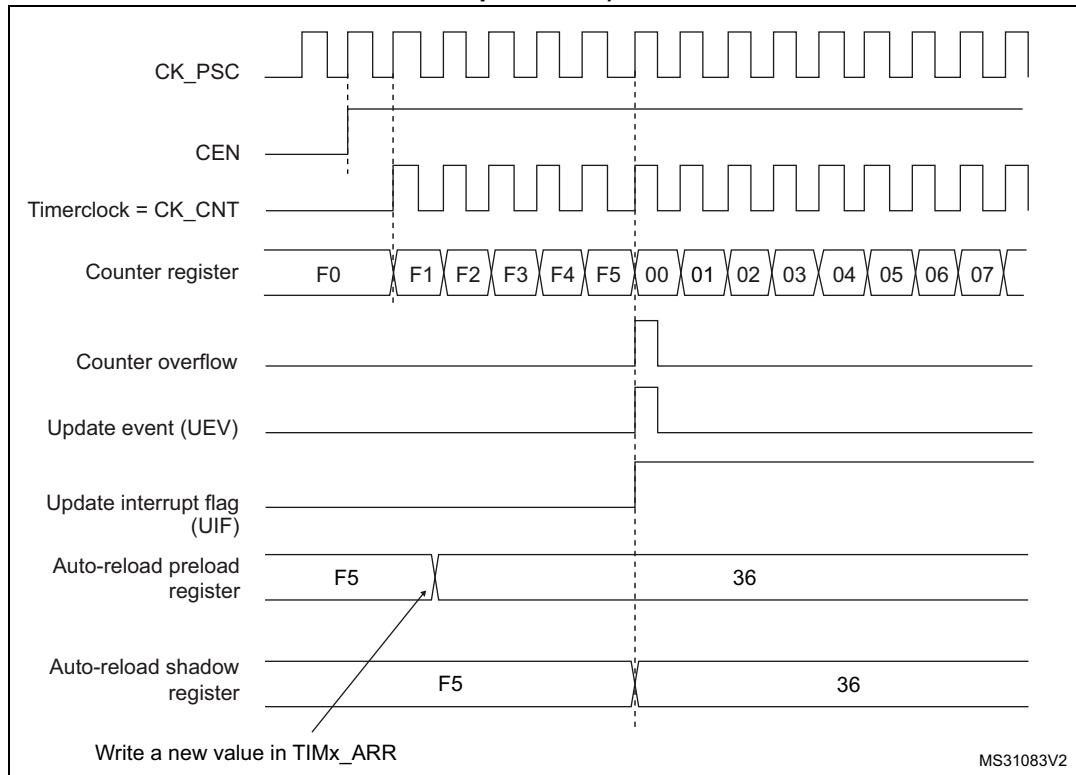
Figure 272. Counter timing diagram, internal clock divided by N**Figure 273. Counter timing diagram, update event when ARPE = 0 (TIMx_ARR not preloaded)**

Figure 274. Counter timing diagram, update event when ARPE=1 (TIMx_ARR preloaded)



21.3.3 UIF bit remapping

The IUFREMAP bit in the TIMx_CR1 register forces a continuous copy of the Update Interrupt Flag UIF into the timer counter register's bit 31 (TIMxCNT[31]). This allows to atomically read both the counter value and a potential roll-over condition signaled by the UIFCPY flag. In particular cases, it can ease the calculations by avoiding race conditions caused for instance by a processing shared between a background task (counter reading) and an interrupt (Update Interrupt).

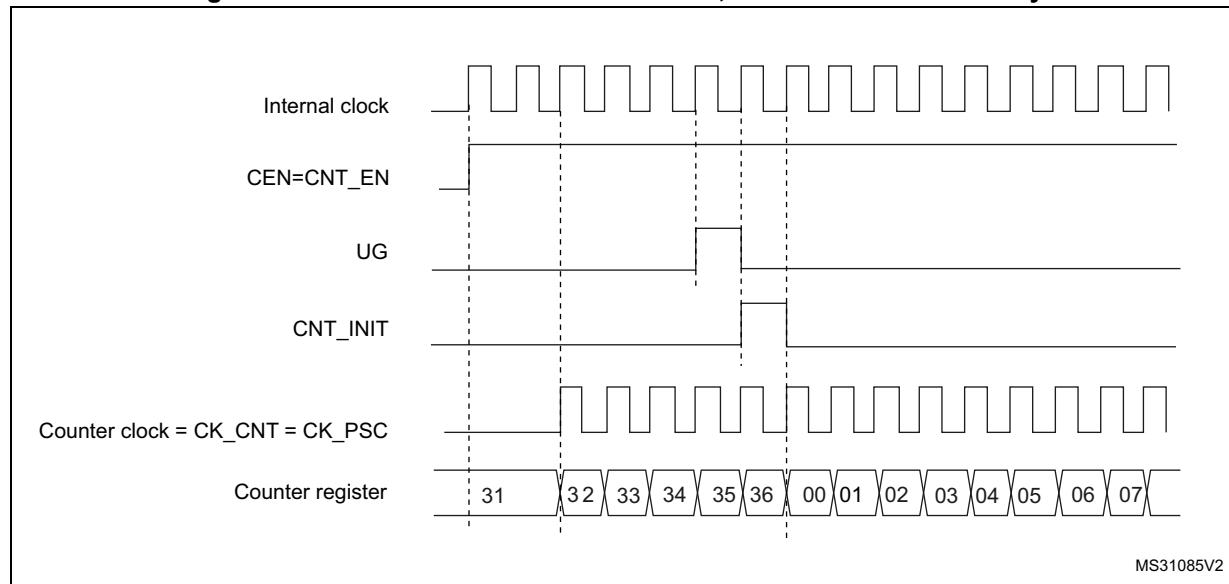
There is no latency between the assertions of the UIF and UIFCPY flags.

21.3.4 Clock source

The counter clock is provided by the Internal clock (CK_INT) source.

The CEN (in the TIMx_CR1 register) and UG bits (in the TIMx_EGR register) are actual control bits and can be changed only by software (except for UG that remains cleared automatically). As soon as the CEN bit is written to 1, the prescaler is clocked by the internal clock CK_INT.

Figure 275 shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

Figure 275. Control circuit in normal mode, internal clock divided by 1

21.3.5 Debug mode

When the microcontroller enters the debug mode (Cortex®-M7 core - halted), the TIMx counter either continues to work normally or stops, depending on the `DBG_TIMx_STOP` configuration bit in the DBG module. For more details, refer to [Section 40.16.2: Debug support for timers, watchdog, bxCAN and I2C](#).

21.4 TIM6/TIM7 registers

Refer to [Section 1.2 on page 53](#) for a list of abbreviations used in register descriptions.

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

21.4.1 TIM6/TIM7 control register 1 (TIMx_CR1)

Address offset: 0x000

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	UIFRE MAP	Res.	Res.	Res.	ARPE	Res.	Res.	Res.	OPM	URS	UDIS	CEN
				rw				rw				rw	rw	rw	rw

Bits 15:12 Reserved, must be kept at reset value.

Bit 11 **UIFREMAP**: UIF status bit remapping

0: No remapping. UIF status bit is not copied to TIMx_CNT register bit 31.

1: Remapping enabled. UIF status bit is copied to TIMx_CNT register bit 31.

Bits 10:8 Reserved, must be kept at reset value.

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

- 0: TIMx_ARR register is not buffered.
- 1: TIMx_ARR register is buffered.

Bits 6:4 Reserved, must be kept at reset value.

Bit 3 **OPM**: One-pulse mode

- 0: Counter is not stopped at update event
- 1: Counter stops counting at the next update event (clearing the CEN bit).

Bit 2 **URS**: Update request source

This bit is set and cleared by software to select the UEV event sources.

- 0: Any of the following events generates an update interrupt or DMA request if enabled.
These events can be:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

- 1: Only counter overflow/underflow generates an update interrupt or DMA request if enabled.

Bit 1 **UDIS**: Update disable

This bit is set and cleared by software to enable/disable UEV event generation.

- 0: UEV enabled. The Update (UEV) event is generated by one of the following events:
 - Counter overflow/underflow
 - Setting the UG bit
 - Update generation through the slave mode controllerBuffered registers are then loaded with their preload values.

- 1: UEV disabled. The Update event is not generated, shadow registers keep their value (ARR, PSC). However the counter and the prescaler are reinitialized if the UG bit is set or if a hardware reset is received from the slave mode controller.

Bit 0 **CEN**: Counter enable

- 0: Counter disabled
- 1: Counter enabled

Note: Gated mode can work only if the CEN bit has been previously set by software.

However trigger mode can set the CEN bit automatically by hardware.

CEN is cleared automatically in one-pulse mode, when an update event occurs.

21.4.2 TIM6/TIM7 control register 2 (TIMx_CR2)

Address offset: 0x04

Reset value: 0x0000

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

Bits 15:7 Reserved, must be kept at reset value.

Bits 6:4 **MMS**: Master mode selection

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

000: **Reset** - the UG bit from the TIMx_EGR register is used as a trigger output (TRGO). If reset is generated by the trigger input (slave mode controller configured in reset mode) then the signal on TRGO is delayed compared to the actual reset.

001: **Enable** - the Counter enable signal, CNT_EN, is used as a trigger output (TRGO). It is useful to start several timers at the same time or to control a window in which a slave timer is enabled. The Counter Enable signal is generated by a logic OR between CEN control bit and the trigger input when configured in gated mode.

When the Counter Enable signal is controlled by the trigger input, there is a delay on TRGO, except if the master/slave mode is selected (see the MSM bit description in the TIMx_SMCR register).

010: **Update** - The update event is selected as a trigger output (TRGO). For instance a master timer can then be used as a prescaler for a slave timer.

Note: The clock of the slave timer or ADC must be enabled prior to receive events from the master timer, and must not be changed on-the-fly while triggers are received from the master timer.

Bits 3:0 Reserved, must be kept at reset value.

21.4.3 TIM6/TIM7 DMA/Interrupt enable register (TIMx_DIER)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	UDE	Res.	UIE												
							rw								rw

Bits 15:9 Reserved, must be kept at reset value.

Bit 8 **UDE**: Update DMA request enable

- 0: Update DMA request disabled.
- 1: Update DMA request enabled.

Bits 7:1 Reserved, must be kept at reset value.

Bit 0 **UIE**: Update interrupt enable

- 0: Update interrupt disabled.
- 1: Update interrupt enabled.

21.4.4 TIM6/TIM7 status register (TIMx_SR)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	UIF														
															rc_w0

Bits 15:1 Reserved, must be kept at reset value.

Bit 0 **UIF**: Update interrupt flag

This bit is set by hardware on an update event. It is cleared by software.

0: No update occurred.

1: Update interrupt pending. This bit is set by hardware when the registers are updated:

- At overflow or underflow regarding the repetition counter value and if UDIS = 0 in the TIMx_CR1 register.
- When CNT is reinitialized by software using the UG bit in the TIMx_EGR register, if URS = 0 and UDIS = 0 in the TIMx_CR1 register.

21.4.5 TIM6/TIM7 event generation register (TIMx_EGR)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	UG														
															w

Bits 15:1 Reserved, must be kept at reset value.

Bit 0 **UG**: Update generation

This bit can be set by software, it is automatically cleared by hardware.

0: No action.

1: Re-initializes the timer counter and generates an update of the registers. Note that the prescaler counter is cleared too (but the prescaler ratio is not affected).

21.4.6 TIM6/TIM7 counter (TIMx_CNT)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UIF CPY	Res.														
r															
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **UIFCPY**: UIF Copy

This bit is a read-only copy of the UIF bit of the TIMx_ISR register. If the UIFREMAP bit in TIMx_CR1 is reset, bit 31 is reserved and read as 0.

Bits 30:16 Reserved, must be kept at reset value.

Bits 15:0 **CNT[15:0]**: Counter value

21.4.7 TIM6/TIM7 prescaler (TIMx_PSC)

Address offset: 0x28

Reset value: 0x0000

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

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK_CNT is equal to $f_{CK_PSC} / (PSC[15:0] + 1)$.

PSC contains the value to be loaded into the active prescaler register at each update event.

(including when the counter is cleared through UG bit of TIMx_EGR register or through trigger controller when configured in “reset mode”).

21.4.8 TIM6/TIM7 auto-reload register (TIMx_ARR)

Address offset: 0x2C

Reset value: 0xFFFF

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

Bits 15:0 **ARR[15:0]**: Prescaler value

ARR is the value to be loaded into the actual auto-reload register.

Refer to [Section 21.3.1: Time-base unit on page 725](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

21.4.9 TIM6/TIM7 register map

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

Table 127. TIM6/TIM7 register map and reset values

Offset	Register name	Reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	TIMx_CR1		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value																																	
0x04	TIMx_CR2		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																	
0x08																																		
0x0C	TIMx_DIER		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																	
0x10	TIMx_SR		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																	
0x14	TIMx_EGR		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																	
0x18-0x20																																		
0x24	TIMx_CNT		UFCPY or Res.																															
	Reset value	0																																
0x28	TIMx_PSC		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value																																	
0x2C	TIMx_ARR		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value																																	

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

22 Low-power timer (LPTIM)

22.1 Introduction

The LPTIM is a 16-bit timer that benefits from the ultimate developments in power consumption reduction. Thanks to its diversity of clock sources, the LPTIM is able to keep running in all power modes except for Standby mode. Given its capability to run even with no internal clock source, the LPTIM can be used as a “Pulse Counter” which can be useful in some applications. Also, the LPTIM capability to wake up the system from low-power modes, makes it suitable to realize “Timeout functions” with extremely low power consumption.

The LPTIM introduces a flexible clock scheme that provides the needed functionalities and performance, while minimizing the power consumption.

22.2 LPTIM main features

- 16 bit upcounter
- 3-bit prescaler with 8 possible dividing factors (1,2,4,8,16,32,64,128)
- Selectable clock
 - Internal clock sources: LSE, LSI, HSI or APB clock
 - External clock source over LPTIM input (working with no LP oscillator running, used by Pulse Counter application)
- 16 bit ARR autoreload register
- 16 bit compare register
- Continuous/One-shot mode
- Selectable software/hardware input trigger
- Programmable Digital Glitch filter
- Configurable output: Pulse, PWM
- Configurable I/O polarity
- Encoder mode

22.3 LPTIM implementation

Table 128 describes LPTIM implementation on STM32F72xxx and STM32F73xxx devices: the full set of features is implemented in LPTIM1.

Table 128. STM32F72xxx and STM32F73xxx LPTIM features

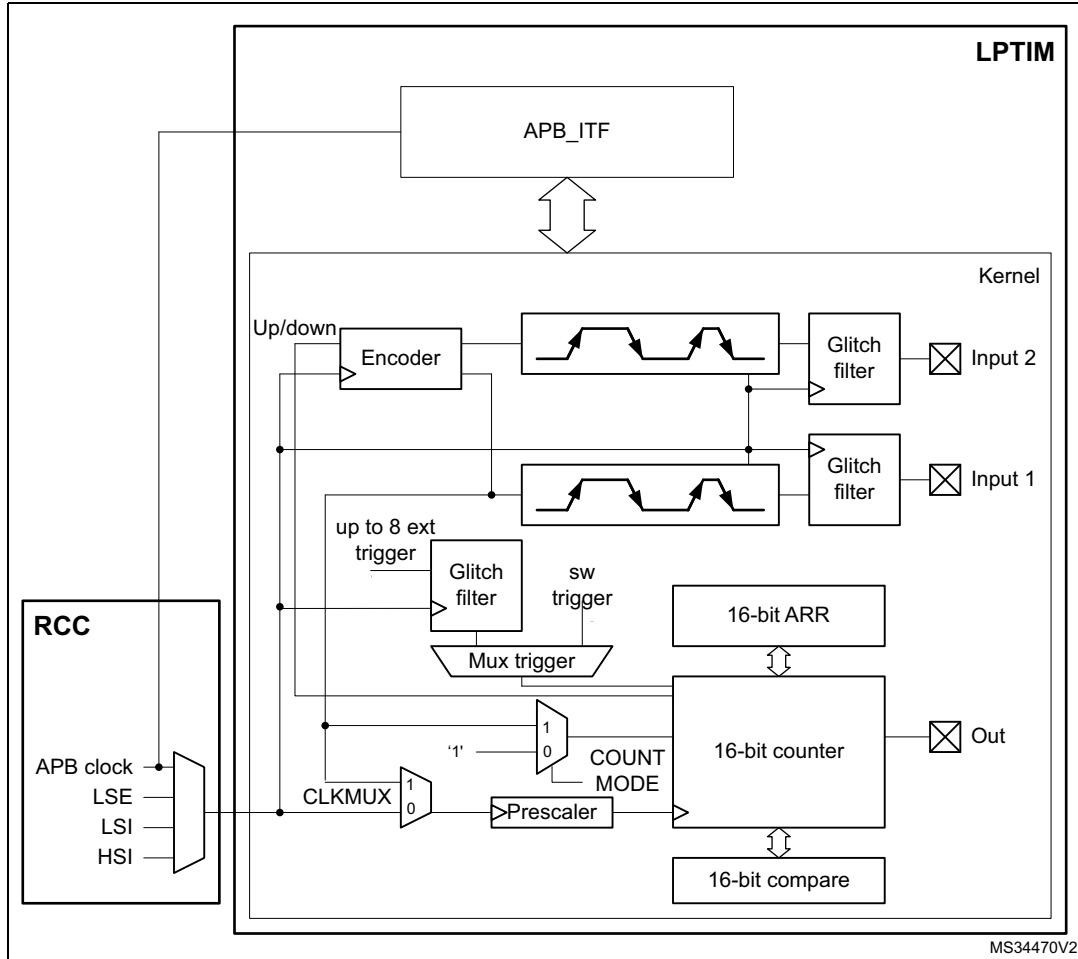
LPTIM modes/features ⁽¹⁾	LPTIM1
Encoder mode	X

1. X = supported.

22.4 LPTIM functional description

22.4.1 LPTIM block diagram

Figure 276. Low-power timer block diagram



22.4.2 LPTIM trigger mapping

The LPTIM external trigger connections are detailed hereafter:

Table 129. LPTIM1 external trigger connection

TRIGSEL	External trigger
lptim_ext_trig0	GPIO
lptim_ext_trig1	RTC_ALARMA
lptim_ext_trig2	RTC_ALARMB
lptim_ext_trig3	RTC_TAMP1_OUT
lptim_ext_trig4	RTC_TAMP2_OUT
lptim_ext_trig5	RTC_TAMP3_OUT

Table 129. LPTIM1 external trigger connection (continued)

TRIGSEL	External trigger
lptim_ext_trig6	Reserved
lptim_ext_trig7	Reserved

22.4.3 LPTIM reset and clocks

The LPTIM can be clocked using several clock sources. It can be clocked using an internal clock signal which can be chosen among APB, LSI, LSE or HSI sources through the Reset and Clock controller (RCC). Also, the LPTIM can be clocked using an external clock signal injected on its external Input1. When clocked with an external clock source, the LPTIM may run in one of these two possible configurations:

- The first configuration is when the LPTIM is clocked by an external signal but in the same time an internal clock signal is provided to the LPTIM either from APB or any other embedded oscillator including LSE, LSI and HSI.
- The second configuration is when the LPTIM is solely clocked by an external clock source through its external Input1. This configuration is the one used to realize Timeout function or Pulse counter function when all the embedded oscillators are turned off after entering a low-power mode.

Programming the CKSEL and COUNTMODE bits allows controlling whether the LPTIM will use an external clock source or an internal one.

When configured to use an external clock source, the CKPOL bits are used to select the external clock signal active edge. If both edges are configured to be active ones, an internal clock signal should also be provided (first configuration). In this case, the internal clock signal frequency should be at least four times higher than the external clock signal frequency.

22.4.4 Glitch filter

The LPTIM inputs, either external (mapped to microcontroller GPIOs) or internal (mapped on the chip-level to other embedded peripherals, such as embedded comparators), are protected with digital filters that prevent any glitches and noise perturbations to propagate inside the LPTIM. This is in order to prevent spurious counts or triggers.

Before activating the digital filters, an internal clock source should first be provided to the LPTIM. This is necessary to guarantee the proper operation of the filters.

The digital filters are divided into two groups:

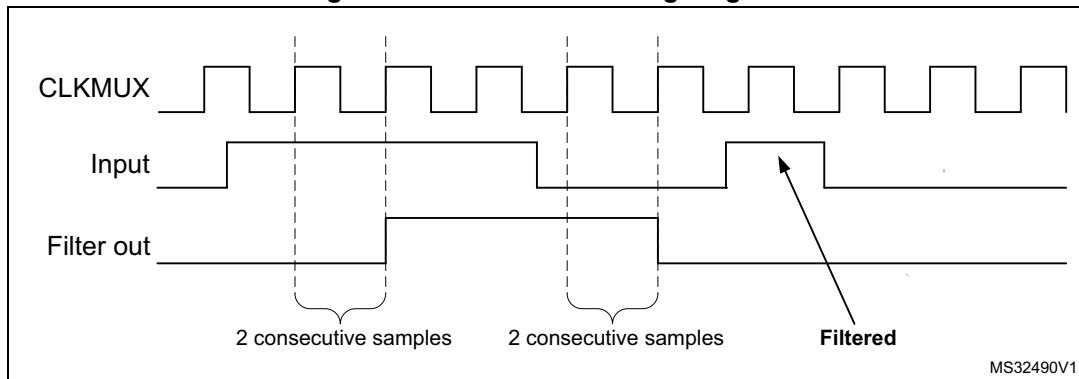
- The first group of digital filters protects the LPTIM external inputs. The digital filters sensitivity is controlled by the CKFLT bits
- The second group of digital filters protects the LPTIM internal trigger inputs. The digital filters sensitivity is controlled by the TRGFLT bits.

Note: *The digital filters sensitivity is controlled by groups. It is not possible to configure each digital filter sensitivity separately inside the same group.*

The filter sensitivity acts on the number of consecutive equal samples that should be detected on one of the LPTIM inputs to consider a signal level change as a valid transition.

Figure 277 shows an example of glitch filter behavior in case of a 2 consecutive samples programmed.

Figure 277. Glitch filter timing diagram



Note: In case no internal clock signal is provided, the digital filter must be deactivated by setting the CKFLT and TRGFLT bits to '0'. In that case, an external analog filter may be used to protect the LPTIM external inputs against glitches.

22.4.5 Prescaler

The LPTIM 16-bit counter is preceded by a configurable power-of-2 prescaler. The prescaler division ratio is controlled by the PRESC[2:0] 3-bit field. The table below lists all the possible division ratios:

Table 130. Prescaler division ratios

programming	dividing factor
000	/1
001	/2
010	/4
011	/8
100	/16
101	/32
110	/64
111	/128

22.4.6 Trigger multiplexer

The LPTIM counter may be started either by software or after the detection of an active edge on one of the 8 trigger inputs.

TRIGEN[1:0] is used to determine the LPTIM trigger source:

- When TRIGEN[1:0] equals '00', The LPTIM counter is started as soon as one of the CNTSTRT or the SNGSTRT bits is set by software. The three remaining possible values for the TRIGEN[1:0] are used to configure the active edge used by the trigger inputs. The LPTIM counter starts as soon as an active edge is detected.
- When TRIGEN[1:0] is different than '00', TRIGSEL[2:0] is used to select which of the 8 trigger inputs is used to start the counter.

The external triggers are considered asynchronous signals for the LPTIM. So after a trigger detection, a two-counter-clock period latency is needed before the timer starts running due to the synchronization.

If a new trigger event occurs when the timer is already started it will be ignored (unless timeout function is enabled).

Note: *The timer must be enabled before setting the SNGSTRT/CNTSTRT bits. Any write on these bits when the timer is disabled will be discarded by hardware.*

22.4.7 Operating mode

The LPTIM features two operating modes:

- The Continuous mode: the timer is free running, the timer is started from a trigger event and never stops until the timer is disabled
- One-shot mode: the timer is started from a trigger event and stops when reaching the ARR value.

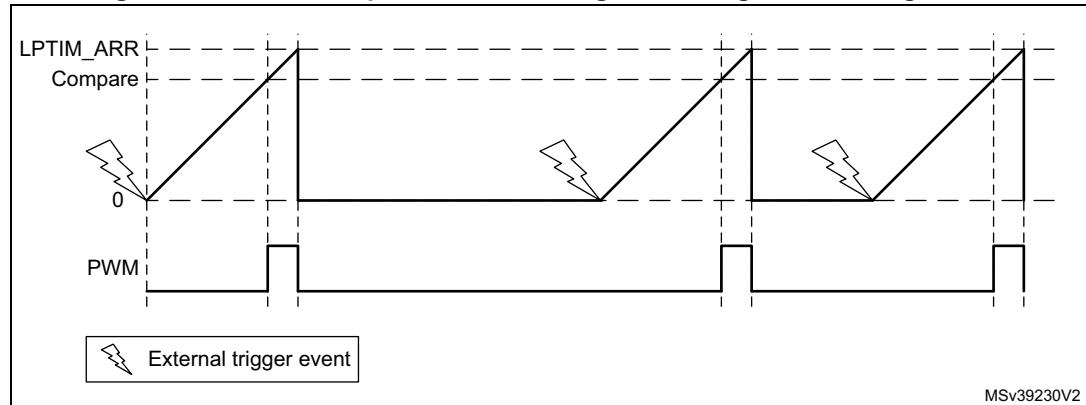
One-shot mode

To enable the one-shot counting, the SNGSTRT bit must be set.

A new trigger event will re-start the timer. Any trigger event occurring after the counter starts and before the counter reaches ARR will be discarded.

In case an external trigger is selected, each external trigger event arriving after the SNGSTRT bit is set, and after the counter register has stopped (contains zero value), will start the counter for a new one-shot counting cycle as shown in [Figure 278](#).

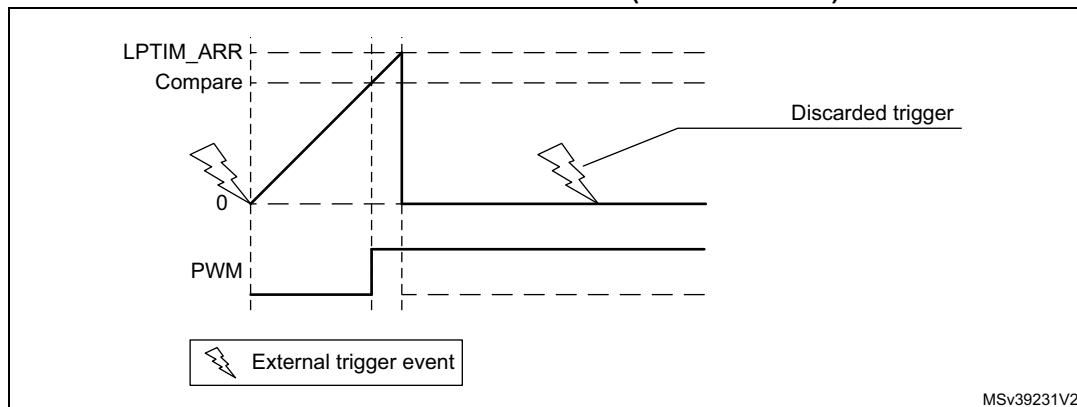
Figure 278. LPTIM output waveform, single counting mode configuration



- Set-once mode activated:

It should be noted that when the WAVE bit-field in the LPTIM_CFGR register is set, the Set-once mode is activated. In this case, the counter is only started once following the first trigger, and any subsequent trigger event is discarded as shown in [Figure 279](#).

Figure 279. LPTIM output waveform, Single counting mode configuration and Set-once mode activated (WAVE bit is set)



In case of software start (TRIGEN[1:0] = '00'), the SNGSTRT setting will start the counter for one-shot counting.

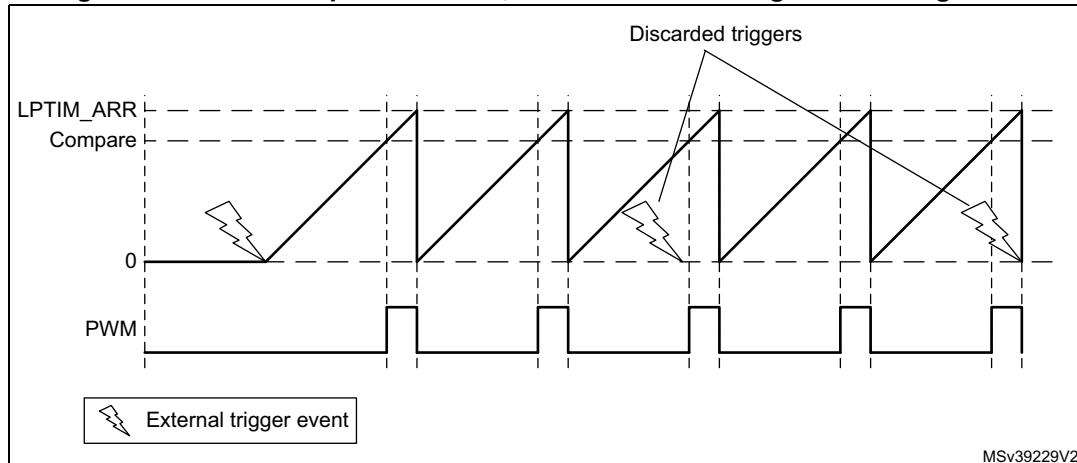
Continuous mode

To enable the continuous counting, the CNTSTRT bit must be set.

In case an external trigger is selected, an external trigger event arriving after CNTSTRT is set will start the counter for continuous counting. Any subsequent external trigger event will be discarded as shown in [Figure 280](#).

In case of software start (TRIGEN[1:0] = '00'), setting CNTSTRT will start the counter for continuous counting.

Figure 280. LPTIM output waveform, Continuous counting mode configuration



SNGSTRT and CNTSTRT bits can only be set when the timer is enabled (The ENABLE bit is set to '1'). It is possible to change "on the fly" from One-shot mode to Continuous mode.

If the Continuous mode was previously selected, setting SNGSTRT will switch the LPTIM to the One-shot mode. The counter (if active) will stop as soon as it reaches ARR.

If the One-shot mode was previously selected, setting CNTSTRT will switch the LPTIM to the Continuous mode. The counter (if active) will restart as soon as it reaches ARR.

22.4.8 Timeout function

The detection of an active edge on one selected trigger input can be used to reset the LPTIM counter. This feature is controlled through the TIMOUT bit.

The first trigger event will start the timer, any successive trigger event will reset the counter and the timer will restart.

The first trigger event will start the timer, any successive trigger event will reset the counter and the timer will restart.

A low-power timeout function can be realized. The timeout value corresponds to the compare value; if no trigger occurs within the expected time frame, the MCU is waked-up by the compare match event.

22.4.9 Waveform generation

Two 16-bit registers, the LPTIM_ARR (autoreload register) and LPTIM_CMP (compare register), are used to generate several different waveforms on LPTIM output

The timer can generate the following waveforms:

- The PWM mode: the LPTIM output is set as soon as the counter value in LPTIM_CNT exceeds the compare value in LPTIM_CMP. The LPTIM output is reset as soon as a match occurs between the LPTIM_ARR and the LPTIM_CNT registers.
- The One-pulse mode: the output waveform is similar to the one of the PWM mode for the first pulse, then the output is permanently reset
- The Set-once mode: the output waveform is similar to the One-pulse mode except that the output is kept to the last signal level (depends on the output configured polarity).

The above described modes require that the LPTIM_ARR register value be strictly greater than the LPTIM_CMP register value.

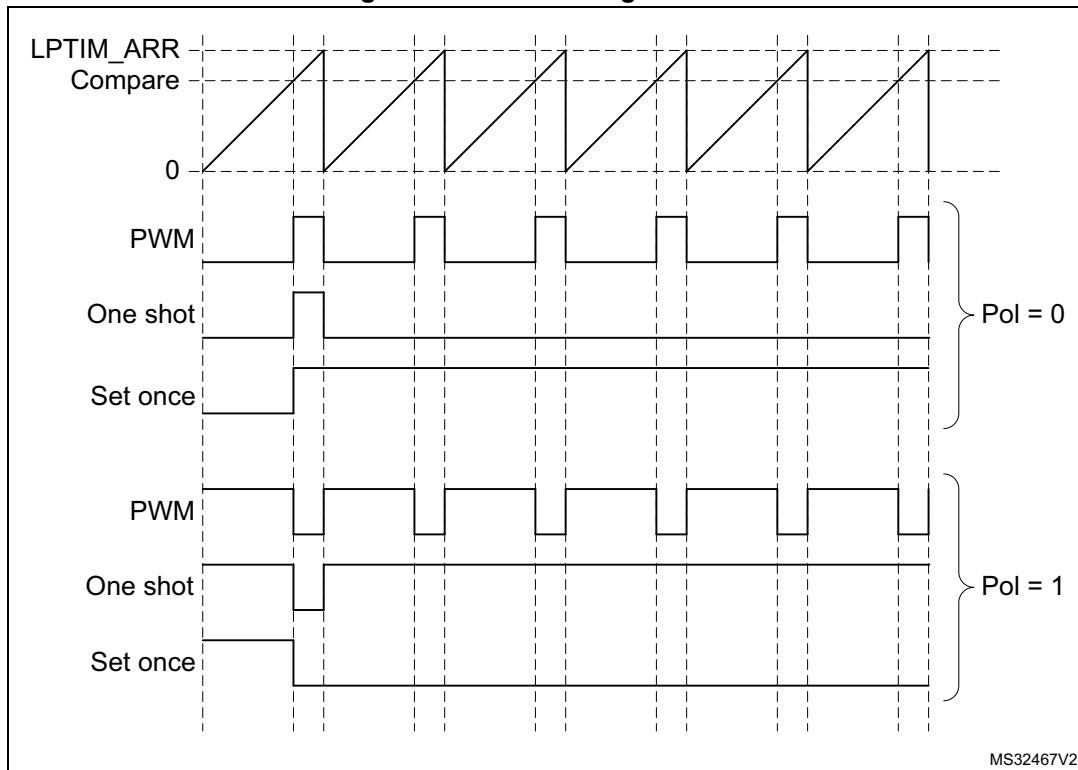
The LPTIM output waveform can be configured through the WAVE bit as follow:

- Resetting the WAVE bit to '0' forces the LPTIM to generate either a PWM waveform or a One pulse waveform depending on which bit is set: CNTSTART or SNGSTART.
- Setting the WAVE bit to '1' forces the LPTIM to generate a Set-once mode waveform.

The WAVPOL bit controls the LPTIM output polarity. The change takes effect immediately, so the output default value will change immediately after the polarity is re-configured, even before the timer is enabled.

Signals with frequencies up to the LPTIM clock frequency divided by 2 can be generated. [Figure 281](#) below shows the three possible waveforms that can be generated on the LPTIM output. Also, it shows the effect of the polarity change using the WAVPOL bit.

Figure 281. Waveform generation



22.4.10 Register update

The LPTIM_ARR register and LPTIM_CMP register are updated immediately after the APB bus write operation, or at the end of the current period if the timer is already started.

The PRELOAD bit controls how the LPTIM_ARR and the LPTIM_CMP registers are updated:

- When the PRELOAD bit is reset to ‘0’, the LPTIM_ARR and the LPTIM_CMP registers are immediately updated after any write access.
- When the PRELOAD bit is set to ‘1’, the LPTIM_ARR and the LPTIM_CMP registers are updated at the end of the current period, if the timer has been already started.

The LPTIM APB interface and the LPTIM kernel logic use different clocks, so there is some latency between the APB write and the moment when these values are available to the counter comparator. Within this latency period, any additional write into these registers must be avoided.

The ARROK flag and the CMPOK flag in the LPTIM_ISR register indicate when the write operation is completed to respectively the LPTIM_ARR register and the LPTIM_CMP register.

After a write to the LPTIM_ARR register or the LPTIM_CMP register, a new write operation to the same register can only be performed when the previous write operation is completed. Any successive write before respectively the ARROK flag or the CMPOK flag be set, will lead to unpredictable results.

22.4.11 Counter mode

The LPTIM counter can be used to count external events on the LPTIM Input1 or it can be used to count internal clock cycles. The CKSEL and COUNTMODE bits control which source will be used for updating the counter.

In case the LPTIM is configured to count external events on Input1, the counter can be updated following a rising edge, falling edge or both edges depending on the value written to the CKPOL[1:0] bits.

The count modes below can be selected, depending on CKSEL and COUNTMODE values:

- CKSEL = 0: the LPTIM is clocked by an internal clock source
 - COUNTMODE = 0
The LPTIM is configured to be clocked by an internal clock source and the LPTIM counter is configured to be updated following each internal clock pulse.
 - COUNTMODE = 1
The LPTIM external Input1 is sampled with the internal clock provided to the LPTIM.

Consequently, in order not to miss any event, the frequency of the changes on the external Input1 signal should never exceed the frequency of the internal clock provided to the LPTIM. Also, the internal clock provided to the LPTIM must not be prescaled (PRESC[2:0] = 000).

- CKSEL = 1: the LPTIM is clocked by an external clock source
COUNTMODE value is don't care.

In this configuration, the LPTIM has no need for an internal clock source (except if the glitch filters are enabled). The signal injected on the LPTIM external Input1 is used as system clock for the LPTIM. This configuration is suitable for operation modes where no embedded oscillator is enabled.

For this configuration, the LPTIM counter can be updated either on rising edges or falling edges of the input1 clock signal but not on both rising and falling edges.

Since the signal injected on the LPTIM external Input1 is also used to clock the LPTIM kernel logic, there is some initial latency (after the LPTIM is enabled) before the counter is incremented. More precisely, the first five active edges on the LPTIM external Input1 (after LPTIM is enable) are lost.

22.4.12 Timer enable

The ENABLE bit located in the LPTIM_CR register is used to enable/disable the LPTIM kernel logic. After setting the ENABLE bit, a delay of two counter clock is needed before the LPTIM is actually enabled.

The LPTIM_CFGR and LPTIM_IER registers must be modified only when the LPTIM is disabled.

22.4.13 Encoder mode

This mode allows handling signals from quadrature encoders used to detect angular position of rotary elements. 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 programmed into the LPTIM_ARR register (0 up to ARR or ARR down to 0 depending on the direction). Therefore you must configure LPTIM_ARR before starting. From the two external input signals, Input1 and Input2, a clock signal is generated to clock the LPTIM counter. The phase between those two signals determines the counting direction.

The Encoder mode is only available when the LPTIM is clocked by an internal clock source. The signals frequency on both Input1 and Input2 inputs must not exceed the LPTIM internal clock frequency divided by 4. This is mandatory in order to guarantee a proper operation of the LPTIM.

Direction change is signalized by the two Down and Up flags in the LPTIM_ISR register. Also, an interrupt can be generated for both direction change events if enabled through the DOWNIE bit.

To activate the Encoder mode the ENC bit has to be set to '1'. The LPTIM must first be configured in Continuous mode.

When Encoder mode is active, the LPTIM counter is modified automatically following the speed and the direction of the incremental encoder. Therefore, its content always represents the encoder's position. The count direction, signaled by the Up and Down flags, correspond to the rotation direction of the encoder rotor.

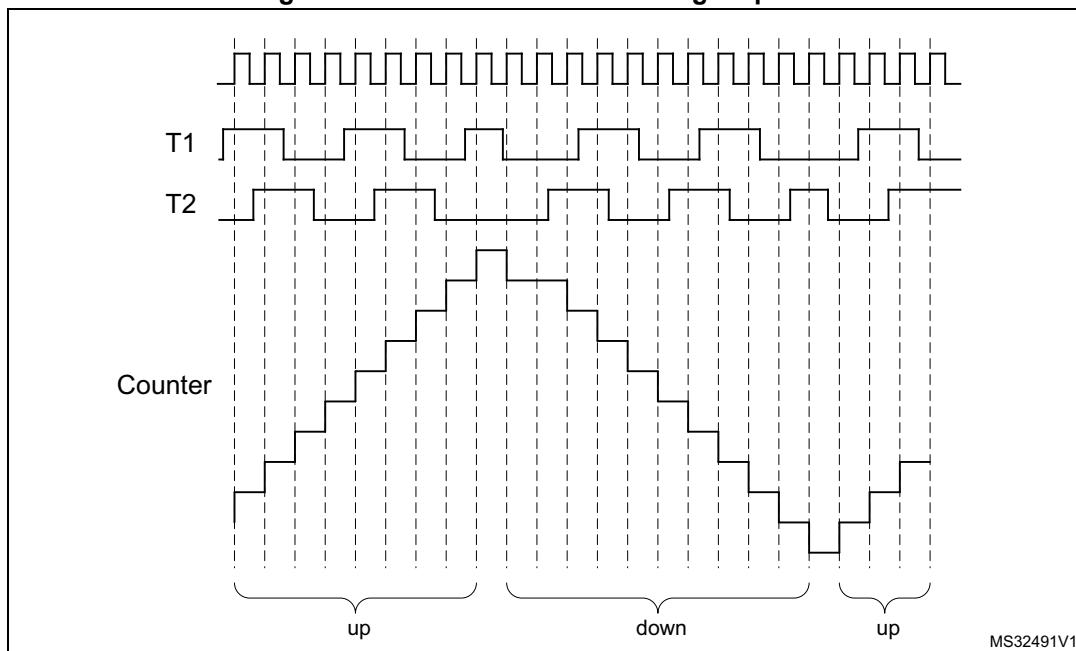
According to the edge sensitivity configured using the CKPOL[1:0] bits, different counting scenarios are possible. The following table summarizes the possible combinations, assuming that Input1 and Input2 do not switch at the same time.

Table 131. Encoder counting scenarios

Active edge	Level on opposite signal (Input1 for Input2, Input2 for Input1)	Input1 signal		Input2 signal	
		Rising	Falling	Rising	Falling
Rising Edge	High	Down	No count	Up	No count
	Low	Up	No count	Down	No count
Falling Edge	High	No count	Up	No count	Down
	Low	No count	Down	No count	Up
Both Edges	High	Down	Up	Up	Down
	Low	Up	Down	Down	Up

The following figure shows a counting sequence for Encoder mode where both-edge sensitivity is configured.

Caution: In this mode the LPTIM must be clocked by an internal clock source, so the CKSEL bit must be maintained to its reset value which is equal to '0'. Also, the prescaler division ratio must be equal to its reset value which is 1 (PRESC[2:0] bits must be '000').

Figure 282. Encoder mode counting sequence

22.4.14 Debug mode

When the microcontroller enters debug mode (core halted), the LPTIM counter either continues to work normally or stops, depending on the `DBG_LPTIM_STOP` configuration bit in the DBG module.

22.5 LPTIM low-power modes

Table 132. Effect of low-power modes on the LPTIM

Mode	Description
Sleep	No effect. LPTIM interrupts cause the device to exit Sleep mode.
Stop	The LPTIM peripheral is active when it is clocked by LSE or LSI. LPTIM interrupts cause the device to exit Stop mode
Standby	The LPTIM peripheral is powered down and must be reinitialized after exiting Standby mode.

22.6 LPTIM interrupts

The following events generate an interrupt/wake-up event, if they are enabled through the LPTIM_IER register:

- Compare match
- Auto-reload match (whatever the direction if encoder mode)
- External trigger event
- Autoreload register write completed
- Compare register write completed
- Direction change (encoder mode), programmable (up / down / both).

Note: *If any bit in the LPTIM_IER register (Interrupt Enable Register) is set after that its corresponding flag in the LPTIM_ISR register (Status Register) is set, the interrupt is not asserted.*

Table 133. Interrupt events

Interrupt event	Description
Compare match	Interrupt flag is raised when the content of the Counter register (LPTIM_CNT) matches the content of the compare register (LPTIM_CMP).
Auto-reload match	Interrupt flag is raised when the content of the Counter register (LPTIM_CNT) matches the content of the Auto-reload register (LPTIM_ARR).
External trigger event	Interrupt flag is raised when an external trigger event is detected
Auto-reload register update OK	Interrupt flag is raised when the write operation to the LPTIM_ARR register is complete.
Compare register update OK	Interrupt flag is raised when the write operation to the LPTIM_CMP register is complete.
Direction change	Used in Encoder mode. Two interrupt flags are embedded to signal direction change: – UP flag signals up-counting direction change – DOWN flag signals down-counting direction change.

22.7 LPTIM registers

22.7.1 LPTIM interrupt and status register (LPTIM_ISR)

Address offset: 0x000

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	DOWN	UP	ARROK	CMPOK	EXTTRIG	ARRM	CMPM								
								r	r	r	r	r	r	r	r

Bits 31:22 Reserved, must be kept at reset value.

Bit 21 Reserved, must be kept at reset value.

Bit 20 Reserved, must be kept at reset value.

Bit 19 Reserved, must be kept at reset value.

Bits 18:16 Reserved, must be kept at reset value.

Bit 15 Reserved, must be kept at reset value.

Bit 14 Reserved, must be kept at reset value.

Bit 13 Reserved, must be kept at reset value.

Bit 12 Reserved, must be kept at reset value.

Bit 11 Reserved, must be kept at reset value.

Bit 10 Reserved, must be kept at reset value.

Bit 9 Reserved, must be kept at reset value.

Bits 8:7 Reserved, must be kept at reset value.

Bit 6 DOWN: Counter direction change up to down

In Encoder mode, DOWN bit is set by hardware to inform application that the counter direction has changed from up to down. DOWN flag can be cleared by writing 1 to the DOWNCF bit in the LPTIM_ICR register.

Bit 5 UP: Counter direction change down to up

In Encoder mode, UP bit is set by hardware to inform application that the counter direction has changed from down to up. UP flag can be cleared by writing 1 to the UPCF bit in the LPTIM_ICR register.

Bit 4 ARROK: Autoreload register update OK

ARROK is set by hardware to inform application that the APB bus write operation to the LPTIM_ARR register has been successfully completed. ARROK flag can be cleared by writing 1 to the ARROKCF bit in the LPTIM_ICR register.

Bit 3 CMPOK: Compare register update OK

CMPOK is set by hardware to inform application that the APB bus write operation to the LPTIM_CMP register has been successfully completed.

Bit 2 EXTTRIG: External trigger edge event

EXTTRIG is set by hardware to inform application that a valid edge on the selected external trigger input has occurred. If the trigger is ignored because the timer has already started, then this flag is not set. EXTTRIG flag can be cleared by writing 1 to the EXTTRIGCF bit in the LPTIM_ICR register.

Bit 1 ARRM: Autoreload match

ARRM is set by hardware to inform application that LPTIM_CNT register's value reached the LPTIM_ARR register's value. ARRM flag can be cleared by writing 1 to the ARRMCF bit in the LPTIM_ICR register.

Bit 0 CMPM: Compare match

The CMPM bit is set by hardware to inform application that LPTIM_CNT register value reached the LPTIM_CMP register's value.

22.7.2 LPTIM interrupt clear register (LPTIM_ICR)

Address offset: 0x004

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	DOWN CF	UPCF	ARRO KCF	CMPO KCF	EXTTR IGCF	ARRM CF	CMPM CF								
								w	w	w	w	w	w	w	w

Bits 31:22 Reserved, must be kept at reset value.

Bit 21 Reserved, must be kept at reset value.

Bit 20 Reserved, must be kept at reset value.

Bit 19 Reserved, must be kept at reset value.

Bits 18:16 Reserved, must be kept at reset value.

Bit 15 Reserved, must be kept at reset value.

Bit 14 Reserved, must be kept at reset value.

Bit 13 Reserved, must be kept at reset value.

Bit 12 Reserved, must be kept at reset value.

Bit 11 Reserved, must be kept at reset value.

Bit 10 Reserved, must be kept at reset value.

Bit 9 Reserved, must be kept at reset value.

Bits 8:7 Reserved, must be kept at reset value.

Bit 6 **DOWNCF**: Direction change to down clear flag

Writing 1 to this bit clear the DOWN flag in the LPTIM_ISR register.

Bit 5 **UPCF**: Direction change to UP clear flag

Writing 1 to this bit clear the UP flag in the LPTIM_ISR register.

Bit 4 **ARROKCF**: Autoreload register update OK clear flag

Writing 1 to this bit clears the ARROK flag in the LPTIM_ISR register

Bit 3 **CMPOKCF**: Compare register update OK clear flag

Writing 1 to this bit clears the CMPOK flag in the LPTIM_ISR register

Bit 2 **EXTTRIGCF**: External trigger valid edge clear flag

Writing 1 to this bit clears the EXTTRIG flag in the LPTIM_ISR register

Bit 1 **ARRMCF**: Autoreload match clear flag

Writing 1 to this bit clears the ARRM flag in the LPTIM_ISR register

Bit 0 **CMPMCF**: Compare match clear flag

Writing 1 to this bit clears the CMP flag in the LPTIM_ISR register

22.7.3 LPTIM interrupt enable register (LPTIM_IER)

Address offset: 0x008

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	DOWNIE	UPIE	ARROKIE	CMPOKIE	EXTTRIGIE	ARRMIE	CMPMIE								
									rw	rw	rw	rw	rw	rw	rw

Bits 31:28 Reserved, must be kept at reset value.

Bit 27 Reserved, must be kept at reset value.

Bit 26 Reserved, must be kept at reset value.

Bit 25 Reserved, must be kept at reset value.

Bit 24 Reserved, must be kept at reset value.

Bit 23 Reserved, must be kept at reset value.

Bit 22 Reserved, must be kept at reset value.

Bit 21 Reserved, must be kept at reset value.

Bit 20 Reserved, must be kept at reset value.

Bit 19 Reserved, must be kept at reset value.

Bits 18:17 Reserved, must be kept at reset value.

Bit 16 Reserved, must be kept at reset value.

Bit 15 Reserved, must be kept at reset value.

Bit 14 Reserved, must be kept at reset value.

Bit 13 Reserved, must be kept at reset value.

Bit 12 Reserved, must be kept at reset value.

Bit 11 Reserved, must be kept at reset value.

Bit 10 Reserved, must be kept at reset value.

Bit 9 Reserved, must be kept at reset value.

Bits 8:7 Reserved, must be kept at reset value.

Bit 6 **DOWNIE**: Direction change to down Interrupt Enable

0: DOWN interrupt disabled

1: DOWN interrupt enabled

Bit 5 **UPIE**: Direction change to UP Interrupt Enable

0: UP interrupt disabled

1: UP interrupt enabled

Bit 4 **ARROKIE**: Autoreload register update OK Interrupt Enable

- 0: ARROK interrupt disabled
- 1: ARROK interrupt enabled

Bit 3 **CMPOKIE**: Compare register update OK Interrupt Enable

- 0: CMPOK interrupt disabled
- 1: CMPOK interrupt enabled

Bit 2 **EXTTRIGIE**: External trigger valid edge Interrupt Enable

- 0: EXTTRIG interrupt disabled
- 1: EXTTRIG interrupt enabled

Bit 1 **ARRMIE**: Autoreload match Interrupt Enable

- 0: ARRM interrupt disabled
- 1: ARRM interrupt enabled

Bit 0 **CMPMIE**: Compare match Interrupt Enable

- 0: CMPM interrupt disabled
- 1: CMPM interrupt enabled

Caution: The LPTIM_IER register must only be modified when the LPTIM is disabled (ENABLE bit reset to '0')

22.7.4 LPTIM configuration register (LPTIM_CFGR)

Address offset: 0x00C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	ENC	COUNT MODE	PRELOAD	WAVPOL	WAVE	TIMOUT	TRIGEN[1:0]	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
TRIGSEL[2:0]			Res.	PRESC[2:0]			Res.	TRGFLT[1:0]		Res.	CKFLT[1:0]		CKPOL[1:0]		CKSEL
rw	rw	rw		rw	rw	rw		rw	rw		rw	rw	rw	rw	rw

Bits 31:30 Reserved, must be kept at reset value.

Bit 29 Reserved, must be kept at reset value.

Bits 28:25 Reserved, must be kept at reset value.

Bit 24 **ENC**: Encoder mode enable

The ENC bit controls the Encoder mode

- 0: Encoder mode disabled
- 1: Encoder mode enabled

Bit 23 **COUNTMODE**: counter mode enabled

The COUNTMODE bit selects which clock source is used by the LPTIM to clock the counter:

- 0: the counter is incremented following each internal clock pulse
- 1: the counter is incremented following each valid clock pulse on the LPTIM external Input1

Bit 22 PRELOAD: Registers update mode

The PRELOAD bit controls the LPTIM_ARR and the LPTIM_CMP registers update modality

- 0: Registers are updated after each APB bus write access
- 1: Registers are updated at the end of the current LPTIM period

Bit 21 WAVPOL: Waveform shape polarity

The WAVEPOL bit controls the output polarity

- 0: The LPTIM output reflects the compare results between LPTIM_ARR and LPTIM_CMP registers
- 1: The LPTIM output reflects the inverse of the compare results between LPTIM_ARR and LPTIM_CMP registers

Bit 20 WAVE: Waveform shape

The WAVE bit controls the output shape

- 0: Deactivate Set-once mode, PWM / One Pulse waveform (depending on OPMODE bit)
- 1: Activate the Set-once mode

Bit 19 TIMOUT: Timeout enable

The TIMOUT bit controls the Timeout feature

- 0: a trigger event arriving when the timer is already started will be ignored
- 1: A trigger event arriving when the timer is already started will reset and restart the counter
- 1: A trigger event arriving when the timer is already started will reset and restart the counter

Bits 18:17 TRIGEN[1:0]: Trigger enable and polarity

The TRIGEN bits controls whether the LPTIM counter is started by an external trigger or not. If the external trigger option is selected, three configurations are possible for the trigger active edge:

- 00: software trigger (counting start is initiated by software)
- 01: rising edge is the active edge
- 10: falling edge is the active edge
- 11: both edges are active edges

Bit 16 Reserved, must be kept at reset value.**Bits 15:13 TRIGSEL[2:0]:** Trigger selector

The TRIGSEL bits select the trigger source that will serve as a trigger event for the LPTIM among the below 8 available sources:

- 000: lptim_ext_trig0
- 001: lptim_ext_trig1
- 010: lptim_ext_trig2
- 011: lptim_ext_trig3
- 100: lptim_ext_trig4
- 101: lptim_ext_trig5
- 110: lptim_ext_trig6
- 111: lptim_ext_trig7

See [Section 22.4.2: LPTIM trigger mapping](#) for details.

Bit 12 Reserved, must be kept at reset value.

Bits 11:9 PRESC[2:0]: Clock prescaler

The PRESC bits configure the prescaler division factor. It can be one among the following division factors:

- 000: /1
- 001: /2
- 010: /4
- 011: /8
- 100: /16
- 101: /32
- 110: /64
- 111: /128

Bit 8 Reserved, must be kept at reset value.

Bits 7:6 TRGFLT[1:0]: Configurable digital filter for trigger

The TRGFLT value sets the number of consecutive equal samples that should be detected when a level change occurs on an internal trigger before it is considered as a valid level transition. An internal clock source must be present to use this feature

- 00: any trigger active level change is considered as a valid trigger
- 01: trigger active level change must be stable for at least 2 clock periods before it is considered as valid trigger.
- 10: trigger active level change must be stable for at least 4 clock periods before it is considered as valid trigger.
- 11: trigger active level change must be stable for at least 8 clock periods before it is considered as valid trigger.

Bit 5 Reserved, must be kept at reset value.

Bits 4:3 **CKFLT[1:0]**: Configurable digital filter for external clock

The CKFLT value sets the number of consecutive equal samples that should be detected when a level change occurs on an external clock signal before it is considered as a valid level transition. An internal clock source must be present to use this feature

00: any external clock signal level change is considered as a valid transition

01: external clock signal level change must be stable for at least 2 clock periods before it is considered as valid transition.

10: external clock signal level change must be stable for at least 4 clock periods before it is considered as valid transition.

11: external clock signal level change must be stable for at least 8 clock periods before it is considered as valid transition.

Bits 2:1 **CKPOL[1:0]**: Clock Polarity

If LPTIM is clocked by an external clock source:

When the LPTIM is clocked by an external clock source, CKPOL bits is used to configure the active edge or edges used by the counter:

00: the rising edge is the active edge used for counting

01: the falling edge is the active edge used for counting

10: both edges are active edges. When both external clock signal edges are considered active ones, the LPTIM must also be clocked by an internal clock source with a frequency equal to at least four time the external clock frequency.

11: not allowed

If the LPTIM is configured in Encoder mode (ENC bit is set):

00: the encoder sub-mode 1 is active

01: the encoder sub-mode 2 is active

10: the encoder sub-mode 3 is active

Refer to [Section 22.4.13: Encoder mode](#) for more details about Encoder mode sub-modes.

Bit 0 **CKSEL**: Clock selector

The CKSEL bit selects which clock source the LPTIM will use:

0: LPTIM is clocked by internal clock source (APB clock or any of the embedded oscillators)

1: LPTIM is clocked by an external clock source through the LPTIM external Input1

Caution: The LPTIM_CFGR register must only be modified when the LPTIM is disabled (ENABLE bit reset to '0').

22.7.5 LPTIM control register (LPTIM_CR)

Address offset: 0x010

Reset value: 0x0000 0000

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

Bits 31:3 Reserved, must be kept at reset value.

Bit 2 CNTSTRT: Timer start in Continuous mode

This bit is set by software and cleared by hardware.

In case of software start (TRIGEN[1:0] = '00'), setting this bit starts the LPTIM in Continuous mode.

If the software start is disabled (TRIGEN[1:0] different than '00'), setting this bit starts the timer in Continuous mode as soon as an external trigger is detected.

If this bit is set when a single pulse mode counting is ongoing, then the timer will not stop at the next match between the LPTIM_ARR and LPTIM_CNT registers and the LPTIM counter keeps counting in Continuous mode.

This bit can be set only when the LPTIM is enabled. It will be automatically reset by hardware.

Bit 1 SNGSTRT: LPTIM start in Single mode

This bit is set by software and cleared by hardware.

In case of software start (TRIGEN[1:0] = '00'), setting this bit starts the LPTIM in single pulse mode.

If the software start is disabled (TRIGEN[1:0] different than '00'), setting this bit starts the LPTIM in single pulse mode as soon as an external trigger is detected.

If this bit is set when the LPTIM is in continuous counting mode, then the LPTIM will stop at the following match between LPTIM_ARR and LPTIM_CNT registers.

This bit can only be set when the LPTIM is enabled. It will be automatically reset by hardware.

Bit 0 ENABLE: LPTIM enable

The ENABLE bit is set and cleared by software.

0:LPTIM is disabled

1:LPTIM is enabled

22.7.6 LPTIM compare register (LPTIM_CMP)

Address offset: 0x014

Reset value: 0x0000 0000

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

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

Bits 15:0 CMP[15:0]: Compare value

CMP is the compare value used by the LPTIM.

Caution: The LPTIM_CMP register must only be modified when the LPTIM is enabled (ENABLE bit set to '1').

22.7.7 LPTIM autoreload register (LPTIM_ARR)

Address offset: 0x018

Reset value: 0x0000 0001

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 15:0 **ARR[15:0]: Auto reload value**

ARR is the autoreload value for the LPTIM.

This value must be strictly greater than the CMP[15:0] value.

Caution: The LPTIM_ARR register must only be modified when the LPTIM is enabled (ENABLE bit set to '1').

22.7.8 LPTIM counter register (LPTIM_CNT)

Address offset: 0x01C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
CNT[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

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

Bits 15:0 **CNT[15:0]: Counter value**

When the LPTIM is running with an asynchronous clock, reading the LPTIM_CNT register may return unreliable values. So in this case it is necessary to perform two consecutive read accesses and verify that the two returned values are identical.

It should be noted that for a reliable LPTIM_CNT register read access, two consecutive read accesses must be performed and compared. A read access can be considered reliable when the values of the two consecutive read accesses are equal.

22.7.9 LPTIM register map

The following table summarizes the LPTIM registers.

Table 134. LPTIM register map and reset values

Offset	Register name	Reset value
0x000	LPTIM_ISR	Res. 31
	Reset value	Res. 30
0x004	LPTIM_ICR	Res. 29
	Reset value	Res. 28
0x008	LPTIM_IER	Res. 27
	Reset value	Res. 26
0x00C	LPTIM_CFGR	Res. 25
	Reset value	Res. 24
0x010	LPTIM_CR	Res. 23
	Reset value	Res. 22
0x014	LPTIM_CMP	Res. 21
	Reset value	Res. 20
0x018	LPTIM_ARR	Res. 19
	Reset value	Res. 18
0x01C	LPTIM_CNT	Res. 17
	Reset value	Res. 16
0x020	LPTIM1_OR	Res. 15
	Reset value	Res. 14
		Res. 13
		Res. 12
		Res. 11
		Res. 10
		Res. 9
		Res. 8
		Res. 7
		Res. 6
		Res. 5
		Res. 4
		Res. 3
		Res. 2
		Res. 1
		Res. 0

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

23 Independent watchdog (IWDG)

23.1 Introduction

The devices feature an embedded watchdog peripheral that offers a combination of high safety level, timing accuracy and flexibility of use. The Independent watchdog peripheral detects and solves malfunctions due to software failure, and triggers system reset when the counter reaches a given timeout value.

The independent watchdog (IWDG) is clocked by its own dedicated low-speed clock (LSI) and thus stays active even if the main clock fails.

The IWDG is best suited for applications that require the watchdog to run as a totally independent process outside the main application, but have lower timing accuracy constraints. For further information on the window watchdog, refer to [Section 24 on page 768](#).

23.2 IWDG main features

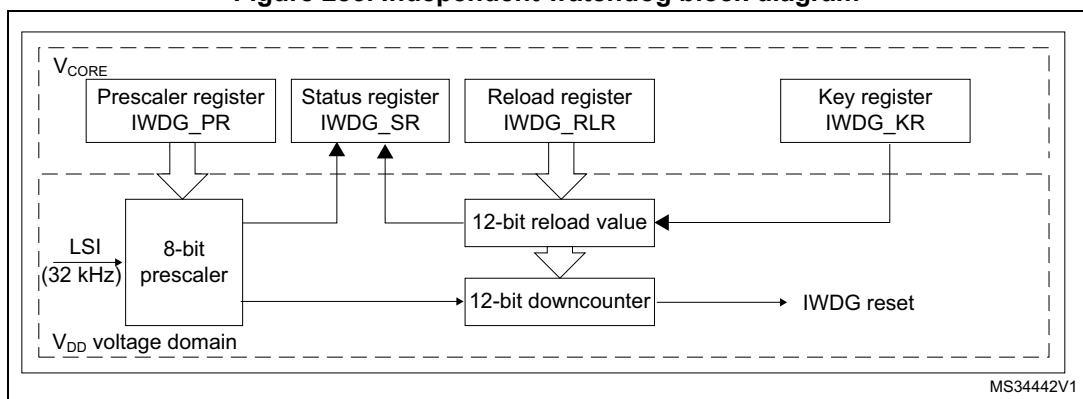
- Free-running downcounter
- Clocked from an independent RC oscillator (can operate in Standby and Stop modes)
- Conditional Reset
 - Reset (if watchdog activated) when the downcounter value becomes lower than 0x000
 - Reset (if watchdog activated) if the downcounter is reloaded outside the window

23.3 IWDG functional description

23.3.1 IWDG block diagram

Figure 283 shows the functional blocks of the independent watchdog module.

Figure 283. Independent watchdog block diagram



1. The watchdog function is implemented in the V_{CORE} voltage domain that is still functional in Stop and Standby modes.

When the independent watchdog is started by writing the value 0x0000 CCCC 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 0x0000 AAAA is written in the [Key register \(IWDG_KR\)](#), the IWDG_RLR value is reloaded in the counter and the watchdog reset is prevented.

23.3.2 Window option

The IWDG can also work as a window watchdog by setting the appropriate window in the [Window register \(IWDG_WINR\)](#).

If the reload operation is performed while the counter is greater than the value stored in the [Window register \(IWDG_WINR\)](#), then a reset is provided.

The default value of the [Window register \(IWDG_WINR\)](#) is 0x0000 0FFF, so if it is not updated, the window option is disabled.

As soon as the window value is changed, a reload operation is performed in order to reset the downcounter to the [Reload register \(IWDG_RLR\)](#) value and ease the cycle number calculation to generate the next reload.

Configuring the IWDG when the window option is enabled

1. Enable the IWDG by writing 0x0000 CCCC in the [Key register \(IWDG_KR\)](#).
2. Enable register access by writing 0x0000 5555 in the [Key register \(IWDG_KR\)](#).
3. Write the IWDG prescaler by programming [Prescaler register \(IWDG_PR\)](#) from 0 to 7.
4. Write the [Reload register \(IWDG_RLR\)](#).
5. Wait for the registers to be updated (IWDG_SR = 0x0000 0000).
6. Write to the [Window register \(IWDG_WINR\)](#). This automatically refreshes the counter value in the [Reload register \(IWDG_RLR\)](#).

Note: Writing the window value allows to refresh the Counter value by the RLR when [Status register \(IWDG_SR\)](#) is set to 0x0000 0000.

Configuring the IWDG when the window option is disabled

When the window option is not used, the IWDG can be configured as follows:

1. Enable the IWDG by writing 0x0000 CCCC in the [Key register \(IWDG_KR\)](#).
2. Enable register access by writing 0x0000 5555 in the [Key register \(IWDG_KR\)](#).
3. Write the prescaler by programming the [Prescaler register \(IWDG_PR\)](#) from 0 to 7.
4. Write the [Reload register \(IWDG_RLR\)](#).
5. Wait for the registers to be updated (IWDG_SR = 0x0000 0000).
6. Refresh the counter value with IWDG_RLR (IWDG_KR = 0x0000 AAAA).

23.3.3 Hardware watchdog

If the “Hardware watchdog” feature is enabled through the device option bits, the watchdog is automatically enabled at power-on, and generates a reset unless the [Key register \(IWDG_KR\)](#) is written by the software before the counter reaches end of count or if the downcounter is reloaded inside the window.

23.3.4 Low-power freeze

Depending on the IWDG_STOP and IWDG_STBY options configuration, the IWDG can continue counting or not during the Stop mode and the Standby mode respectively. If the IWDG is kept running during Stop or Standby modes, it can wake up the device from this mode. Refer to [Section : User and read protection option bytes](#) for more details.

23.3.5 Behavior in Stop and Standby modes

Once running, the IWDG cannot be stopped.

23.3.6 Register access protection

Write access to [Prescaler register \(IWDG_PR\)](#), [Reload register \(IWDG_RLR\)](#) and [Window register \(IWDG_WINR\)](#) is protected. To modify them, the user must first write the code 0x0000 5555 in the [Key register \(IWDG_KR\)](#). A write access to this register with a different value will break the sequence and register access will be protected again. This is the case of the reload operation (writing 0x0000 AAAA).

A status register is available to indicate that an update of the prescaler or the down-counter reload value or the window value is on going.

23.3.7 Debug mode

When the microcontroller enters Debug mode (core halted), the IWDG counter either continues to work normally or stops, depending on DBG_IWDG_STOP configuration bit in DBG module.

23.4 IWDG registers

Refer to [Section 1.2 on page 53](#) for a list of abbreviations used in register descriptions.

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

23.4.1 Key register (IWDG_KR)

Address offset: 0x00

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

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

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

Bits 15:0 **KEY[15:0]**: Key value (write only, read 0x0000)

These bits must be written by software at regular intervals with the key value 0xAAAA, otherwise the watchdog generates a reset when the counter reaches 0.

Writing the key value 0x5555 to enable access to the IWDG_PR, IWDG_RLR and IWDG_WINR registers (see [Section 23.3.6: Register access protection](#))

Writing the key value 0xCCCC starts the watchdog (except if the hardware watchdog option is selected)

23.4.2 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	
Res.	Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	PR[2:0]															
														rw	rw	rw

Bits 31:3 Reserved, must be kept at reset value.

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

These bits are write access protected see [Section 23.3.6: Register access protection](#). They are written by software to select the prescaler divider feeding the counter clock. PVU bit of the [Status register \(IWDG_SR\)](#) must be reset in order to be able to change the prescaler divider.

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

Note: Reading this register returns the prescaler value from the V_{DD} 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 [Status register \(IWDG_SR\)](#) is reset.

23.4.3 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
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.												RL[11:0]
				rw											

Bits 31:12 Reserved, must be kept at reset value.

Bits 11:0 **RL[11:0]**: Watchdog counter reload value

These bits are write access protected see [Register access protection](#). They are written by software to define the value to be loaded in the watchdog counter each time the value 0xAAAA is written in the [Key register \(IWDG_KR\)](#). The watchdog counter counts down from this value. The timeout period is a function of this value and the clock prescaler. Refer to the datasheet for the timeout information.

The RVU bit in the [Status register \(IWDG_SR\)](#) must be reset to be able to change the reload value.

Note: *Reading this register returns the reload value from the V_{DD} voltage domain. This value may not be up to date/valid if a write operation to this register is ongoing on it. For this reason the value read from this register is valid only when the RVU bit in the [Status register \(IWDG_SR\)](#) is reset.*

23.4.4 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
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	WVU	RVU	PVU												
													r	r	r

Bits 31:3 Reserved, must be kept at reset value.

Bit 2 **WVU**: Watchdog counter window value update

This bit is set by hardware to indicate that an update of the window value is ongoing. It is reset by hardware when the reload value update operation is completed in the V_{DD} voltage domain (takes up to five RC 40 kHz cycles).

Window value can be updated only when WVU bit is reset.

This bit is generated only if generic “window” = 1

Bit 1 **RVU**: Watchdog counter reload value update

This bit is set by hardware to indicate that an update of the reload value is ongoing. It is reset by hardware when the reload value update operation is completed in the V_{DD} voltage domain (takes up to five RC 40 kHz cycles).

Reload value can be updated only when RVU bit is reset.

Bit 0 **PVU**: Watchdog prescaler value update

This bit is set by hardware to indicate that an update of the prescaler value is ongoing. It is reset by hardware when the prescaler update operation is completed in the V_{DD} voltage domain (takes up to five RC 40 kHz cycles).

Prescaler value can be updated only when PVU bit is reset.

Note:

If several reload, prescaler, or window values are used by the application, it is mandatory to wait until RVU bit is reset before changing the reload value, to wait until PVU bit is reset before changing the prescaler value, and to wait until WVU bit is reset before changing the window value. However, after updating the prescaler and/or the reload/window value it is not necessary to wait until RVU or PVU or WVU is reset before continuing code execution except in case of low-power mode entry.

23.4.5 Window register (IWDG_WINR)

Address offset: 0x10

Reset value: 0x0000 0FFF (reset by Standby mode)

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

Bits 31:12 Reserved, must be kept at reset value.

Bits 11:0 **WIN[11:0]**: Watchdog counter window value

These bits are write access protected, see [Section 23.3.6](#), they contain the high limit of the window value to be compared with the downcounter.

To prevent a reset, the downcounter must be reloaded when its value is lower than the window register value and greater than 0x0

The WVU bit in the [Status register \(IWDG_SR\)](#) must be reset in order to be able to change the reload value.

Note: Reading this register returns the reload value from the V_{DD} voltage domain. This value may not be valid if a write operation to this register is ongoing. For this reason the value read from this register is valid only when the WVU bit in the Status register (IWDG_SR) is reset.

23.4.6 IWDG register map

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

Table 135. IWDG register map and reset values

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	IWDG_KR	Res	Res																														
	Reset value																																
0x04	IWDG_PR	Res	PR[2:0]	0 0 0																													
	Reset value																																
0x08	IWDG_RLR	Res	RL[11:0]	1 1																													
	Reset value																																
0x0C	IWDG_SR	Res	WVU	0 0 0																													
	Reset value																																
0x10	IWDG_WINR	Res	WIN[11:0]	1 1																													
	Reset value																																

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

24 System window watchdog (WWDG)

24.1 Introduction

The system window watchdog (WWDG) is used to detect the occurrence of a software fault, usually generated by external interference or by unforeseen logical conditions, which causes the application program to abandon its normal sequence. The watchdog circuit generates an MCU reset on expiry of a programmed time period, unless the program refreshes the contents of the 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.

The WWDG clock is prescaled from the APB clock and has a configurable time-window that can be programmed to detect abnormally late or early application behavior.

The WWDG is best suited for applications which require the watchdog to react within an accurate timing window.

24.2 WWDG main features

- Programmable free-running downcounter
- Conditional reset
 - Reset (if watchdog activated) when the downcounter value becomes lower than 0x40
 - Reset (if watchdog activated) if the downcounter is reloaded outside the window (see [Figure 285](#))
- Early wakeup interrupt (EWI): triggered (if enabled and the watchdog activated) when the downcounter is equal to 0x40.

24.3 WWDG functional description

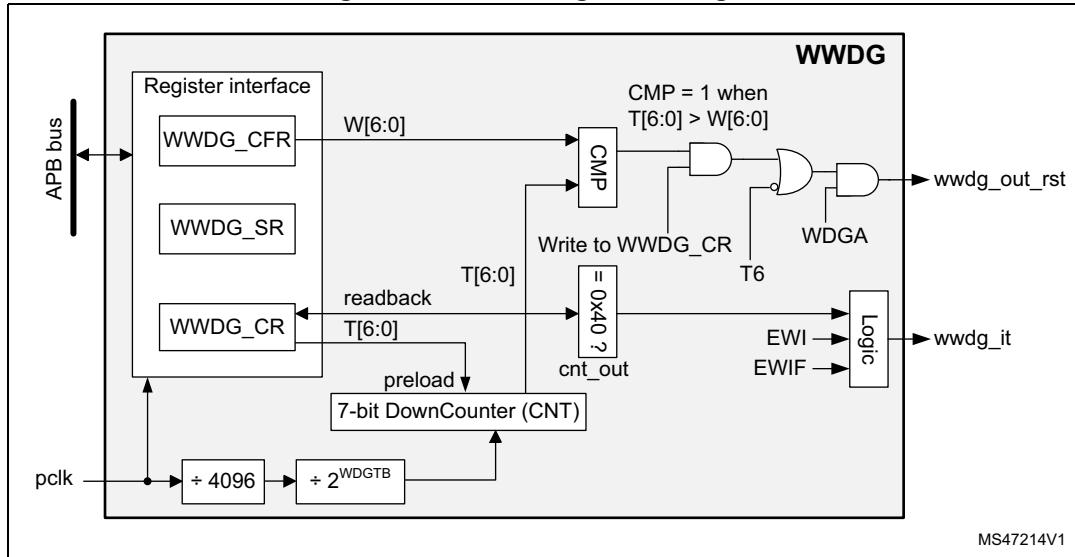
If the watchdog is activated (the WDGA bit is set in the WWDG_CR register) and when the 7-bit downcounter (T[6:0] bits) is decremented from 0x40 to 0x3F (T6 becomes cleared), it initiates a reset. If the software reloads the counter while the counter is greater than the value stored in the window register,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

The application program must write in the WWDG_CR register at regular intervals during normal operation to prevent an MCU reset. This operation must occur only when the counter value is lower than the window register value and higher than 0x3F. The value to be stored in the WWDG_CR register must be between 0xFF and 0xC0.

Refer to [Figure 284](#) for the WWDG block diagram.

24.3.1 WWDG block diagram

Figure 284. Watchdog block diagram



24.3.2 Enabling the watchdog

When the user option WWDG_SW selects “Software window 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.

When the user option WWDG_SW selects “Hardware window watchdog”, the watchdog is always enabled after a reset, it cannot be disabled.

24.3.3 Controlling the downcounter

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

The T[5:0] bits contain the number of increments 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 285](#)). 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 285](#) describes the window watchdog process.

Note: *The T6 bit can be used to generate a software reset (the WDGA bit is set and the T6 bit is cleared).*

24.3.4 Advanced watchdog interrupt feature

The early wakeup interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. The EWI interrupt is enabled by setting the EWI bit in the WWDG_CFR register. When the downcounter reaches the value 0x40, an EWI interrupt is generated and the corresponding interrupt service routine (ISR)

can be used to trigger specific actions (such as communications or data logging), before resetting the device.

In some applications, the EWI interrupt can be used to manage a software system check and/or system recovery/graceful degradation, without generating a WWDG reset. In this case, the corresponding interrupt service routine (ISR) should reload the WWDG counter to avoid the WWDG reset, then trigger the required actions.

The EWI interrupt is cleared by writing '0' to the EWIF bit in the WWDG_SR register.

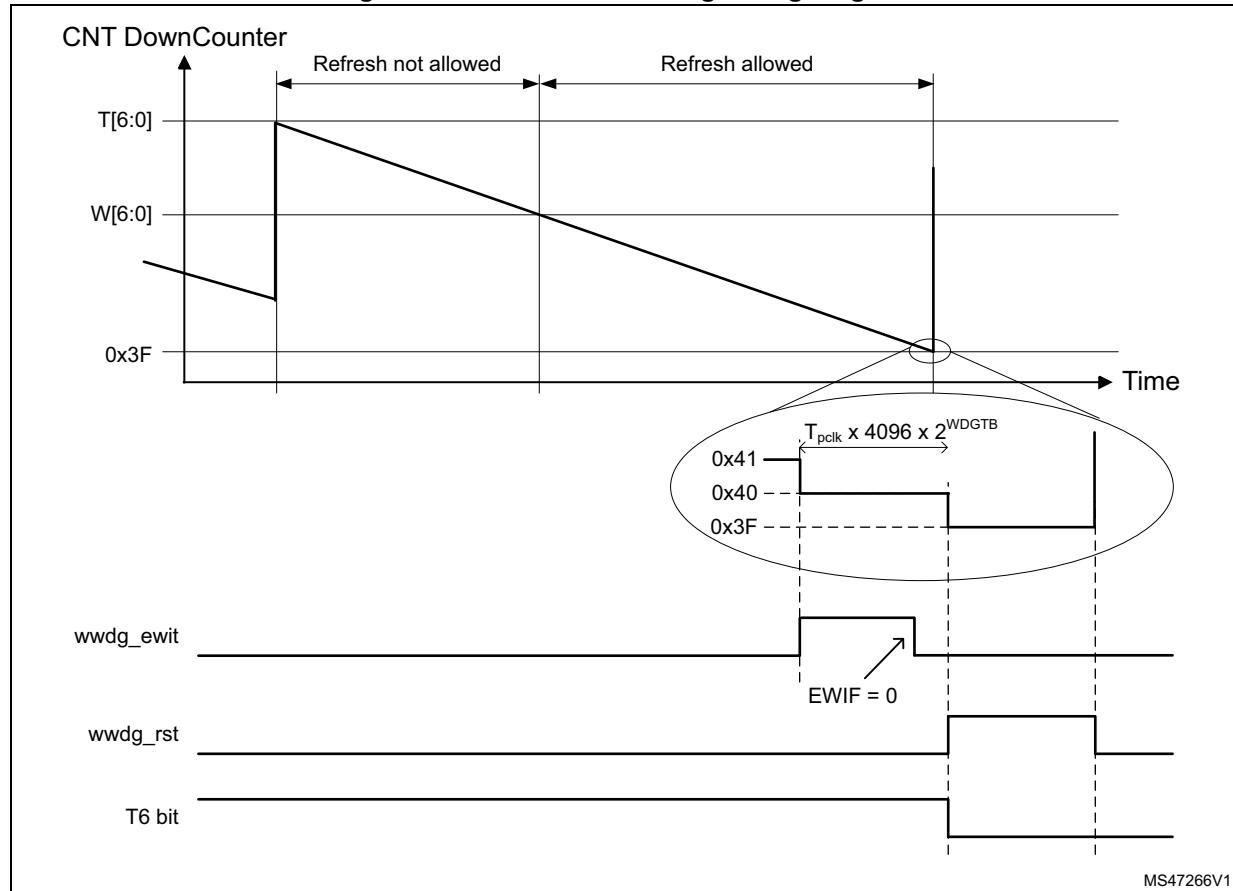
Note: *When the EWI interrupt cannot be served, e.g. due to a system lock in a higher priority task, the WWDG reset is eventually generated.*

24.3.5 How to program the watchdog timeout

Use the formula in [Figure 285](#) to calculate the WWDG timeout.

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

Figure 285. Window watchdog timing diagram



The formula to calculate the timeout value is given by:

$$t_{\text{WWDG}} = t_{\text{PCLK}} \times 4096 \times 2^{\text{WDGTB}[1:0]} \times (\text{T}[5:0] + 1) \quad (\text{ms})$$

where:

t_{WWDG} : WWDG timeout

t_{PCLK} : APB clock period measured in ms

4096: value corresponding to internal divider

As an example, lets assume APB frequency is equal to 48 MHz, WDGTB1:0] is set to 3 and T[5:0] is set to 63:

$$t_{\text{WWDG}} = (1/48000) \times 4096 \times 2^3 \times (63 + 1) = 43.69\text{ms}$$

Refer to the datasheet for the minimum and maximum values of the t_{WWDG} .

24.3.6 Debug mode

When the microcontroller enters debug mode (processor halted), the WWDG counter either continues to work normally or stops, depending on the configuration bit in DBG module. For more details refer to [Section 40.16.2: Debug support for timers, watchdog, bxCAN and I2C](#).

24.4 WWDG registers

Refer to [Section 1.2 on page 53](#) for a list of abbreviations used in register descriptions.

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

24.4.1 Control register (WWDG_CR)

Address offset: 0x000

Reset value: 0x0000 007F

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

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

Bit 7 **WDGA**: Activation bit

This bit is set by software and only cleared by hardware after a reset. When WDGA = 1, the watchdog can generate a reset.

- 0: Watchdog disabled
- 1: Watchdog enabled

Bits 6:0 **T[6:0]**: 7-bit counter (MSB to LSB)

These bits contain the value of the watchdog counter, decremented every $(4096 \times 2^{\text{WDGTB1:0}})$ PCLK cycles. A reset is produced when it is decremented from 0x40 to 0x3F (T6 becomes cleared).

24.4.2 Configuration register (WWDG_CFR)

Address offset: 0x004

Reset value: 0x0000 007F

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

Bits 31:10 Reserved, must be kept at reset value.

Bit 9 **EWI**: Early wakeup interrupt

When set, an interrupt occurs whenever the counter reaches the value 0x40. This interrupt is only cleared by hardware after a reset.

Bits 8:7 **WDGTB[1:0]**: Timer base

The time base of the prescaler can be modified as follows:

- 00: CK Counter Clock (PCLK div 4096) div 1
- 01: CK Counter Clock (PCLK div 4096) div 2
- 10: CK Counter Clock (PCLK div 4096) div 4
- 11: CK Counter Clock (PCLK div 4096) div 8

Bits 6:0 **W[6:0]**: 7-bit window value

These bits contain the window value to be compared with the downcounter.

24.4.3 Status register (WWDG_SR)

Address offset: 0x008

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	EWIF														
															rc_w0

Bits 31:1 Reserved, must be kept at reset value.

Bit 0 **EWIF**: Early wakeup interrupt flag

This bit is set by hardware when the counter has reached the value 0x40. It must be cleared by software by writing '0'. Writing '1' has no effect. This bit is also set if the interrupt is not enabled.

24.4.4 WWDG register map

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

Table 136. WWDG register map and reset values

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

25 Real-time clock (RTC)

25.1 Introduction

The RTC provides an automatic wakeup to manage all low-power modes.

The real-time clock (RTC) is an independent BCD timer/counter. The RTC provides a time-of-day clock/calendar with programmable alarm interrupts.

The RTC includes also a periodic programmable wakeup flag with interrupt capability.

Two 32-bit registers contain the seconds, minutes, hours (12- or 24-hour format), day (day of week), date (day of month), month, and year, expressed in binary coded decimal format (BCD). The sub-seconds value is also available in binary format.

Compensations for 28-, 29- (leap year), 30-, and 31-day months are performed automatically. Daylight saving time compensation can also be performed.

Additional 32-bit registers contain the programmable alarm subseconds, seconds, minutes, hours, day, and date.

A digital calibration feature is available to compensate for any deviation in crystal oscillator accuracy.

After Backup domain reset, all RTC registers are protected against possible parasitic write accesses.

As long as the supply voltage remains in the operating range, the RTC never stops, regardless of the device status (Run mode, low-power mode or under reset).

25.2 RTC main features

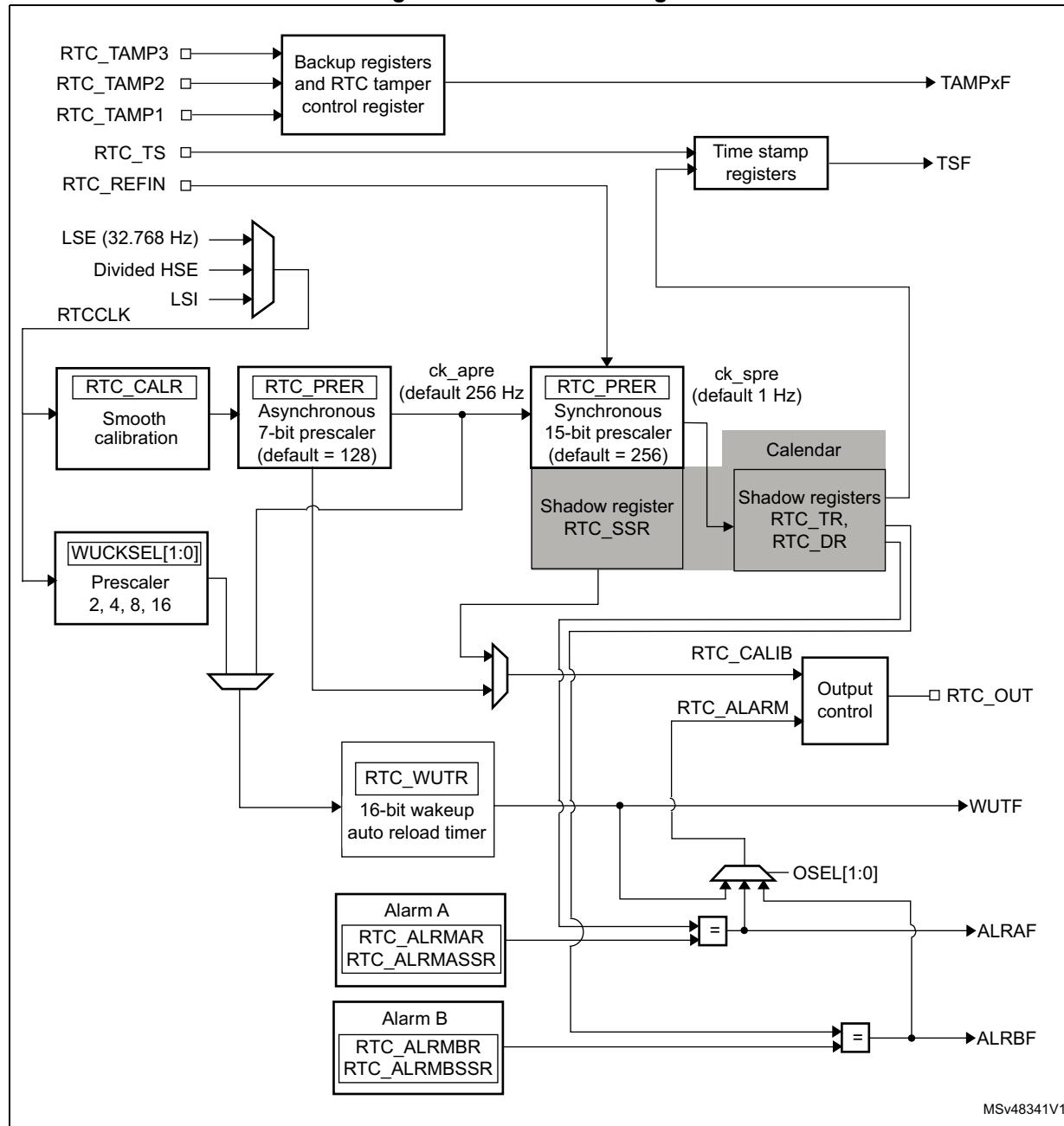
The RTC unit main features are the following (see [Figure 286: RTC block diagram](#)):

- Calendar with subseconds, seconds, minutes, hours (12 or 24 format), day (day of week), date (day of month), month, and year.
- Daylight saving compensation programmable by software.
- Programmable alarm with interrupt function. The alarm can be triggered by any combination of the calendar fields.
- Automatic wakeup unit generating a periodic flag that triggers an automatic wakeup interrupt.
- Reference clock detection: a more precise second source clock (50 or 60 Hz) can be used to enhance the calendar precision.
- Accurate synchronization with an external clock using the subsecond shift feature.
- Digital calibration circuit (periodic counter correction): 0.95 ppm accuracy, obtained in a calibration window of several seconds
- Time-stamp function for event saving
- Tamper detection event with configurable filter and internal pull-up
- Maskable interrupts/events:
 - Alarm A
 - Alarm B
 - Wakeup interrupt
 - Time-stamp
 - Tamper detection
- 32 backup registers.

25.3 RTC functional description

25.3.1 RTC block diagram

Figure 286. RTC block diagram



The RTC includes:

- Two alarms
- Three tamper events from I/Os
 - Tamper detection erases the backup registers.
- One timestamp event from I/O
- Tamper event detection can generate a timestamp event
- Timestamp can be generated when a switch to V_{BAT} occurs
- 32 x 32-bit backup registers
 - The backup registers (RTC_BKPxR) are implemented in the RTC domain that remains powered-on by VBAT when the VDD power is switched off.
- Output functions: RTC_OUT which selects one of the following two outputs:
 - RTC_CALIB: 512 Hz or 1Hz clock output (with an LSE frequency of 32.768 kHz). This output is enabled by setting the COE bit in the RTC_CR register.
 - RTC_ALARM: This output is enabled by configuring the OSEL[1:0] bits in the RTC_CR register which select the Alarm A, Alarm B or Wakeup outputs.
- Input functions:
 - RTC_TS: timestamp event
 - RTC_TAMP1: tamper1 event detection
 - RTC_TAMP2: tamper2 event detection
 - RTC_TAMP3: tamper3 event detection
 - RTC_REFIN: 50 or 60 Hz reference clock input

25.3.2 GPIOs controlled by the RTC

RTC_OUT, RTC_TS and RTC_TAMP1 are mapped on the same pin (PC13). PC13 pin configuration is controlled by the RTC, whatever the PC13 GPIO configuration, except for the RTC_ALARM output open-drain mode. The RTC functions mapped on PC13 are available in all low-power modes and in VBAT mode.

The output mechanism follows the priority order shown in [Table 137](#).

Table 137. RTC pin PC13 configuration⁽¹⁾

PC13 Pin configuration and function	OSEL[1:0] bits (RTC_ALARM output enable)	COE bit (RTC_CALIB output enable)	RTC_ALARM_TYPE bit	TAMP1E bit (RTC_TAMP1 input enable)	TSE bit (RTC_TS input enable)	TSINSEL bits
RTC_ALARM output OD	01 or 10 or 11	Don't care	0	Don't care	Don't care	Don't care
RTC_ALARM output PP	01 or 10 or 11	Don't care	1	Don't care	Don't care	Don't care
RTC_CALIB output PP	00	1	Don't care	Don't care	Don't care	Don't care

Table 137. RTC pin PC13 configuration⁽¹⁾ (continued)

PC13 Pin configuration and function	OSEL[1:0] bits (RTC_ALARM output enable)	COE bit (RTC_CALIB output enable)	RTC_ALARM_TYPE bit	TAMP1E bit (RTC_TAMP1 input enable)	TSE bit (RTC_TS input enable)	TSINSEL bits
RTC_TAMP1 input floating	00	0	Don't care	1	0	Don't care
	00	1				
	01 or 10 or 11	0				
RTC_TS and RTC_TAMP1 input floating	00	0	Don't care	1	1	00
	00	1				
	01 or 10 or 11	0				
RTC_TS input floating	00	0	Don't care	0	1	00
	00	1				
	01 or 10 or 11	0				
Wakeup pin or Standard GPIO	00	0	Don't care	0	0	Don't care

1. OD: open drain; PP: push-pull.

RTC_TAMP2 and RTC_TS are mapped on the same pin (PI8). PI8 configuration is controlled by the RTC, whatever the PI8 GPIO configuration. The RTC functions mapped on PI8 are available in all low-power modes and in VBAT mode.

The output mechanism follows the priority order shown in [Table 138](#).

Table 138. RTC pin PI8 configuration

PI8 pin configuration and function	TAMP2E bit (RTC_TAMP2 input enable)	TSE bit (RTC_TS input enable)	TSINSEL bit (Timestamp pin selection)
RTC_TAMP2 input floating	1	0	Don't care
RTC_TS and RTC_TAMP2 input floating	1	1	01
RTC_TS input floating	0	1	01
Wakeup pin or Standard GPIO	0	0	Don't care

RTC_TAMP3 and RTC_TS are mapped on the same pin (PC1). PC1 configuration is controlled by the RTC, whatever the PC1 GPIO configuration. The RTC functions mapped on PC1 are available in all low-power modes, but are not available in VBAT mode.

Table 139. RTC pin PC2 configuration

PC2 pin configuration and function	TAMP2E bit (RTC_TAMP3 input enable)	TSE bit (RTC_TS input enable)	TSINSEL bit (Timestamp pin selection)
RTC_TAMP3 input floating	1	0	Don't care
RTC_TS and RTC_TAMP3 input floating	1	1	10 or 11
RTC_TS input floating	0	1	01
Wakeup pin or Standard GPIO	0	0	Don't care

RTC_REFIN is mapped on PB15. PB15 must be configured in alternate function mode to allow RTC_REFIN function. RTC_REFIN is not available in VBAT and in Standby mode.

The table below summarizes the RTC pins and functions capabilities in all modes.

Table 140. RTC functions over modes

Pin	RTC functions	Functional in all low-power modes except Standby modes	Functional in Standby mode	Functional in VBAT mode
PC13	RTC_TAMP1 RTC_TS RTC_OUT	YES	YES	YES
PI8	RTC_TAMP2 RTC_TS	YES	YES	YES
PC1	RTC_TAMP3 RTC_TS	YES	YES	NO
PB15	RTC_REFIN	YES	NO	NO

25.3.3 Clock and prescalers

The RTC clock source (RTCCCLK) is selected through the clock controller among the LSE clock, the LSI oscillator clock, and the HSE clock. For more information on the RTC clock source configuration, refer to [Section 5: Reset and clock control \(RCC\)](#).

A programmable prescaler stage generates a 1 Hz clock which is used to update the calendar. To minimize power consumption, the prescaler is split into 2 programmable prescalers (see [Figure 286: RTC block diagram](#)):

- A 7-bit asynchronous prescaler configured through the PREDIV_A bits of the RTC_PRER register.
- A 15-bit synchronous prescaler configured through the PREDIV_S bits of the RTC_PRER register.

Note: When both prescalers are used, it is recommended to configure the asynchronous prescaler to a high value to minimize consumption.

The asynchronous prescaler division factor is set to 128, and the synchronous division factor to 256, to obtain an internal clock frequency of 1 Hz (`ck_spre`) with an LSE frequency of 32.768 kHz.

The minimum division factor is 1 and the maximum division factor is 2^{22} .

This corresponds to a maximum input frequency of around 4 MHz.

f_{ck_apre} is given by the following formula:

$$f_{CK_APRE} = \frac{f_{RTCCLK}}{\text{PREDIV_A} + 1}$$

The `ck_apre` clock is used to clock the binary RTC_SSR subseconds downcounter. When it reaches 0, RTC_SSR is reloaded with the content of PREDIV_S.

f_{ck_spre} is given by the following formula:

$$f_{CK_SPRE} = \frac{f_{RTCCLK}}{(\text{PREDIV_S} + 1) \times (\text{PREDIV_A} + 1)}$$

The `ck_spre` clock can be used either to update the calendar or as timebase for the 16-bit wakeup auto-reload timer. To obtain short timeout periods, the 16-bit wakeup auto-reload timer can also run with the RTCCLK divided by the programmable 4-bit asynchronous prescaler (see [Section 25.3.6: Periodic auto-wakeup](#) for details).

25.3.4 Real-time clock and calendar

The RTC calendar time and date registers are accessed through shadow registers which are synchronized with PCLK (APB clock). They can also be accessed directly in order to avoid waiting for the synchronization duration.

- RTC_SSR for the subseconds
- RTC_TR for the time
- RTC_DR for the date

Every RTCCLK period, the current calendar value is copied into the shadow registers, and the RSF bit of RTC_ISR register is set (see [Section 25.6.4: RTC initialization and status register \(RTC_ISR\)](#)). The copy is not performed in Stop and Standby mode. When exiting these modes, the shadow registers are updated after up to 1 RTCCLK period.

When the application reads the calendar registers, it accesses the content of the shadow registers. It is possible to make a direct access to the calendar registers by setting the BYPSHAD control bit in the RTC_CR register. By default, this bit is cleared, and the user accesses the shadow registers.

When reading the RTC_SSR, RTC_TR or RTC_DR registers in BYPSHAD=0 mode, the frequency of the APB clock (f_{APB}) must be at least 7 times the frequency of the RTC clock (f_{RTCCLK}).

The shadow registers are reset by system reset.

25.3.5 Programmable alarms

The RTC unit provides programmable alarm: Alarm A and Alarm B. The description below is given for Alarm A, but can be translated in the same way for Alarm B.

The programmable alarm function is enabled through the ALRAE bit in the RTC_CR register. The ALRAF is set to 1 if the calendar subseconds, seconds, minutes, hours, date or day match the values programmed in the alarm registers RTC_ALRMASSR and RTC_ALRMAR. Each calendar field can be independently selected through the MSKx bits of the RTC_ALRMAR register, and through the MASKSSx bits of the RTC_ALRMASSR register. The alarm interrupt is enabled through the ALRAIE bit in the RTC_CR register.

Caution: If the seconds field is selected (MSK1 bit reset in RTC_ALRMAR), the synchronous prescaler division factor set in the RTC_PRER register must be at least 3 to ensure correct behavior.

Alarm A and Alarm B (if enabled by bits OSEL[1:0] in RTC_CR register) can be routed to the RTC_ALARM output. RTC_ALARM output polarity can be configured through bit POL the RTC_CR register.

25.3.6 Periodic auto-wakeup

The periodic wakeup flag is generated by a 16-bit programmable auto-reload down-counter. The wakeup timer range can be extended to 17 bits.

The wakeup function is enabled through the WUTE bit in the RTC_CR register.

The wakeup timer clock input can be:

- RTC clock (RTCCLK) divided by 2, 4, 8, or 16.
When RTCCLK is LSE(32.768kHz), this allows to configure the wakeup interrupt period from 122 µs to 32 s, with a resolution down to 61 µs.
- ck_spre (usually 1 Hz internal clock)
When ck_spre frequency is 1Hz, this allows to achieve a wakeup time from 1 s to around 36 hours with one-second resolution. This large programmable time range is divided in 2 parts:
 - from 1s to 18 hours when WUCKSEL [2:1] = 10
 - and from around 18h to 36h when WUCKSEL[2:1] = 11. In this last case 2¹⁶ is added to the 16-bit counter current value. When the initialization sequence is complete (see [Programming the wakeup timer on page 784](#)), the timer starts counting down. When the wakeup function is enabled, the down-counting remains active in low-power modes. In addition, when it reaches 0, the WUTF flag is set in the RTC_ISR register, and the wakeup counter is automatically reloaded with its reload value (RTC_WUTR register value).

The WUTF flag must then be cleared by software.

When the periodic wakeup interrupt is enabled by setting the WUTIE bit in the RTC_CR register, it can exit the device from low-power modes.

The periodic wakeup flag can be routed to the RTC_ALARM output provided it has been enabled through bits OSEL[1:0] of RTC_CR register. RTC_ALARM output polarity can be configured through the POL bit in the RTC_CR register.

System reset, as well as low-power modes (Sleep, Stop and Standby) have no influence on the wakeup timer.

25.3.7 RTC initialization and configuration

RTC register access

The RTC registers are 32-bit registers. The APB interface introduces 2 wait-states in RTC register accesses except on read accesses to calendar shadow registers when BYPSHAD=0.

RTC register write protection

After system reset, the RTC registers are protected against parasitic write access by clearing the DBP bit in the PWR_CR1 register (refer to the power control section). DBP bit must be set in order to enable RTC registers write access.

After Backup domain reset, all the RTC registers are write-protected. Writing to the RTC registers is enabled by writing a key into the Write Protection register, RTC_WPR.

The following steps are required to unlock the write protection on all the RTC registers except for RTC_TAMPCR, RTC_BKPxR, RTC_OR and RTC_ISR[13:8].

1. Write '0xCA' into the RTC_WPR register.
2. Write '0x53' into the RTC_WPR register.

Writing a wrong key reactivates the write protection.

The protection mechanism is not affected by system reset.

Calendar initialization and configuration

To program the initial time and date calendar values, including the time format and the prescaler configuration, the following sequence is required:

1. Set INIT bit to 1 in the RTC_ISR register to enter initialization mode. In this mode, the calendar counter is stopped and its value can be updated.
2. Poll INITF bit of in the RTC_ISR register. The initialization phase mode is entered when INITF is set to 1. It takes around 2 RTCCLK clock cycles (due to clock synchronization).
3. To generate a 1 Hz clock for the calendar counter, program both the prescaler factors in RTC_PRER register.
4. Load the initial time and date values in the shadow registers (RTC_TR and RTC_DR), and configure the time format (12 or 24 hours) through the FMT bit in the RTC_CR register.
5. Exit the initialization mode by clearing the INIT bit. The actual calendar counter value is then automatically loaded and the counting restarts after 4 RTCCLK clock cycles.

When the initialization sequence is complete, the calendar starts counting.

Note:

After a system reset, the application can read the INITS flag in the RTC_ISR register to check if the calendar has been initialized or not. If this flag equals 0, the calendar has not been initialized since the year field is set at its Backup domain reset default value (0x00).

To read the calendar after initialization, the software must first check that the RSF flag is set in the RTC_ISR register.

Daylight saving time

The daylight saving time management is performed through bits SUB1H, ADD1H, and BKP of the RTC_CR register.

Using SUB1H or ADD1H, the software can subtract or add one hour to the calendar in one single operation without going through the initialization procedure.

In addition, the software can use the BKP bit to memorize this operation.

Programming the alarm

A similar procedure must be followed to program or update the programmable alarms. The procedure below is given for Alarm A but can be translated in the same way for Alarm B.

1. Clear ALRAE in RTC_CR to disable Alarm A.
2. Program the Alarm A registers (RTC_ALRMASSR/RTC_ALRMAR).
3. Set ALRAE in the RTC_CR register to enable Alarm A again.

Note:

Each change of the RTC_CR register is taken into account after around 2 RTCCLK clock cycles due to clock synchronization.

Programming the wakeup timer

The following sequence is required to configure or change the wakeup timer auto-reload value (WUT[15:0] in RTC_WUTR):

1. Clear WUTE in RTC_CR to disable the wakeup timer.
2. Poll WUTWF until it is set in RTC_ISR to make sure the access to wakeup auto-reload counter and to WUCKSEL[2:0] bits is allowed. It takes around 2 RTCCLK clock cycles (due to clock synchronization).
3. Program the wakeup auto-reload value WUT[15:0], and the wakeup clock selection (WUCKSEL[2:0] bits in RTC_CR). Set WUTE in RTC_CR to enable the timer again. The wakeup timer restarts down-counting. The WUTWF bit is cleared up to 2 RTCCLK clock cycles after WUTE is cleared, due to clock synchronization.

25.3.8 Reading the calendar

When BYPSHAD control bit is cleared in the RTC_CR register

To read the RTC calendar registers (RTC_SSR, RTC_TR and RTC_DR) properly, the APB clock frequency (f_{PCLK}) must be equal to or greater than seven times the RTC clock frequency (f_{RTCCLK}). This ensures a secure behavior of the synchronization mechanism.

If the APB clock frequency is less than seven times the RTC clock frequency, the software must read the calendar time and date registers twice. If the second read of the RTC_TR gives the same result as the first read, this ensures that the data is correct. Otherwise a third read access must be done. In any case the APB clock frequency must never be lower than the RTC clock frequency.

The RSF bit is set in RTC_ISR register each time the calendar registers are copied into the RTC_SSR, RTC_TR and RTC_DR shadow registers. The copy is performed every RTCCLK cycles. To ensure consistency between the 3 values, reading either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read. In case the software makes read accesses to the calendar in a time interval smaller than 2 RTCCLK periods: RSF must be cleared by software after the first calendar read, and

then the software must wait until RSF is set before reading again the RTC_SSR, RTC_TR and RTC_DR registers.

After waking up from low-power mode (Stop or Standby), RSF must be cleared by software. The software must then wait until it is set again before reading the RTC_SSR, RTC_TR and RTC_DR registers.

The RSF bit must be cleared after wakeup and not before entering low-power mode.

After a system reset, the software must wait until RSF is set before reading the RTC_SSR, RTC_TR and RTC_DR registers. Indeed, a system reset resets the shadow registers to their default values.

After an initialization (refer to [Calendar initialization and configuration on page 783](#)): the software must wait until RSF is set before reading the RTC_SSR, RTC_TR and RTC_DR registers.

After synchronization (refer to [Section 25.3.10: RTC synchronization](#)): the software must wait until RSF is set before reading the RTC_SSR, RTC_TR and RTC_DR registers.

When the BYPSHAD control bit is set in the RTC_CR register (bypass shadow registers)

Reading the calendar registers gives the values from the calendar counters directly, thus eliminating the need to wait for the RSF bit to be set. This is especially useful after exiting from low-power modes (STOP or Standby), since the shadow registers are not updated during these modes.

When the BYPSHAD bit is set to 1, the results of the different registers might not be coherent with each other if an RTCCLK edge occurs between two read accesses to the registers. Additionally, the value of one of the registers may be incorrect if an RTCCLK edge occurs during the read operation. The software must read all the registers twice, and then compare the results to confirm that the data is coherent and correct. Alternatively, the software can just compare the two results of the least-significant calendar register.

Note: While BYPSHAD=1, instructions which read the calendar registers require one extra APB cycle to complete.

25.3.9 Resetting the RTC

The calendar shadow registers (RTC_SSR, RTC_TR and RTC_DR) and some bits of the RTC status register (RTC_ISR) are reset to their default values by all available system reset sources.

On the contrary, the following registers are reset to their default values by a Backup domain reset and are not affected by a system reset: the RTC current calendar registers, the RTC control register (RTC_CR), the prescaler register (RTC_PRER), the RTC calibration register (RTC_CALR), the RTC shift register (RTC_SHIFTR), the RTC timestamp registers (RTC_TSSSR, RTC_TSTR and RTC_TSDDR), the RTC tamper configuration register (RTC_TAMPCCR), the RTC backup registers (RTC_BKPxR), the wakeup timer register (RTC_WUTR), the Alarm A and Alarm B registers (RTC_ALRMASSR/RTC_ALRMAR and RTC_ALRMBSSR/RTC_ALRMBR), and the Option register (RTC_OR).

In addition, when it is clocked by the LSE, the RTC keeps on running under system reset if the reset source is different from the Backup domain reset one (refer to the RTC clock section of the Reset and clock controller for details on the list of RTC clock sources not

affected by system reset). When a Backup domain reset occurs, the RTC is stopped and all the RTC registers are set to their reset values.

25.3.10 RTC synchronization

The RTC can be synchronized to a remote clock with a high degree of precision. After reading the sub-second field (RTC_SSR or RTC_TSSSR), a calculation can be made of the precise offset between the times being maintained by the remote clock and the RTC. The RTC can then be adjusted to eliminate this offset by “shifting” its clock by a fraction of a second using RTC_SHIFTR.

RTC_SSR contains the value of the synchronous prescaler counter. This allows one to calculate the exact time being maintained by the RTC down to a resolution of $1 / (\text{PREDIV_S} + 1)$ seconds. As a consequence, the resolution can be improved by increasing the synchronous prescaler value (PREDIV_S[14:0]. The maximum resolution allowed (30.52 μ s with a 32768 Hz clock) is obtained with PREDIV_S set to 0x7FFF.

However, increasing PREDIV_S means that PREDIV_A must be decreased in order to maintain the synchronous prescaler output at 1 Hz. In this way, the frequency of the asynchronous prescaler output increases, which may increase the RTC dynamic consumption.

The RTC can be finely adjusted using the RTC shift control register (RTC_SHIFTR). Writing to RTC_SHIFTR can shift (either delay or advance) the clock by up to a second with a resolution of $1 / (\text{PREDIV_S} + 1)$ seconds. The shift operation consists of adding the SUBFS[14:0] value to the synchronous prescaler counter SS[15:0]: this will delay the clock. If at the same time the ADD1S bit is set, this results in adding one second and at the same time subtracting a fraction of second, so this will advance the clock.

Caution: Before initiating a shift operation, the user must check that SS[15] = 0 in order to ensure that no overflow will occur.

As soon as a shift operation is initiated by a write to the RTC_SHIFTR register, the SHPF flag is set by hardware to indicate that a shift operation is pending. This bit is cleared by hardware as soon as the shift operation has completed.

Caution: This synchronization feature is not compatible with the reference clock detection feature: firmware must not write to RTC_SHIFTR when REFCKON=1.

25.3.11 RTC reference clock detection

The update of the RTC calendar can be synchronized to a reference clock, RTC_REFIN, which is usually the mains frequency (50 or 60 Hz). The precision of the RTC_REFIN reference clock should be higher than the 32.768 kHz LSE clock. When the RTC_REFIN detection is enabled (REFCKON bit of RTC_CR set to 1), the calendar is still clocked by the LSE, and RTC_REFIN is used to compensate for the imprecision of the calendar update frequency (1 Hz).

Each 1 Hz clock edge is compared to the nearest RTC_REFIN clock edge (if one is found within a given time window). In most cases, the two clock edges are properly aligned. When the 1 Hz clock becomes misaligned due to the imprecision of the LSE clock, the RTC shifts the 1 Hz clock a bit so that future 1 Hz clock edges are aligned. Thanks to this mechanism, the calendar becomes as precise as the reference clock.

The RTC detects if the reference clock source is present by using the 256 Hz clock (ck_apre) generated from the 32.768 kHz quartz. The detection is performed during a time

window around each of the calendar updates (every 1 s). The window equals 7 ck_apre periods when detecting the first reference clock edge. A smaller window of 3 ck_apre periods is used for subsequent calendar updates.

Each time the reference clock is detected in the window, the synchronous prescaler which outputs the ck_spre clock is forced to reload. This has no effect when the reference clock and the 1 Hz clock are aligned because the prescaler is being reloaded at the same moment. When the clocks are not aligned, the reload shifts future 1 Hz clock edges a little for them to be aligned with the reference clock.

If the reference clock halts (no reference clock edge occurred during the 3 ck_apre window), the calendar is updated continuously based solely on the LSE clock. The RTC then waits for the reference clock using a large 7 ck_apre period detection window centered on the ck_spre edge.

When the RTC_REFIN detection is enabled, PREDIV_A and PREDIV_S must be set to their default values:

- PREDIV_A = 0x007F
- PREDIV_S = 0x00FF

Note: *RTC_REFIN clock detection is not available in Standby mode.*

25.3.12 RTC smooth digital calibration

The RTC frequency can be digitally calibrated with a resolution of about 0.954 ppm with a range from -487.1 ppm to +488.5 ppm. The correction of the frequency is performed using series of small adjustments (adding and/or subtracting individual RTCCLK pulses). These adjustments are fairly well distributed so that the RTC is well calibrated even when observed over short durations of time.

The smooth digital calibration is performed during a cycle of about 2^{20} RTCCLK pulses, or 32 seconds when the input frequency is 32768 Hz. This cycle is maintained by a 20-bit counter, cal_cnt[19:0], clocked by RTCCLK.

The smooth calibration register (RTC_CALR) specifies the number of RTCCLK clock cycles to be masked during the 32-second cycle:

- Setting the bit CALM[0] to 1 causes exactly one pulse to be masked during the 32-second cycle.
- Setting CALM[1] to 1 causes two additional cycles to be masked
- Setting CALM[2] to 1 causes four additional cycles to be masked
- and so on up to CALM[8] set to 1 which causes 256 clocks to be masked.

Note: *CALM[8:0] (RTC_CALR) specifies the number of RTCCLK pulses to be masked during the 32-second cycle. Setting the bit CALM[0] to '1' causes exactly one pulse to be masked during the 32-second cycle at the moment when cal_cnt[19:0] is 0x80000; CALM[1]=1 causes two other cycles to be masked (when cal_cnt is 0x40000 and 0xC0000); CALM[2]=1 causes four other cycles to be masked (cal_cnt = 0x20000/0x60000/0xA0000/ 0xE0000); and so on up to CALM[8]=1 which causes 256 clocks to be masked (cal_cnt = 0xXX800).*

While CALM allows the RTC frequency to be reduced by up to 487.1 ppm with fine resolution, the bit CALP can be used to increase the frequency by 488.5 ppm. Setting CALP to '1' effectively inserts an extra RTCCLK pulse every 2^{11} RTCCLK cycles, which means that 512 clocks are added during every 32-second cycle.

Using CALM together with CALP, an offset ranging from -511 to +512 RTCCLK cycles can be added during the 32-second cycle, which translates to a calibration range of -487.1 ppm to +488.5 ppm with a resolution of about 0.954 ppm.

The formula to calculate the effective calibrated frequency (F_{CAL}) given the input frequency (F_{RTCCLK}) is as follows:

$$F_{CAL} = F_{RTCCLK} \times [1 + (CALP \times 512 - CALM) / (2^{20} + CALM - CALP \times 512)]$$

Calibration when PREDIV_A<3

The CALP bit can not be set to 1 when the asynchronous prescaler value (PREDIV_A bits in RTC_PRER register) is less than 3. If CALP was already set to 1 and PREDIV_A bits are set to a value less than 3, CALP is ignored and the calibration operates as if CALP was equal to 0.

To perform a calibration with PREDIV_A less than 3, the synchronous prescaler value (PREDIV_S) should be reduced so that each second is accelerated by 8 RTCCLK clock cycles, which is equivalent to adding 256 clock cycles every 32 seconds. As a result, between 255 and 256 clock pulses (corresponding to a calibration range from 243.3 to 244.1 ppm) can effectively be added during each 32-second cycle using only the CALM bits.

With a nominal RTCCLK frequency of 32768 Hz, when PREDIV_A equals 1 (division factor of 2), PREDIV_S should be set to 16379 rather than 16383 (4 less). The only other interesting case is when PREDIV_A equals 0, PREDIV_S should be set to 32759 rather than 32767 (8 less).

If PREDIV_S is reduced in this way, the formula given the effective frequency of the calibrated input clock is as follows:

$$F_{CAL} = F_{RTCCLK} \times [1 + (256 - CALM) / (2^{20} + CALM - 256)]$$

In this case, CALM[7:0] equals 0x100 (the midpoint of the CALM range) is the correct setting if RTCCLK is exactly 32768.00 Hz.

Verifying the RTC calibration

RTC precision is ensured by measuring the precise frequency of RTCCLK and calculating the correct CALM value and CALP values. An optional 1 Hz output is provided to allow applications to measure and verify the RTC precision.

Measuring the precise frequency of the RTC over a limited interval can result in a measurement error of up to 2 RTCCLK clock cycles over the measurement period, depending on how the digital calibration cycle is aligned with the measurement period.

However, this measurement error can be eliminated if the measurement period is the same length as the calibration cycle period. In this case, the only error observed is the error due to the resolution of the digital calibration.

- By default, the calibration cycle period is 32 seconds.

Using this mode and measuring the accuracy of the 1 Hz output over exactly 32 seconds guarantees that the measure is within 0.477 ppm (0.5 RTCCLK cycles over 32 seconds, due to the limitation of the calibration resolution).

- CALW16 bit of the RTC_CALR register can be set to 1 to force a 16- second calibration cycle period.

In this case, the RTC precision can be measured during 16 seconds with a maximum error of 0.954 ppm (0.5 RTCCLK cycles over 16 seconds). However, since the calibration

resolution is reduced, the long term RTC precision is also reduced to 0.954 ppm: CALM[0] bit is stuck at 0 when CALW16 is set to 1.

- CALW8 bit of the RTC_CALR register can be set to 1 to force a 8- second calibration cycle period.

In this case, the RTC precision can be measured during 8 seconds with a maximum error of 1.907 ppm (0.5 RTCCLK cycles over 8s). The long term RTC precision is also reduced to 1.907 ppm: CALM[1:0] bits are stuck at 00 when CALW8 is set to 1.

Re-calibration on-the-fly

The calibration register (RTC_CALR) can be updated on-the-fly while RTC_ISR/INITF=0, by using the follow process:

1. Poll the RTC_ISR/RECALPF (re-calibration pending flag).
2. If it is set to 0, write a new value to RTC_CALR, if necessary. RECALPF is then automatically set to 1
3. Within three ck_apre cycles after the write operation to RTC_CALR, the new calibration settings take effect.

25.3.13 Time-stamp function

Time-stamp is enabled by setting the TSE or ITSE bits of RTC_CR register to 1.

When TSE is set:

The calendar is saved in the time-stamp registers (RTC_TSSSR, RTC_TSTR, RTC_TSDR) when a time-stamp event is detected on the RTC_TS pin.

When ITSE is set:

The calendar is saved in the time-stamp registers (RTC_TSSSR, RTC_TSTR, RTC_TSDR) when an internal time-stamp event is detected. The internal timestamp event is generated by the switch to the VBAT supply.

When a time-stamp event occurs, due to internal or external event, the time-stamp flag bit (TSF) in RTC_ISR register is set. In case the event is internal, the ITSF flag is also set in RTC_ISR register.

By setting the TSIE bit in the RTC_CR register, an interrupt is generated when a time-stamp event occurs.

If a new time-stamp event is detected while the time-stamp flag (TSF) is already set, the time-stamp overflow flag (TSOVF) flag is set and the time-stamp registers (RTC_TSTR and RTC_TSDR) maintain the results of the previous event.

Note: *TSF is set 2 ck_apre cycles after the time-stamp event occurs due to synchronization process.*

There is no delay in the setting of TSOVF. This means that if two time-stamp events are close together, TSOVF can be seen as '1' while TSF is still '0'. As a consequence, it is recommended to poll TSOVF only after TSF has been set.

Caution: If a time-stamp event occurs immediately after the TSF bit is supposed to be cleared, then both TSF and TSOVF bits are set. To avoid masking a time-stamp event occurring at the same moment, the application must not write '0' into TSF bit unless it has already read it to '1'.

Optionally, a tamper event can cause a time-stamp to be recorded. See the description of the TAMPTS control bit in [Section 25.6.16: RTC tamper configuration register \(RTC_TAMPCR\)](#).

25.3.14 Tamper detection

The RTC_TAMPx input events can be configured either for edge detection, or for level detection with filtering.

The tamper detection can be configured for the following purposes:

- erase the RTC backup registers (default configuration)
- generate an interrupt, capable to wakeup from Stop and Standby modes
- generate a hardware trigger for the low-power timers

RTC backup registers

The backup registers (RTC_BKPxR) are not reset by system reset or when the device wakes up from Standby mode.

The backup registers are reset when a tamper detection event occurs (see [Section 25.6.20: RTC backup registers \(RTC_BKPxR\)](#) and [Tamper detection initialization on page 790](#)) except if the TAMPxNOERASE bit is set, or if TAMPxF is set in the RTC_TAMPCR register.

Tamper detection initialization

Each input can be enabled by setting the corresponding TAMPxE bits to 1 in the RTC_TAMPCR register.

Each RTC_TAMPx tamper detection input is associated with a flag TAMPxF in the RTC_ISR register.

When TAMPxF is cleared:

The TAMPxF flag is asserted after the tamper event on the pin, with the latency provided below:

- 3 ck_apre cycles when TAMPFLT differs from 0x0 (Level detection with filtering)
- 3 ck_apre cycles when TAMPTS=1 (Timestamp on tamper event)
- No latency when TAMPFLT=0x0 (Edge detection) and TAMPTS=0

A new tamper occurring on the same pin during this period and as long as TAMPxF is set cannot be detected.

When TAMPxF is set:

A new tamper occurring on the same pin cannot be detected during the latency described above and 2.5 ck_RTC additional cycles.

By setting the TAMPIE bit in the RTC_TAMPCR register, an interrupt is generated when a tamper detection event occurs (when TAMPxF is set). Setting TAMPIE is not allowed when one or more TAMPxF is set.

When TAMPIE is cleared, each tamper pin event interrupt can be individually enabled by setting the corresponding TAMPxIE bit in the RTC_TAMPCR register. Setting TAMPxIE is not allowed when the corresponding TAMPxF is set.

Trigger output generation on tamper event

The tamper event detection can be used as trigger input by the low-power timers.

When TAMPxMF bit in cleared in RTC_TAMPCR register, the TAMPxF flag must be cleared by software in order to allow a new tamper detection on the same pin.

When TAMPxMF bit is set, the TAMPxF flag is masked, and kept cleared in RTC_ISR register. This configuration allows to trig automatically the low-power timers in Stop mode, without requiring the system wakeup to perform the TAMPxF clearing. In this case, the backup registers are not cleared.

Timestamp on tamper event

With TAMPTS set to '1', any tamper event causes a timestamp to occur. In this case, either the TSF bit or the TSOVF bit are set in RTC_ISR, in the same manner as if a normal timestamp event occurs. The affected tamper flag register TAMPxF is set at the same time that TSF or TSOVF is set.

Edge detection on tamper inputs

If the TAMPFLT bits are "00", the RTC_TAMPx pins generate tamper detection events when either a rising edge or a falling edge is observed depending on the corresponding TAMPxTRG bit. The internal pull-up resistors on the RTC_TAMPx inputs are deactivated when edge detection is selected.

Caution: When using the edge detection, it is recommended to check by software the tamper pin level just after enabling the tamper detection (by reading the GPIO registers), and before writing sensitive values in the backup registers, to ensure that an active edge did not occur before enabling the tamper event detection.

When TAMPFLT="00" and TAMPxTRG = 0 (rising edge detection), a tamper event may be detected by hardware if the tamper input is already at high level before enabling the tamper detection.

After a tamper event has been detected and cleared, the RTC_TAMPx should be disabled and then re-enabled (TAMPxE set to 1) before re-programming the backup registers (RTC_BKPxR). This prevents the application from writing to the backup registers while the RTC_TAMPx input value still indicates a tamper detection. This is equivalent to a level detection on the RTC_TAMPx input.

Note: *Tamper detection is still active when V_{DD} power is switched off. To avoid unwanted resetting of the backup registers, the pin to which the RTC_TAMPx is mapped should be externally tied to the correct level.*

Level detection with filtering on RTC_TAMPx inputs

Level detection with filtering is performed by setting TAMPFLT to a non-zero value. A tamper detection event is generated when either 2, 4, or 8 (depending on TAMPFLT) consecutive samples are observed at the level designated by the TAMPxTRG bits.

The RTC_TAMPx inputs are precharged through the I/O internal pull-up resistance before its state is sampled, unless disabled by setting TAMPPUDIS to 1. The duration of the precharge is determined by the TAMPPRCH bits, allowing for larger capacitances on the RTC_TAMPx inputs.

The trade-off between tamper detection latency and power consumption through the pull-up can be optimized by using TAMPFREQ to determine the frequency of the sampling for level detection.

Note: Refer to the datasheets for the electrical characteristics of the pull-up resistors.

25.3.15 Calibration clock output

When the COE bit is set to 1 in the RTC_CR register, a reference clock is provided on the RTC_CALIB device output.

If the COSEL bit in the RTC_CR register is reset and PREDIV_A = 0x7F, the RTC_CALIB frequency is $f_{RTCCLK}/64$. This corresponds to a calibration output at 512 Hz for an RTCCLK frequency at 32.768 kHz. The RTC_CALIB duty cycle is irregular: there is a light jitter on falling edges. It is therefore recommended to use rising edges.

When COSEL is set and “PREDIV_S+1” is a non-zero multiple of 256 (i.e: PREDIV_S[7:0] = 0xFF), the RTC_CALIB frequency is $f_{RTCCLK}/(256 * (PREDIV_A+1))$. This corresponds to a calibration output at 1 Hz for prescaler default values (PREDIV_A = 0x7F, PREDIV_S = 0xFF), with an RTCCLK frequency at 32.768 kHz. The 1 Hz output is affected when a shift operation is on going and may toggle during the shift operation (SHPF=1).

Note: When the RTC_CALIB or RTC_ALARM output is selected, the RTC_OUT pin is automatically configured as output.

When COSEL bit is cleared, the RTC_CALIB output is the output of the 6th stage of the asynchronous prescaler.

When COSEL bit is set, the RTC_CALIB output is the output of the 8th stage of the synchronous prescaler.

25.3.16 Alarm output

The OSEL[1:0] control bits in the RTC_CR register are used to activate the alarm output RTC_ALARM, and to select the function which is output. These functions reflect the contents of the corresponding flags in the RTC_ISR register.

The polarity of the output is determined by the POL control bit in RTC_CR so that the opposite of the selected flag bit is output when POL is set to 1.

Alarm output

The RTC_ALARM pin can be configured in output open drain or output push-pull using the control bit RTC_ALARM_TYPE in the RTC_OR register.

Note: Once the RTC_ALARM output is enabled, it has priority over RTC_CALIB (COE bit is don't care and must be kept cleared).

When the RTC_CALIB or RTC_ALARM output is selected, the RTC_OUT pin is automatically configured as output.

25.4 RTC low-power modes

Table 141. Effect of low-power modes on RTC

Mode	Description
Sleep	No effect RTC interrupts cause the device to exit the Sleep mode.
Stop	The RTC remains active when the RTC clock source is LSE or LSI. RTC alarm, RTC tamper event, RTC timestamp event, and RTC Wakeup cause the device to exit the Stop mode.
Standby	The RTC remains active when the RTC clock source is LSE or LSI. RTC alarm, RTC tamper event, RTC timestamp event, and RTC Wakeup cause the device to exit the Standby mode.

25.5 RTC interrupts

All RTC interrupts are connected to the EXTI controller. Refer to [Section 11: Extended interrupts and events controller \(EXTI\)](#).

To enable the RTC Alarm interrupt, the following sequence is required:

1. Configure and enable the EXTI line corresponding to the RTC Alarm event in interrupt mode and select the rising edge sensitivity.
2. Configure and enable the RTC_ALARM IRQ channel in the NVIC.
3. Configure the RTC to generate RTC alarms.

To enable the RTC Tamper interrupt, the following sequence is required:

1. Configure and enable the EXTI line corresponding to the RTC Tamper event in interrupt mode and select the rising edge sensitivity.
2. Configure and Enable the RTC_TAMP_STAMP IRQ channel in the NVIC.
3. Configure the RTC to detect the RTC tamper event.

To enable the RTC TimeStamp interrupt, the following sequence is required:

1. Configure and enable the EXTI line corresponding to the RTC TimeStamp event in interrupt mode and select the rising edge sensitivity.
2. Configure and Enable the RTC_TAMP_STAMP IRQ channel in the NVIC.
3. Configure the RTC to detect the RTC time-stamp event.

To enable the Wakeup timer interrupt, the following sequence is required:

1. Configure and enable the EXTI line corresponding to the Wakeup timer even in interrupt mode and select the rising edge sensitivity.
2. Configure and Enable the RTC_WKUP IRQ channel in the NVIC.
3. Configure the RTC to detect the RTC Wakeup timer event.

Table 142. Interrupt control bits

Interrupt event	Event flag	Enable control bit	Exit from Sleep mode	Exit from Stop mode	Exit from Standby mode
Alarm A	ALRAF	ALRAIE	yes	yes ⁽¹⁾	yes ⁽¹⁾
Alarm B	ALRBF	ALRBIE	yes	yes ⁽¹⁾	yes ⁽¹⁾
RTC_TS input (timestamp)	TSF	TSIE	yes	yes ⁽¹⁾	yes ⁽¹⁾
RTC_TAMP1 input detection	TAMP1F	TAMPIE	yes	yes ⁽¹⁾	yes ⁽¹⁾
RTC_TAMP2 input detection	TAMP2F	TAMPIE	yes	yes ⁽¹⁾	yes ⁽¹⁾
RTC_TAMP3 input detection	TAMP3F	TAMPIE	yes	yes ⁽¹⁾	yes ⁽¹⁾
Wakeup timer interrupt	WUTF	WUTIE	yes	yes ⁽¹⁾	yes ⁽¹⁾

1. Wakeup from STOP and Standby modes is possible only when the RTC clock source is LSE or LSI.

25.6 RTC registers

Refer to [Section 1.2 on page 53](#) of the reference manual for a list of abbreviations used in register descriptions.

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

25.6.1 RTC time register (RTC_TR)

The RTC_TR is the calendar time shadow register. This register must be written in initialization mode only. Refer to [Calendar initialization and configuration on page 783](#) and [Reading the calendar on page 784](#).

This register is write protected. The write access procedure is described in [RTC register write protection on page 783](#).

Address offset: 0x00

Backup domain reset value: 0x0000 0000

System reset: 0x0000 0000 when BYPSHAD = 0. Not affected when BYPSHAD = 1.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PM	HT[1:0]		HU[3:0]			
									rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	MNT[2:0]			MNU[3:0]			Res.	ST[2:0]			SU[3:0]				
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31-23 Reserved, must be kept at reset value

Bit 22 **PM**: AM/PM notation

- 0: AM or 24-hour format
- 1: PM

Bits 21:20 **HT[1:0]**: Hour tens in BCD format

Bits 19:16 **HU[3:0]**: Hour units in BCD format

Bit 15 Reserved, must be kept at reset value.

Bits 14:12 **MNT[2:0]**: Minute tens in BCD format

Bits 11:8 **MNU[3:0]**: Minute units in BCD format

Bit 7 Reserved, must be kept at reset value.

Bits 6:4 **ST[2:0]**: Second tens in BCD format

Bits 3:0 **SU[3:0]**: Second units in BCD format

25.6.2 RTC date register (RTC_DR)

The RTC_DR is the calendar date shadow register. This register must be written in initialization mode only. Refer to [Calendar initialization and configuration on page 783](#) and [Reading the calendar on page 784](#).

This register is write protected. The write access procedure is described in [RTC register write protection on page 783](#).

Address offset: 0x04

Backup domain reset value: 0x0000 2101

System reset: 0x0000 2101 when BYPSHAD = 0. Not affected when BYPSHAD = 1.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	YT[3:0]				YU[3:0]			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDU[2:0]				MT	MU[3:0]				Res.	Res.	DT[1:0]		DU[3:0]		
rw	rw	rw	rw	rw	rw	rw	rw				rw	rw	rw	rw	rw

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

Bits 23:20 **YT[3:0]**: Year tens in BCD format

Bits 19:16 **YU[3:0]**: Year units in BCD format

Bits 15:13 **WDU[2:0]**: Week day units

000: forbidden

001: Monday

...

111: Sunday

Bit 12 **MT**: Month tens in BCD format

Bits 11:8 **MU**: Month units in BCD format

Bits 7:6 Reserved, must be kept at reset value.

Bits 5:4 **DT[1:0]**: Date tens in BCD format

Bits 3:0 **DU[3:0]**: Date units in BCD format

25.6.3 RTC control register (RTC_CR)

Address offset: 0x08

Backup domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	ITSE	COE	OSEL[1:0]	POL	COSEL	BKP	SUB1H	ADD1H	
							rw	rw	rw	rw	rw	rw	rw	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TSIE	WUTIE	ALRBIE	ALRAIE	TSE	WUTE	ALRBE	ALRAE	Res.	FMT	BYPS HAD	REFCKON	TSEDGE	WUCKSEL[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

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

Bit 24 **ITSE**: timestamp on internal event enable

- 0: internal event timestamp disabled
- 1: internal event timestamp enabled

Bit 23 **COE**: Calibration output enable

- This bit enables the RTC_CALIB output
- 0: Calibration output disabled
 - 1: Calibration output enabled

Bits 22:21 **OSEL[1:0]**: Output selection

These bits are used to select the flag to be routed to RTC_ALARM output

- 00: Output disabled
- 01: Alarm A output enabled
- 10: Alarm B output enabled
- 11: Wakeup output enabled

Bit 20 **POL**: Output polarity

This bit is used to configure the polarity of RTC_ALARM output

- 0: The pin is high when ALRAF/ALRBF/WUTF is asserted (depending on OSEL[1:0])
- 1: The pin is low when ALRAF/ALRBF/WUTF is asserted (depending on OSEL[1:0]).

Bit 19 **COSEL**: Calibration output selection

When COE=1, this bit selects which signal is output on RTC_CALIB.

- 0: Calibration output is 512 Hz (with default prescaler setting)
- 1: Calibration output is 1 Hz (with default prescaler setting)

These frequencies are valid for RTCCLK at 32.768 kHz and prescalers at their default values (PREDIV_A=127 and PREDIV_S=255). Refer to [Section 25.3.15: Calibration clock output](#)

Bit 18 **BKP**: Backup

This bit can be written by the user to memorize whether the daylight saving time change has been performed or not.

Bit 17 **SUB1H**: Subtract 1 hour (winter time change)

When this bit is set, 1 hour is subtracted to the calendar time if the current hour is not 0. This bit is always read as 0.

Setting this bit has no effect when current hour is 0.

0: No effect

1: Subtracts 1 hour to the current time. This can be used for winter time change outside initialization mode.

Bit 16 **ADD1H**: Add 1 hour (summer time change)

When this bit is set, 1 hour is added to the calendar time. This bit is always read as 0.

0: No effect

1: Adds 1 hour to the current time. This can be used for summer time change outside initialization mode.

Bit 15 **TSIE**: Time-stamp interrupt enable

0: Time-stamp Interrupt disable

1: Time-stamp Interrupt enable

Bit 14 **WUTIE**: Wakeup timer interrupt enable

0: Wakeup timer interrupt disabled

1: Wakeup timer interrupt enabled

Bit 13 **ALRBIE**: Alarm B interrupt enable

0: Alarm B Interrupt disable

1: Alarm B Interrupt enable

Bit 12 **ALRAIE**: Alarm A interrupt enable

0: Alarm A interrupt disabled

1: Alarm A interrupt enabled

Bit 11 **TSE**: timestamp enable

0: timestamp disable

1: timestamp enable

Bit 10 **WUTE**: Wakeup timer enable

0: Wakeup timer disabled

1: Wakeup timer enabled

Note: When the wakeup timer is disabled, wait for WUTWF=1 before enabling it again.

Bit 9 **ALRBE**: Alarm B enable

0: Alarm B disabled

1: Alarm B enabled

Bit 8 **ALRAE**: Alarm A enable

0: Alarm A disabled

1: Alarm A enabled

Bit 7 Reserved, must be kept at reset value.

Bit 6 **FMT**: Hour format

0: 24 hour/day format

1: AM/PM hour format

Bit 5 **BYPSHAD**: Bypass the shadow registers

0: Calendar values (when reading from RTC_SSR, RTC_TR, and RTC_DR) are taken from the shadow registers, which are updated once every two RTCCLK cycles.

1: Calendar values (when reading from RTC_SSR, RTC_TR, and RTC_DR) are taken directly from the calendar counters.

Note: If the frequency of the APB clock is less than seven times the frequency of RTCCLK, BYPSHAD must be set to '1'.

Bit 4 **REFCKON**: RTC_REFIN reference clock detection enable (50 or 60 Hz)

0: RTC_REFIN detection disabled

1: RTC_REFIN detection enabled

Note: PREDIV_S must be 0x00FF.

Bit 3 **TSEDGE**: Time-stamp event active edge

0: RTC_TS input rising edge generates a time-stamp event

1: RTC_TS input falling edge generates a time-stamp event

TSE must be reset when TSEDGE is changed to avoid unwanted TSF setting.

Bits 2:0 **WUCKSEL[2:0]**: Wakeup clock selection

000: RTC/16 clock is selected

001: RTC/8 clock is selected

010: RTC/4 clock is selected

011: RTC/2 clock is selected

10x: ck_spre (usually 1 Hz) clock is selected

11x: ck_spre (usually 1 Hz) clock is selected and 2^{16} is added to the WUT counter value (see note below)

Note: Bits 7, 6 and 4 of this register can be written in initialization mode only (RTC_ISR/INITF = 1).

WUT = Wakeup unit counter value. WUT = (0x0000 to 0xFFFF) + 0x10000 added when WUCKSEL[2:1 = 11].

Bits 2 to 0 of this register can be written only when RTC_CR WUTE bit = 0 and RTC_ISR WUTWF bit = 1.

It is recommended not to change the hour during the calendar hour increment as it could mask the incrementation of the calendar hour.

ADD1H and SUB1H changes are effective in the next second.

This register is write protected. The write access procedure is described in [RTC register write protection on page 783](#).

Caution: TSE must be reset when TSEDGE is changed to avoid spuriously setting of TSF.

25.6.4 RTC initialization and status register (RTC_ISR)

This register is write protected (except for RTC_ISR[13:8] bits). The write access procedure is described in [RTC register write protection on page 783](#).

Address offset: 0x0C

Backup domain reset value: 0x0000 0007

System reset: not affected except INIT, INITF, and RSF bits which are cleared to '0'

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ITSF	RECALPF
														rc_w0	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TAMP3F	TAMP2F	TAMP1F	TSOVF	TSF	WUTF	ALRBF	ALRAF	INIT	INITF	RSF	INITS	SHPF	WUTWF	ALRB WF	ALRAWF
rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rw	r	rc_w0	r	r	r	r	r

Bits 31:18 Reserved, must be kept at reset value

Bit 17 **ITSF**: Internal tTime-stamp flag

This flag is set by hardware when a time-stamp on the internal event occurs.

This flag is cleared by software by writing 0, and must be cleared together with TSF bit by writing 0 in both bits.

Bit 16 **RECALPF**: Recalibration pending Flag

The RECALPF status flag is automatically set to '1' when software writes to the RTC_CALR register, indicating that the RTC_CALR register is blocked. When the new calibration settings are taken into account, this bit returns to '0'. Refer to [Re-calibration on-the-fly](#).

Bit 15 **TAMP3F**: RTC_TAMP3 detection flag

This flag is set by hardware when a tamper detection event is detected on the RTC_TAMP3 input.

It is cleared by software writing 0

Bit 14 **TAMP2F**: RTC_TAMP2 detection flag

This flag is set by hardware when a tamper detection event is detected on the RTC_TAMP2 input.

It is cleared by software writing 0

Bit 13 **TAMP1F**: RTC_TAMP1 detection flag

This flag is set by hardware when a tamper detection event is detected on the RTC_TAMP1 input.

It is cleared by software writing 0

Bit 12 **TSOVF**: Time-stamp overflow flag

This flag is set by hardware when a time-stamp event occurs while TSF is already set.

This flag is cleared by software by writing 0. It is recommended to check and then clear TSOVF only after clearing the TSF bit. Otherwise, an overflow might not be noticed if a time-stamp event occurs immediately before the TSF bit is cleared.

Bit 11 **TSF**: Time-stamp flag

This flag is set by hardware when a time-stamp event occurs.

This flag is cleared by software by writing 0. If ITSF flag is set, TSF must be cleared together with ITSF by writing 0 in both bits.

Bit 10 WUTF: Wakeup timer flag

This flag is set by hardware when the wakeup auto-reload counter reaches 0.

This flag is cleared by software by writing 0.

This flag must be cleared by software at least 1.5 RTCCLK periods before WUTF is set to 1 again.

Bit 9 ALRBF: Alarm B flag

This flag is set by hardware when the time/date registers (RTC_TR and RTC_DR) match the Alarm B register (RTC_ALRMBR).

This flag is cleared by software by writing 0.

Bit 8 ALRAF: Alarm A flag

This flag is set by hardware when the time/date registers (RTC_TR and RTC_DR) match the Alarm A register (RTC_ALRMAR).

This flag is cleared by software by writing 0.

Bit 7 INIT: Initialization mode

0: Free running mode

1: Initialization mode used to program time and date register (RTC_TR and RTC_DR), and prescaler register (RTC_PRER). Counters are stopped and start counting from the new value when INIT is reset.

Bit 6 INITF: Initialization flag

When this bit is set to 1, the RTC is in initialization state, and the time, date and prescaler registers can be updated.

0: Calendar registers update is not allowed

1: Calendar registers update is allowed

Bit 5 RSF: Registers synchronization flag

This bit is set by hardware each time the calendar registers are copied into the shadow registers (RTC_SSRx, RTC_TRx and RTC_DRx). This bit is cleared by hardware in initialization mode, while a shift operation is pending (SHPF=1), or when in bypass shadow register mode (BYPSHAD=1). This bit can also be cleared by software.

It is cleared either by software or by hardware in initialization mode.

0: Calendar shadow registers not yet synchronized

1: Calendar shadow registers synchronized

Bit 4 INITS: Initialization status flag

This bit is set by hardware when the calendar year field is different from 0 (Backup domain reset state).

0: Calendar has not been initialized

1: Calendar has been initialized

Bit 3 SHPF: Shift operation pending

0: No shift operation is pending

1: A shift operation is pending

This flag is set by hardware as soon as a shift operation is initiated by a write to the RTC_SHIFTR register. It is cleared by hardware when the corresponding shift operation has been executed. Writing to the SHPF bit has no effect.

Bit 2 WUTWF: Wakeup timer write flag

This bit is set by hardware up to 2 RTCCLK cycles after the WUTE bit has been set to 0 in RTC_CR, and is cleared up to 2 RTCCLK cycles after the WUTE bit has been set to 1. The wakeup timer values can be changed when WUTE bit is cleared and WUTWF is set.

- 0: Wakeup timer configuration update not allowed
- 1: Wakeup timer configuration update allowed

Bit 1 ALRBWF: Alarm B write flag

This bit is set by hardware when Alarm B values can be changed, after the ALRBE bit has been set to 0 in RTC_CR.

It is cleared by hardware in initialization mode.

- 0: Alarm B update not allowed
- 1: Alarm B update allowed

Bit 0 ALRAWF: Alarm A write flag

This bit is set by hardware when Alarm A values can be changed, after the ALRAE bit has been set to 0 in RTC_CR.

It is cleared by hardware in initialization mode.

- 0: Alarm A update not allowed
- 1: Alarm A update allowed

Note: *The bits ALRAF, ALRBF, WUTF and TSF are cleared 2 APB clock cycles after programming them to 0.*

25.6.5 RTC prescaler register (RTC_PRER)

This register must be written in initialization mode only. The initialization must be performed in two separate write accesses. Refer to [Calendar initialization and configuration on page 783](#).

This register is write protected. The write access procedure is described in [RTC register write protection on page 783](#).

Address offset: 0x10

Backup domain reset value: 0x007F 00FF

System reset: not affected

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

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

Bits 22:16 **PREDIV_A[6:0]**: Asynchronous prescaler factor

This is the asynchronous division factor:

$$\text{ck_apre frequency} = \text{RTCCLK frequency}/(\text{PREDIV_A}+1)$$

Bit 15 Reserved, must be kept at reset value.

Bits 14:0 **PREDIV_S[14:0]**: Synchronous prescaler factor

This is the synchronous division factor:

$$\text{ck_spre frequency} = \text{ck_apre frequency}/(\text{PREDIV_S}+1)$$

25.6.6 RTC wakeup timer register (RTC_WUTR)

This register can be written only when WUTWF is set to 1 in RTC_ISR.

This register is write protected. The write access procedure is described in [RTC register write protection on page 783](#).

Address offset: 0x14

Backup domain reset value: 0x0000 FFFF

System reset: not affected

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

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

Bits 15:0 **WUT[15:0]**: Wakeup auto-reload value bits

When the wakeup timer is enabled (WUTE set to 1), the WUTF flag is set every (WUT[15:0] + 1) ck_wut cycles. The ck_wut period is selected through WUCKSEL[2:0] bits of the RTC_CR register

When WUCKSEL[2] = 1, the wakeup timer becomes 17-bits and WUCKSEL[1] effectively becomes WUT[16] the most-significant bit to be reloaded into the timer.

The first assertion of WUTF occurs (WUT+1) ck_wut cycles after WUTE is set. Setting WUT[15:0] to 0x0000 with WUCKSEL[2:0] =011 (RTCCLK/2) is forbidden.

25.6.7 RTC alarm A register (RTC_ALRMAR)

This register can be written only when ALRAWF is set to 1 in RTC_ISR, or in initialization mode.

This register is write protected. The write access procedure is described in [RTC register write protection on page 783](#).

Address offset: 0x1C

Backup domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MSK4	WDSEL	DT[1:0]		DU[3:0]				MSK3	PM	HT[1:0]		HU[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSK2	MNT[2:0]			MNU[3:0]				MSK1	ST[2:0]			SU[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **MSK4**: Alarm A date mask

- 0: Alarm A set if the date/day match
- 1: Date/day don't care in Alarm A comparison

Bit 30 **WDSEL**: Week day selection

- 0: DU[3:0] represents the date units
- 1: DU[3:0] represents the week day. DT[1:0] is don't care.

Bits 29:28 **DT[1:0]**: Date tens in BCD format.

Bits 27:24 **DU[3:0]**: Date units or day in BCD format.

Bit 23 **MSK3**: Alarm A hours mask

- 0: Alarm A set if the hours match
- 1: Hours don't care in Alarm A comparison

Bit 22 **PM**: AM/PM notation

- 0: AM or 24-hour format
- 1: PM

Bits 21:20 **HT[1:0]**: Hour tens in BCD format.

Bits 19:16 **HU[3:0]**: Hour units in BCD format.

Bit 15 **MSK2**: Alarm A minutes mask

- 0: Alarm A set if the minutes match
- 1: Minutes don't care in Alarm A comparison

Bits 14:12 **MNT[2:0]**: Minute tens in BCD format.

Bits 11:8 **MNU[3:0]**: Minute units in BCD format.

Bit 7 **MSK1**: Alarm A seconds mask

- 0: Alarm A set if the seconds match
- 1: Seconds don't care in Alarm A comparison

Bits 6:4 **ST[2:0]**: Second tens in BCD format.

Bits 3:0 **SU[3:0]**: Second units in BCD format.

25.6.8 RTC alarm B register (RTC_ALRMBR)

This register can be written only when ALRBWF is set to 1 in RTC_ISR, or in initialization mode.

This register is write protected. The write access procedure is described in [RTC register write protection on page 783](#).

Address offset: 0x20

Backup domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MSK4	WDSEL	DT[1:0]		DU[3:0]				MSK3	PM	HT[1:0]		HU[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSK2	MNT[2:0]			MNU[3:0]				MSK1	ST[2:0]			SU[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **MSK4**: Alarm B date mask

- 0: Alarm B set if the date and day match
- 1: Date and day don't care in Alarm B comparison

Bit 30 **WDSEL**: Week day selection

- 0: DU[3:0] represents the date units
- 1: DU[3:0] represents the week day. DT[1:0] is don't care.

Bits 29:28 **DT[1:0]**: Date tens in BCD format

Bits 27:24 **DU[3:0]**: Date units or day in BCD format

Bit 23 **MSK3**: Alarm B hours mask

- 0: Alarm B set if the hours match
- 1: Hours don't care in Alarm B comparison

Bit 22 **PM**: AM/PM notation

- 0: AM or 24-hour format
- 1: PM

Bits 21:20 **HT[1:0]**: Hour tens in BCD format

Bits 19:16 **HU[3:0]**: Hour units in BCD format

Bit 15 **MSK2**: Alarm B minutes mask

- 0: Alarm B set if the minutes match
- 1: Minutes don't care in Alarm B comparison

Bits 14:12 **MNT[2:0]**: Minute tens in BCD format

Bits 11:8 **MNU[3:0]**: Minute units in BCD format

Bit 7 **MSK1**: Alarm B seconds mask

- 0: Alarm B set if the seconds match
- 1: Seconds don't care in Alarm B comparison

Bits 6:4 **ST[2:0]**: Second tens in BCD format

Bits 3:0 **SU[3:0]**: Second units in BCD format

25.6.9 RTC write protection register (RTC_WPR)

Address offset: 0x24

Reset value: 0x0000 0000

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

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

Bits 7:0 **KEY**: Write protection key

This byte is written by software.

Reading this byte always returns 0x00.

Refer to [RTC register write protection](#) for a description of how to unlock RTC register write protection.

25.6.10 RTC sub second register (RTC_SSR)

Address offset: 0x28

Backup domain reset value: 0x0000 0000

System reset: 0x0000 0000 when BYPSHAD = 0. Not affected when BYPSHAD = 1.

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

Bits31:16 Reserved, must be kept at reset value

Bits 15:0 **SS**: Sub second value

SS[15:0] is the value in the synchronous prescaler counter. The fraction of a second is given by the formula below:

Second fraction = (PREDIV_S - SS) / (PREDIV_S + 1)

Note: SS can be larger than PREDIV_S only after a shift operation. In that case, the correct time/date is one second less than as indicated by RTC_TR/RTC_DR.

25.6.11 RTC shift control register (RTC_SHIFTR)

This register is write protected. The write access procedure is described in [RTC register write protection on page 783](#).

Address offset: 0x2C

Backup domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADD1S	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
w															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SUBFS[14:0]														
	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bit 31 **ADD1S**: Add one second

0: No effect

1: Add one second to the clock/calendar

This bit is write only and is always read as zero. Writing to this bit has no effect when a shift operation is pending (when SHPF=1, in RTC_ISR).

This function is intended to be used with SUBFS (see description below) in order to effectively add a fraction of a second to the clock in an atomic operation.

Bits 30:15 Reserved, must be kept at reset value

Bits 14:0 **SUBFS**: Subtract a fraction of a second

These bits are write only and is always read as zero. Writing to this bit has no effect when a shift operation is pending (when SHPF=1, in RTC_ISR).

The value which is written to SUBFS is added to the synchronous prescaler counter. Since this counter counts down, this operation effectively subtracts from (delays) the clock by:

Delay (seconds) = SUBFS / (PREDIV_S + 1)

A fraction of a second can effectively be added to the clock (advancing the clock) when the ADD1S function is used in conjunction with SUBFS, effectively advancing the clock by:

Advance (seconds) = (1 - (SUBFS / (PREDIV_S + 1))).

Note: Writing to SUBFS causes RSF to be cleared. Software can then wait until RSF=1 to be sure that the shadow registers have been updated with the shifted time.

25.6.12 RTC timestamp time register (RTC_TSTR)

The content of this register is valid only when TSF is set to 1 in RTC_ISR. It is cleared when TSF bit is reset.

Address offset: 0x30

Backup domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PM	HT[1:0]		HU[3:0]			
									r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	MNT[2:0]			MNU[3:0]				Res.	ST[2:0]			SU[3:0]			
	r	r	r	r	r	r	r		r	r	r	r	r	r	r

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

Bit 22 **PM**: AM/PM notation

0: AM or 24-hour format

1: PM

Bits 21:20 **HT[1:0]**: Hour tens in BCD format.

Bits 19:16 **HU[3:0]**: Hour units in BCD format.

Bit 15 Reserved, must be kept at reset value

Bits 14:12 **MNT[2:0]**: Minute tens in BCD format.

Bits 11:8 **MNU[3:0]**: Minute units in BCD format.

Bit 7 Reserved, must be kept at reset value

Bits 6:4 **ST[2:0]**: Second tens in BCD format.

Bits 3:0 **SU[3:0]**: Second units in BCD format.

25.6.13 RTC timestamp date register (RTC_TSDR)

The content of this register is valid only when TSF is set to 1 in RTC_ISR. It is cleared when TSF bit is reset.

Address offset: 0x34

Backup domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDU[1:0]			MT	MU[3:0]				Res.	Res.	DT[1:0]		DU[3:0]			
r	r	r	r	r	r	r	r			r	r	r	r	r	r

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

Bits 15:13 **WDU[1:0]**: Week day units

Bit 12 **MT**: Month tens in BCD format

Bits 11:8 **MU[3:0]**: Month units in BCD format

Bits 7:6 Reserved, must be kept at reset value

Bits 5:4 **DT[1:0]**: Date tens in BCD format

Bits 3:0 **DU[3:0]**: Date units in BCD format

25.6.14 RTC time-stamp sub second register (RTC_TSSSR)

The content of this register is valid only when RTC_ISR/TSF is set. It is cleared when the RTC_ISR/TSF bit is reset.

Address offset: 0x38

Backup domain reset value: 0x0000 0000

System reset: not affected

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

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

Bits 15:0 **SS**: Sub second value

SS[15:0] is the value of the synchronous prescaler counter when the timestamp event occurred.

25.6.15 RTC calibration register (RTC_CALR)

This register is write protected. The write access procedure is described in [RTC register write protection on page 783](#).

Address offset: 0x3C

Backup domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CALP	CALW8	CALW16	Res.	Res.	Res.	Res.	CALM[8:0]								
rw	rw	rw					rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bit 15 **CALP**: Increase frequency of RTC by 488.5 ppm

0: No RTCCLK pulses are added.

1: One RTCCLK pulse is effectively inserted every 2^{11} pulses (frequency increased by 488.5 ppm).

This feature is intended to be used in conjunction with CALM, which lowers the frequency of the calendar with a fine resolution. If the input frequency is 32768 Hz, the number of RTCCLK pulses added during a 32-second window is calculated as follows: $(512 * \text{CALP}) - \text{CALM}$.

Refer to [Section 25.3.12: RTC smooth digital calibration](#).

Bit 14 **CALW8**: Use an 8-second calibration cycle period

When CALW8 is set to '1', the 8-second calibration cycle period is selected.

Note: CALM[1:0] are stuck at "00" when CALW8='1'. Refer to [Section 25.3.12: RTC smooth digital calibration](#).

Bit 13 **CALW16**: Use a 16-second calibration cycle period

When CALW16 is set to '1', the 16-second calibration cycle period is selected. This bit must not be set to '1' if CALW8=1.

Note: CALM[0] is stuck at '0' when CALW16='1'. Refer to [Section 25.3.12: RTC smooth digital calibration](#).

Bits 12:9 Reserved, must be kept at reset value

Bits 8:0 **CALM[8:0]**: Calibration minus

The frequency of the calendar is reduced by masking CALM out of 2^{20} RTCCLK pulses (32 seconds if the input frequency is 32768 Hz). This decreases the frequency of the calendar with a resolution of 0.9537 ppm.

To increase the frequency of the calendar, this feature should be used in conjunction with CALP. See [Section 25.3.12: RTC smooth digital calibration on page 787](#).

25.6.16 RTC tamper configuration register (RTC_TAMPCCR)

Address offset: 0x40

Backup domain reset value: 0x0000 0000

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	TAMP3 MF	TAMP3 NO ERASE	TAMP3 IE	TAMP2 MF	TAMP2 NO ERASE	TAMP2 IE	TAMP1 MF	TAMP1 NO ERASE	TAMP1 IE
							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
TAMP PUDIS	TAMP PRCH [1:0]	TAMP FLT[1:0]	TAMP FREQ[2:0]	TAMP TS	TAMP3 TRG	TAMP3 E	TAMP2 TRG	TAMP2 E	TAMP1 TRG	TAMP1 E	TAMP1 E				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bit 24 **TAMP3MF**: Tamper 3 mask flag

0: Tamper 3 event generates a trigger event and TAMP3F must be cleared by software to allow next tamper event detection.

1: Tamper 3 event generates a trigger event. TAMP3F is masked and internally cleared by hardware. The backup registers are not erased.

Note: The Tamper 3 interrupt must not be enabled when TAMP3MF is set.

Bit 23 **TAMP3NOERASE**: Tamper 3 no erase

0: Tamper 3 event erases the backup registers.

1: Tamper 3 event does not erase the backup registers.

Bit 22 **TAMP3IE**: Tamper 3 interrupt enable

0: Tamper 3 interrupt is disabled if TAMPIE = 0.

1: Tamper 3 interrupt enabled.

Bit 21 **TAMP2MF**: Tamper 2 mask flag

0: Tamper 2 event generates a trigger event and TAMP2F must be cleared by software to allow next tamper event detection.

1: Tamper 2 event generates a trigger event. TAMP2F is masked and internally cleared by hardware. The backup registers are not erased.

Note: The Tamper 2 interrupt must not be enabled when TAMP2MF is set.

Bit 20 **TAMP2NOERASE**: Tamper 2 no erase

0: Tamper 2 event erases the backup registers.

1: Tamper 2 event does not erase the backup registers.

Bit 19 **TAMP2IE**: Tamper 2 interrupt enable

0: Tamper 2 interrupt is disabled if TAMPIE = 0.

1: Tamper 2 interrupt enabled.

Bit 18 **TAMP1MF**: Tamper 1 mask flag

0: Tamper 1 event generates a trigger event and TAMP1F must be cleared by software to allow next tamper event detection.

1: Tamper 1 event generates a trigger event. TAMP1F is masked and internally cleared by hardware. The backup registers are not erased.

Note: The Tamper 1 interrupt must not be enabled when TAMP1MF is set.

- Bit 17 **TAMP1NOERASE**: Tamper 1 no erase
 0: Tamper 1 event erases the backup registers.
 1: Tamper 1 event does not erase the backup registers.
- Bit 16 **TAMP1IE**: Tamper 1 interrupt enable
 0: Tamper 1 interrupt is disabled if TAMP1IE = 0.
 1: Tamper 1 interrupt enabled.
- Bit 15 **TAMPPUDIS**: RTC_TAMPx pull-up disable
 This bit determines if each of the RTC_TAMPx pins are precharged before each sample.
 0: Precharge RTC_TAMPx pins before sampling (enable internal pull-up)
 1: Disable precharge of RTC_TAMPx pins.
- Bits 14:13 **TAMPPRCH[1:0]**: RTC_TAMPx precharge duration
 These bit determines the duration of time during which the pull-up/is activated before each sample. TAMPPRCH is valid for each of the RTC_TAMPx inputs.
 0x0: 1 RTCCLK cycle
 0x1: 2 RTCCLK cycles
 0x2: 4 RTCCLK cycles
 0x3: 8 RTCCLK cycles
- Bits 12:11 **TAMPFLT[1:0]**: RTC_TAMPx filter count
 These bits determines the number of consecutive samples at the specified level (TAMP*TRG) needed to activate a Tamper event. TAMPFLT is valid for each of the RTC_TAMPx inputs.
 0x0: Tamper event is activated on edge of RTC_TAMPx input transitions to the active level (no internal pull-up on RTC_TAMPx input).
 0x1: Tamper event is activated after 2 consecutive samples at the active level.
 0x2: Tamper event is activated after 4 consecutive samples at the active level.
 0x3: Tamper event is activated after 8 consecutive samples at the active level.
- Bits 10:8 **TAMPFREQ[2:0]**: Tamper sampling frequency
 Determines the frequency at which each of the RTC_TAMPx inputs are sampled.
 0x0: RTCCLK / 32768 (1 Hz when RTCCLK = 32768 Hz)
 0x1: RTCCLK / 16384 (2 Hz when RTCCLK = 32768 Hz)
 0x2: RTCCLK / 8192 (4 Hz when RTCCLK = 32768 Hz)
 0x3: RTCCLK / 4096 (8 Hz when RTCCLK = 32768 Hz)
 0x4: RTCCLK / 2048 (16 Hz when RTCCLK = 32768 Hz)
 0x5: RTCCLK / 1024 (32 Hz when RTCCLK = 32768 Hz)
 0x6: RTCCLK / 512 (64 Hz when RTCCLK = 32768 Hz)
 0x7: RTCCLK / 256 (128 Hz when RTCCLK = 32768 Hz)
- Bit 7 **TAMPTS**: Activate timestamp on tamper detection event
 0: Tamper detection event does not cause a timestamp to be saved
 1: Save timestamp on tamper detection event
 TAMPTS is valid even if TSE=0 in the RTC_CR register.
- Bit 6 **TAMP3TRG**: Active level for RTC_TAMP3 input
 if TAMPFLT ≠ 00:
 0: RTC_TAMP3 input staying low triggers a tamper detection event.
 1: RTC_TAMP3 input staying high triggers a tamper detection event.
 if TAMPFLT = 00:
 0: RTC_TAMP3 input rising edge triggers a tamper detection event.
 1: RTC_TAMP3 input falling edge triggers a tamper detection event.

Note: The Tamper 3 falling edge detection is not allowed when switch to VBAT is used, otherwise a detection would always occurs when entering in Vbat mode.

Bit 5 **TAMP3E**: RTC_TAMP3 detection enable

- 0: RTC_TAMP3 input detection disabled
- 1: RTC_TAMP3 input detection enabled

Bit 4 **TAMP2TRG**: Active level for RTC_TAMP2 input

if TAMPFLT != 00:

- 0: RTC_TAMP2 input staying low triggers a tamper detection event.
 - 1: RTC_TAMP2 input staying high triggers a tamper detection event.
- if TAMPFLT = 00:
- 0: RTC_TAMP2 input rising edge triggers a tamper detection event.
 - 1: RTC_TAMP2 input falling edge triggers a tamper detection event.

Bit 3 **TAMP2E**: RTC_TAMP2 input detection enable

- 0: RTC_TAMP2 detection disabled
- 1: RTC_TAMP2 detection enabled

Bit 2 **TAMPIE**: Tamper interrupt enable

- 0: Tamper interrupt disabled
- 1: Tamper interrupt enabled.

Note: This bit enables the interrupt for all tamper pins events, whatever TAMPxIE level. If this bit is cleared, each tamper event interrupt can be individually enabled by setting TAMPxIE.

Bit 1 **TAMP1TRG**: Active level for RTC_TAMP1 input

If TAMPFLT != 00

- 0: RTC_TAMP1 input staying low triggers a tamper detection event.
- 1: RTC_TAMP1 input staying high triggers a tamper detection event.

if TAMPFLT = 00:

- 0: RTC_TAMP1 input rising edge triggers a tamper detection event.
- 1: RTC_TAMP1 input falling edge triggers a tamper detection event.

Bit 0 **TAMP1E**: RTC_TAMP1 input detection enable

- 0: RTC_TAMP1 detection disabled
- 1: RTC_TAMP1 detection enabled

Caution: When TAMPFLT = 0, TAMPxE must be reset when TAMPxTRG is changed to avoid spuriously setting TAMxF.

25.6.17 RTC alarm A sub second register (RTC_ALRMASSR)

This register can be written only when ALRAE is reset in RTC_CR register, or in initialization mode.

This register is write protected. The write access procedure is described in [RTC register write protection on page 783](#)

Address offset: 0x44

Backup domain reset value: 0x0000 0000

System reset: not affected

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

Bits 31:28 Reserved, must be kept at reset value.

Bits 27:24 MASKSS[3:0]: Mask the most-significant bits starting at this bit

0: No comparison on sub seconds for Alarm A. The alarm is set when the seconds unit is incremented (assuming that the rest of the fields match).

1: SS[14:1] are don't care in Alarm A comparison. Only SS[0] is compared.

2: SS[14:2] are don't care in Alarm A comparison. Only SS[1:0] are compared.

3: SS[14:3] are don't care in Alarm A comparison. Only SS[2:0] are compared.

...

12: SS[14:12] are don't care in Alarm A comparison. SS[11:0] are compared.

13: SS[14:13] are don't care in Alarm A comparison. SS[12:0] are compared.

14: SS[14] is don't care in Alarm A comparison. SS[13:0] are compared.

15: All 15 SS bits are compared and must match to activate alarm.

The overflow bits of the synchronous counter (bits 15) is never compared. This bit can be different from 0 only after a shift operation.

Bits 23:15 Reserved, must be kept at reset value.

Bits 14:0 SS[14:0]: Sub seconds value

This value is compared with the contents of the synchronous prescaler counter to determine if Alarm A is to be activated. Only bits 0 up MASKSS-1 are compared.

25.6.18 RTC alarm B sub second register (RTC_ALRMBSSR)

This register can be written only when ALRBE is reset in RTC_CR register, or in initialization mode.

This register is write protected. The write access procedure is described in [Section : RTC register write protection](#).

Address offset: 0x48

Backup domain reset value: 0x0000 0000

System reset: not affected

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

Bits 31:28 Reserved, must be kept at reset value.

Bits 27:24 **MASKSS[3:0]**: Mask the most-significant bits starting at this bit

0x0: No comparison on sub seconds for Alarm B. The alarm is set when the seconds unit is incremented (assuming that the rest of the fields match).

0x1: SS[14:1] are don't care in Alarm B comparison. Only SS[0] is compared.

0x2: SS[14:2] are don't care in Alarm B comparison. Only SS[1:0] are compared.

0x3: SS[14:3] are don't care in Alarm B comparison. Only SS[2:0] are compared.

...

0xC: SS[14:12] are don't care in Alarm B comparison. SS[11:0] are compared.

0xD: SS[14:13] are don't care in Alarm B comparison. SS[12:0] are compared.

0xE: SS[14] is don't care in Alarm B comparison. SS[13:0] are compared.

0xF: All 15 SS bits are compared and must match to activate alarm.

The overflow bits of the synchronous counter (bits 15) is never compared. This bit can be different from 0 only after a shift operation.

Bits 23:15 Reserved, must be kept at reset value.

Bits 14:0 **SS[14:0]**: Sub seconds value

This value is compared with the contents of the synchronous prescaler counter to determine if Alarm B is to be activated. Only bits 0 up to MASKSS-1 are compared.

25.6.19 RTC option register (RTC_OR)

Address offset: 0x4C

Backup domain reset value: 0x0000 0000

System reset: not affected

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

Bits 31:4 Reserved, must be kept at reset value.

Bit 3 **RTC_ALARM_TYPE**: RTC_ALARM on PC13 output type

0: RTC_ALARM, when mapped on PC13, is open-drain output

1: RTC_ALARM, when mapped on PC13, is push-pull output

Bits 2:1 **TSINSEL[1:0]**: TIMESTAMP mapping

00: TIMESTAMP is mapped on PC13

01: TIMESTAMP is mapped on PI8

10: TIMESTAMP is mapped on PC1

11: TIMESTAMP is mapped on PC1

Bit 0 Reserved, must be kept at reset value.

25.6.20 RTC backup registers (RTC_BKPxR)

Address offset: 0x50 to 0xCC

Backup domain reset value: 0x0000 0000

System reset: not affected

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

Bits 31:0 BKP[31:0]

The application can write or read data to and from these registers.

They are powered-on by V_{BAT} when V_{DD} is switched off, so that they are not reset by System reset, and their contents remain valid when the device operates in low-power mode.

This register is reset on a tamper detection event, as long as TAMPxF=1.

25.6.21 RTC register map

Table 143. RTC register map and reset values

Table 143. RTC register map and reset values (continued)

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

26 Inter-integrated circuit (I2C) interface

26.1 Introduction

The I²C (inter-integrated circuit) bus interface handles communications between the microcontroller and the serial I²C bus. It provides multimaster capability, and controls all I²C bus-specific sequencing, protocol, arbitration and timing. It supports Standard-mode (Sm), Fast-mode (Fm) and Fast-mode Plus (Fm+).

It is also SMBus (system management bus) and PMBus (power management bus) compatible.

DMA can be used to reduce CPU overload.

26.2 I2C main features

- I²C bus specification rev03 compatibility:
 - Slave and master modes
 - Multimaster capability
 - Standard-mode (up to 100 kHz)
 - Fast-mode (up to 400 kHz)
 - Fast-mode Plus (up to 1 MHz)
 - 7-bit and 10-bit addressing mode
 - Multiple 7-bit slave addresses (2 addresses, 1 with configurable mask)
 - All 7-bit addresses acknowledge mode
 - General call
 - Programmable setup and hold times
 - Easy to use event management
 - Optional clock stretching
 - Software reset
- 1-byte buffer with DMA capability
- Programmable analog and digital noise filters

The following additional features are also available depending on the product implementation (see [Section 26.3: I2C implementation](#)):

- SMBus specification rev 3.0 compatibility:
 - Hardware PEC (Packet Error Checking) generation and verification with ACK control
 - Command and data acknowledge control
 - Address resolution protocol (ARP) support
 - Host and Device support
 - SMBus alert
 - Timeouts and idle condition detection
- PMBus rev 1.3 standard compatibility
- Independent clock: a choice of independent clock sources allowing the I2C communication speed to be independent from the PCLK reprogramming

26.3 I2C implementation

This manual describes the full set of features implemented in I2C1, I2C2 and I2C3. In the STM32F72xxx and STM32F73xxx devices I2C1, I2C2 and I2C3 are identical and implement the full set of features as shown in the following table.

Table 144. STM32F72xxx and STM32F73xxx I2C implementation

I2C features ⁽¹⁾	I2C1	I2C2	I2C3
7-bit addressing mode	X	X	X
10-bit addressing mode	X	X	X
Standard-mode (up to 100 kbit/s)	X	X	X
Fast-mode (up to 400 kbit/s)	X	X	X
Fast-mode Plus with 20mA output drive I/Os (up to 1 Mbit/s)	X	X	X
Independent clock	X	X	X
Wakeup from Stop mode	-	-	-
SMBus/PMBus	X	X	X

1. X = supported.

26.4 I2C functional description

In addition to receiving and transmitting data, this interface converts it from serial to parallel format and vice versa. The interrupts are enabled or disabled by software. The interface is connected to the I²C bus by a data pin (SDA) and by a clock pin (SCL). It can be connected with a standard (up to 100 kHz), Fast-mode (up to 400 kHz) or Fast-mode Plus (up to 1 MHz) I²C bus.

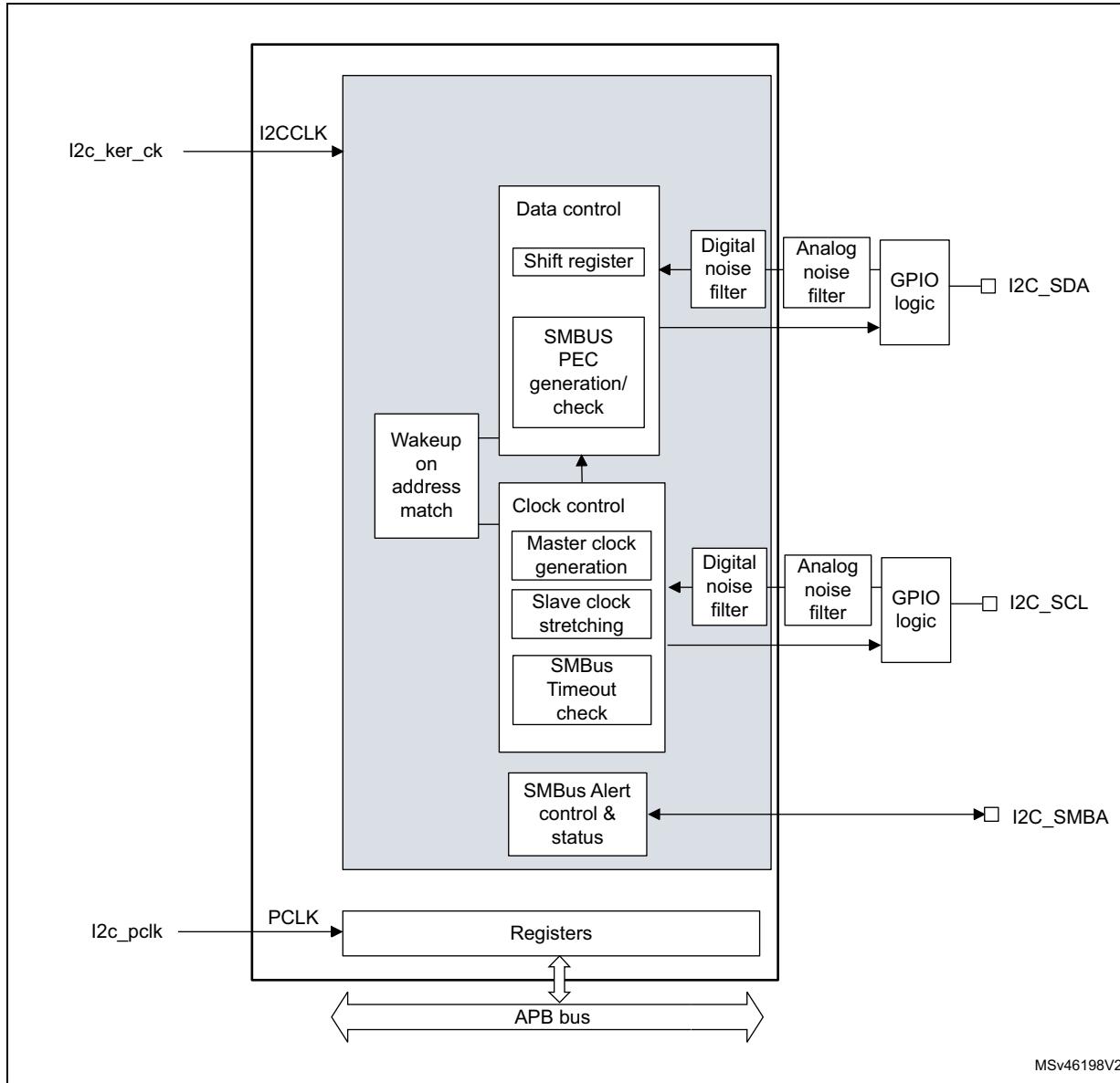
This interface can also be connected to a SMBus with the data pin (SDA) and clock pin (SCL).

If SMBus feature is supported: the additional optional SMBus Alert pin (SMBA) is also available.

26.4.1 I2C block diagram

The block diagram of the I2C interface is shown in [Figure 287](#).

Figure 287. I2C block diagram



The I₂C is clocked by an independent clock source which allows the I₂C to operate independently from the PCLK frequency.

For I₂C I/Os supporting 20 mA output current drive for Fast-mode Plus operation, the driving capability is enabled through control bits in the system configuration controller (SYSCFG). Refer to [Section 26.3: I₂C implementation](#).

26.4.2 I2C clock requirements

The I₂C kernel is clocked by I₂CCLK.

The I₂CCLK period t_{I2CCLK} must respect the following conditions:

$$t_{I2CCLK} < (t_{LOW} - t_{filters}) / 4 \text{ and } t_{I2CCLK} < t_{HIGH}$$

with:

t_{LOW} : SCL low time and t_{HIGH} : SCL high time

$t_{filters}$: when enabled, sum of the delays brought by the analog filter and by the digital filter.

Analog filter delay is maximum 260 ns. Digital filter delay is DNF x t_{I2CCLK} .

The PCLK clock period t_{PCLK} must respect the following condition:

$$t_{PCLK} < 4/3 t_{SCL}$$

with t_{SCL} : SCL period

Caution: When the I₂C kernel is clocked by PCLK, this clock must respect the conditions for t_{I2CCLK} .

26.4.3 Mode selection

The interface can operate in one of the four following modes:

- Slave transmitter
- Slave receiver
- Master transmitter
- Master receiver

By default, it operates in slave mode. The interface automatically switches from slave to master when it generates a START condition, and from master to slave if an arbitration loss or a STOP generation occurs, allowing multimaster capability.

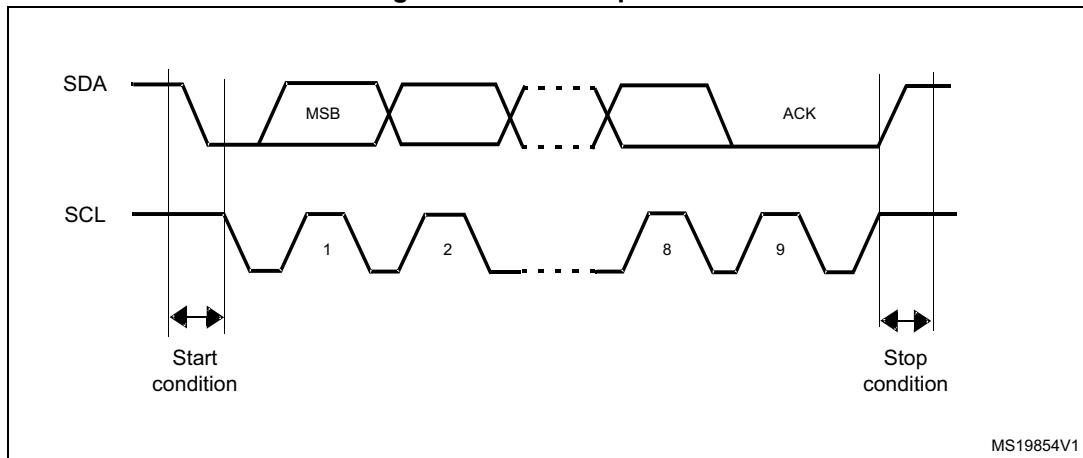
Communication flow

In Master mode, the I₂C interface initiates a data transfer and generates the clock signal. A serial data transfer always begins with a START condition and ends with a STOP condition. Both START and STOP conditions are generated in master mode by software.

In Slave mode, the interface is capable of recognizing its own addresses (7 or 10-bit), and the General Call address. The General Call address detection can be enabled or disabled by software. The reserved SMBus addresses can also be enabled by software.

Data and addresses are transferred as 8-bit bytes, MSB first. The first byte(s) following the START condition contain the address (one in 7-bit mode, two in 10-bit mode). The address is always transmitted in Master mode.

A 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 288. I²C bus protocol

Acknowledge can be enabled or disabled by software. The I²C interface addresses can be selected by software.

26.4.4 I²C initialization

Enabling and disabling the peripheral

The I²C peripheral clock must be configured and enabled in the clock controller.

Then the I²C can be enabled by setting the PE bit in the I²C_CR1 register.

When the I²C is disabled (PE=0), the I²C performs a software reset. Refer to [Section 26.4.5: Software reset](#) for more details.

Noise filters

Before enabling the I²C peripheral by setting the PE bit in I²C_CR1 register, the user must configure the noise filters, if needed. By default, an analog noise filter is present on the SDA and SCL inputs. This analog filter is compliant with the I²C specification which requires the suppression of spikes with a pulse width up to 50 ns in Fast-mode and Fast-mode Plus. The user can disable this analog filter by setting the ANFOFF bit, and/or select a digital filter by configuring the DNF[3:0] bit in the I²C_CR1 register.

When the digital filter is enabled, the level of the SCL or the SDA line is internally changed only if it remains stable for more than DNF x I²CCLK periods. This allows to suppress spikes with a programmable length of 1 to 15 I²CCLK periods.

Table 145. Comparison of analog vs. digital filters

-	Analog filter	Digital filter
Pulse width of suppressed spikes	≥ 50 ns	Programmable length from 1 to 15 I ² C peripheral clocks

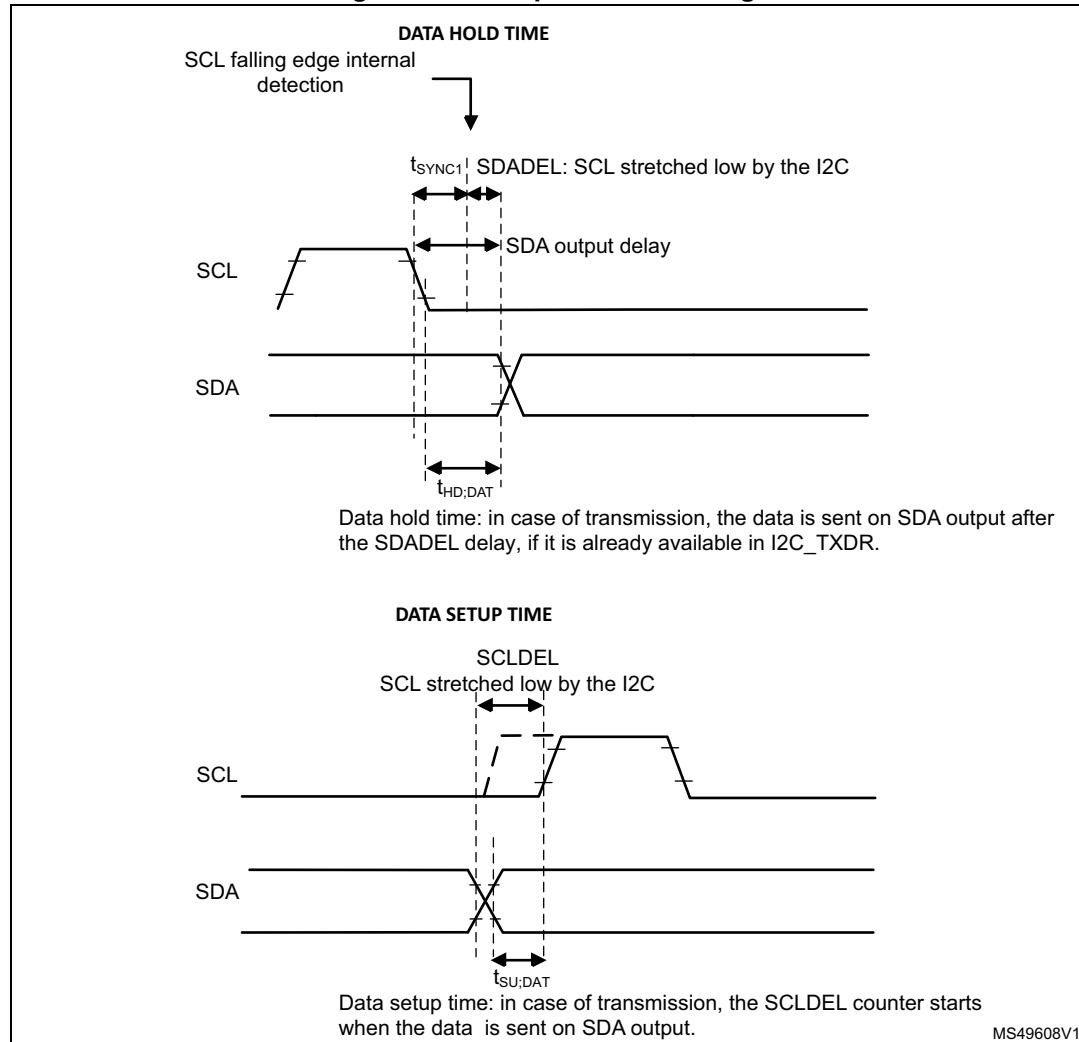
Caution: Changing the filter configuration is not allowed when the I²C is enabled.

I2C timings

The timings must be configured in order to guarantee a correct data hold and setup time, used in master and slave modes. This is done by programming the PRESC[3:0], SCLDEL[3:0] and SDADEL[3:0] bits in the I2C_TIMINGR register.

The STM32CubeMX tool calculates and provides the I2C_TIMINGR content in the I2C configuration window

Figure 289. Setup and hold timings



- When the SCL falling edge is internally detected, a delay is inserted before sending SDA output. This delay is $t_{SDADEL} = SDADEL \times t_{PRESC} + t_{I2CCLK}$ where $t_{PRESC} = (PRESC+1) \times t_{I2CCLK}$.
 t_{SDADEL} impacts the hold time $t_{HD;DAT}$.

The total SDA output delay is:

$$t_{SYNC1} + \{[SDADEL \times (PRESC+1) + 1] \times t_{I2CCLK}\}$$

t_{SYNC1} duration depends on these parameters:

- SCL falling slope
- When enabled, input delay brought by the analog filter: $t_{AF(min)} < t_{AF} < t_{AF(max)}$ ns.
- When enabled, input delay brought by the digital filter: $t_{DNF} = DNF \times t_{I2CCLK}$
- Delay due to SCL synchronization to I2CCLK clock (2 to 3 I2CCLK periods)

In order to bridge the undefined region of the SCL falling edge, the user must program SDADEL in such a way that:

$$\{t_f(max) + t_{HD;DAT}(min) - t_{AF(min)} - [(DNF+3) \times t_{I2CCLK}]\} / \{(PRESC+1) \times t_{I2CCLK}\} \leq SDADEL$$

$$SDADEL \leq \{t_{HD;DAT}(max) - t_{AF(max)} - [(DNF+4) \times t_{I2CCLK}]\} / \{(PRESC+1) \times t_{I2CCLK}\}$$

Note: $t_{AF(min)}$ / $t_{AF(max)}$ are part of the equation only when the analog filter is enabled. Refer to device datasheet for t_{AF} values.

The maximum $t_{HD;DAT}$ can be 3.45 µs, 0.9 µs and 0.45 µs for Standard-mode, Fast-mode and Fast-mode Plus, but must be less than the maximum of $t_{VD;DAT}$ by a transition time. This maximum must only be met if the device does not stretch the LOW period (t_{LOW}) of the SCL signal. If the clock stretches the SCL, the data must be valid by the set-up time before it releases the clock.

The SDA rising edge is usually the worst case, so in this case the previous equation becomes:

$$SDADEL \leq \{t_{VD;DAT}(max) - t_r(max) - 260\text{ ns} - [(DNF+4) \times t_{I2CCLK}]\} / \{(PRESC+1) \times t_{I2CCLK}\}.$$

Note: This condition can be violated when $NOSTRETCH=0$, because the device stretches SCL low to guarantee the set-up time, according to the SCLDEL value.

Refer to [Table 146: I2C-SMBUS specification data setup and hold times](#) for t_f , t_r , $t_{HD;DAT}$ and $t_{VD;DAT}$ standard values.

- After t_{SDADEL} delay, or after sending SDA output in case the slave had to stretch the clock because the data was not yet written in I2C_TXDR register, SCL line is kept at low level during the setup time. This setup time is $t_{SCLDEL} = (SCLDEL+1) \times t_{PRESC}$ where $t_{PRESC} = (PRESC+1) \times t_{I2CCLK}$.

t_{SCLDEL} impacts the setup time $t_{SU;DAT}$.

In order to bridge the undefined region of the SDA transition (rising edge usually worst case), the user must program SCLDEL in such a way that:

$$\{[t_r(max) + t_{SU;DAT}(min)] / [(PRESC+1) \times t_{I2CCLK}]\} - 1 \leq SCLDEL$$

Refer to [Table 146: I2C-SMBUS specification data setup and hold times](#) for t_r and $t_{SU;DAT}$ standard values.

The SDA and SCL transition time values to be used are the ones in the application. Using the maximum values from the standard increases the constraints for the SDADEL and SCLDEL calculation, but ensures the feature whatever the application.

Note: At every clock pulse, after SCL falling edge detection, the I2C master or slave stretches SCL low during at least $[(SDADEL+SCLDEL+1) \times (PRESC+1) + 1] \times t_{I2CCLK}$, in both transmission and reception modes. In transmission mode, in case the data is not yet written in I2C_TXDR when SDADEL counter is finished, the I2C keeps on stretching SCL low until the next data is written. Then new data MSB is sent on SDA output, and SCLDEL counter starts, continuing stretching SCL low to guarantee the data setup time.

If NOSTRETCH=1 in slave mode, the SCL is not stretched. Consequently the SDADEL must be programmed in such a way to guarantee also a sufficient setup time.

Table 146. I²C-SMBUS specification data setup and hold times

Symbol	Parameter	Standard-mode (Sm)		Fast-mode (Fm)		Fast-mode Plus (Fm+)		SMBUS		Unit
		Min.	Max	Min.	Max	Min.	Max	Min.	Max	
$t_{HD;DAT}$	Data hold time	0	-	0	-	0	-	0.3	-	μs
$t_{VD;DAT}$	Data valid time	-	3.45	-	0.9	-	0.45	-	-	
$t_{SU;DAT}$	Data setup time	250	-	100	-	50	-	250	-	ns
t_r	Rise time of both SDA and SCL signals	-	1000	-	300	-	120	-	1000	
t_f	Fall time of both SDA and SCL signals	-	300	-	300	-	120	-	300	

Additionally, in master mode, the SCL clock high and low levels must be configured by programming the PRESC[3:0], SCLH[7:0] and SCLL[7:0] bits in the I2C_TIMINGR register.

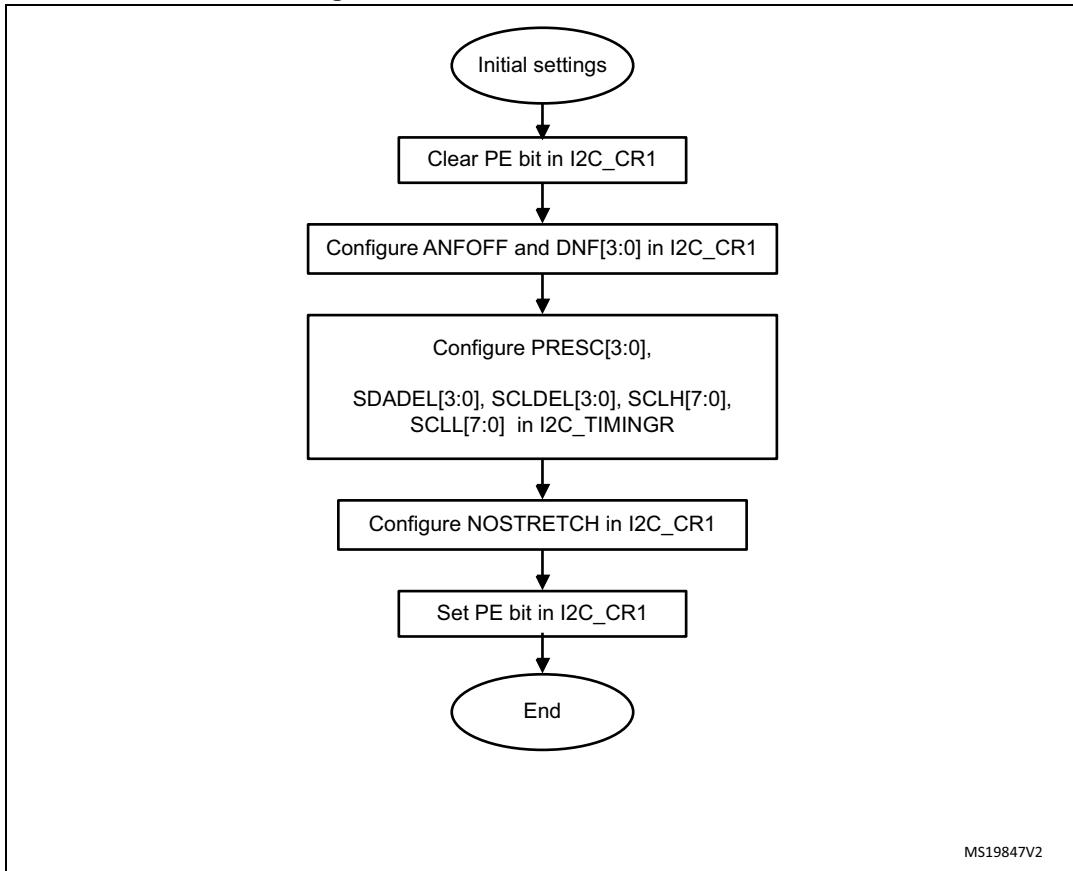
- When the SCL falling edge is internally detected, a delay is inserted before releasing the SCL output. This delay is $t_{SCLL} = (SCLL+1) \times t_{PRESC}$ where $t_{PRESC} = (PRESC+1) \times t_{I2CCLK}$. t_{SCLL} impacts the SCL low time t_{LOW} .
- When the SCL rising edge is internally detected, a delay is inserted before forcing the SCL output to low level. This delay is $t_{SCLH} = (SCLH+1) \times t_{PRESC}$ where $t_{PRESC} = (PRESC+1) \times t_{I2CCLK}$. t_{SCLH} impacts the SCL high time t_{HIGH} .

Refer to [I2C master initialization](#) for more details.

Caution: Changing the timing configuration is not allowed when the I2C is enabled.

The I2C slave NOSTRETCH mode must also be configured before enabling the peripheral. Refer to [I2C slave initialization](#) for more details.

Caution: Changing the NOSTRETCH configuration is not allowed when the I2C is enabled.

Figure 290. I2C initialization flowchart

26.4.5 Software reset

A software reset can be performed by clearing the PE bit in the I2C_CR1 register. In that case I2C lines SCL and SDA are released. Internal states machines are reset and communication control bits, as well as status bits come back to their reset value. The configuration registers are not impacted.

Here is the list of impacted register bits:

1. I2C_CR2 register: START, STOP, NACK
2. I2C_ISR register: BUSY, TXE, TXIS, RXNE, ADDR, NACKF, TCR, TC, STOPF, BERR, ARLO, OVR

and in addition when the SMBus feature is supported:

1. I2C_CR2 register: PECBYTE
2. I2C_ISR register: PECERR, TIMEOUT, ALERT

PE must be kept low during at least 3 APB clock cycles in order to perform the software reset. This is ensured by writing the following software sequence: - Write PE=0 - Check PE=0 - Write PE=1.

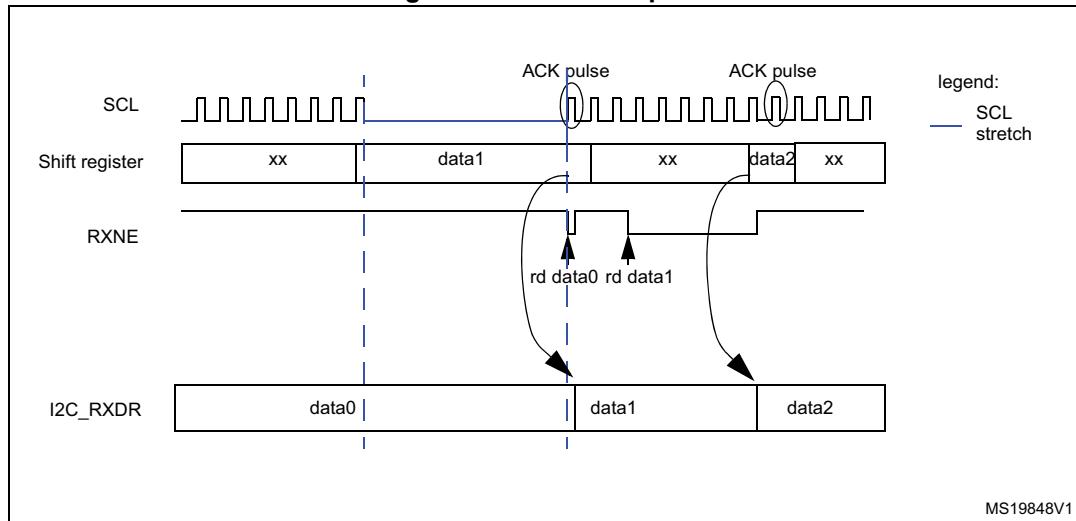
26.4.6 Data transfer

The data transfer is managed through transmit and receive data registers and a shift register.

Reception

The SDA input fills the shift register. After the 8th SCL pulse (when the complete data byte is received), the shift register is copied into I2C_RXDR register if it is empty (RXNE=0). If RXNE=1, meaning that the previous received data byte has not yet been read, the SCL line is stretched low until I2C_RXDR is read. The stretch is inserted between the 8th and 9th SCL pulse (before the Acknowledge pulse).

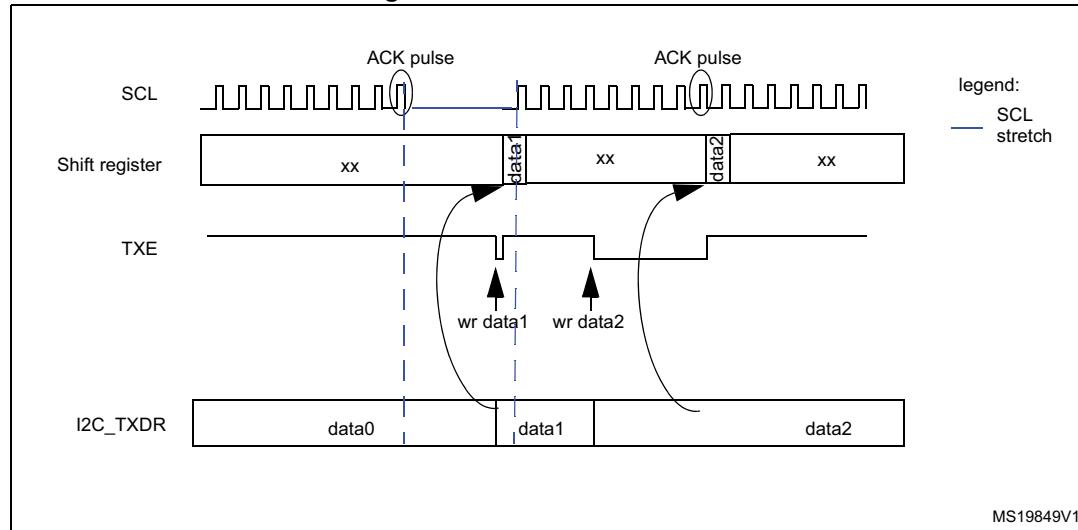
Figure 291. Data reception



Transmission

If the I2C_TXDR register is not empty (TXE=0), its content is copied into the shift register after the 9th SCL pulse (the Acknowledge pulse). Then the shift register content is shifted out on SDA line. If TXE=1, meaning that no data is written yet in I2C_TXDR, SCL line is stretched low until I2C_TXDR is written. The stretch is done after the 9th SCL pulse.

Figure 292. Data transmission



Hardware transfer management

The I2C has a byte counter embedded in hardware in order to manage byte transfer and to close the communication in various modes such as:

- NACK, STOP and ReSTART generation in master mode
- ACK control in slave receiver mode
- PEC generation/checking when SMBus feature is supported

The byte counter is always used in master mode. By default it is disabled in slave mode, but it can be enabled by software by setting the SBC (Slave Byte Control) bit in the I2C_CR2 register.

The number of bytes to be transferred is programmed in the NBYTES[7:0] bit field in the I2C_CR2 register. If the number of bytes to be transferred (NBYTES) is greater than 255, or if a receiver wants to control the acknowledge value of a received data byte, the reload mode must be selected by setting the RELOAD bit in the I2C_CR2 register. In this mode, TCR flag is set when the number of bytes programmed in NBYTES has been transferred, and an interrupt is generated if TCIE is set. SCL is stretched as long as TCR flag is set. TCR is cleared by software when NBYTES is written to a non-zero value.

When the NBYTES counter is reloaded with the last number of bytes, RELOAD bit must be cleared.

When RELOAD=0 in master mode, the counter can be used in 2 modes:

- **Automatic end mode** (AUTOEND = '1' in the I2C_CR2 register). In this mode, the master automatically sends a STOP condition once the number of bytes programmed in the NBYTES[7:0] bit field has been transferred.
- **Software end mode** (AUTOEND = '0' in the I2C_CR2 register). In this mode, software action is expected once the number of bytes programmed in the NBYTES[7:0] bit field has been transferred; the TC flag is set and an interrupt is generated if the TCIE bit is set. The SCL signal is stretched as long as the TC flag is set. The TC flag is cleared by software when the START or STOP bit is set in the I2C_CR2 register. This mode must be used when the master wants to send a RESTART condition.

Caution: The AUTOEND bit has no effect when the RELOAD bit is set.

Table 147. I2C configuration

Function	SBC bit	RELOAD bit	AUTOEND bit
Master Tx/Rx NBYTES + STOP	x	0	1
Master Tx/Rx + NBYTES + RESTART	x	0	0
Slave Tx/Rx all received bytes ACKed	0	x	x
Slave Rx with ACK control	1	1	x

26.4.7 I2C slave mode

I2C slave initialization

In order to work in slave mode, the user must enable at least one slave address. Two registers I2C_OAR1 and I2C_OAR2 are available in order to program the slave own addresses OA1 and OA2.

- OA1 can be configured either in 7-bit mode (by default) or in 10-bit addressing mode by setting the OA1MODE bit in the I2C_OAR1 register.
OA1 is enabled by setting the OA1EN bit in the I2C_OAR1 register.
- If additional slave addresses are required, the 2nd slave address OA2 can be configured. Up to 7 OA2 LSB can be masked by configuring the OA2MSK[2:0] bits in the I2C_OAR2 register. Therefore for OA2MSK configured from 1 to 6, only OA2[7:2], OA2[7:3], OA2[7:4], OA2[7:5], OA2[7:6] or OA2[7] are compared with the received address. As soon as OA2MSK is not equal to 0, the address comparator for OA2 excludes the I2C reserved addresses (0000 XXX and 1111 XXX), which are not acknowledged. If OA2MSK=7, all received 7-bit addresses are acknowledged (except reserved addresses). OA2 is always a 7-bit address.

These reserved addresses can be acknowledged if they are enabled by the specific enable bit, if they are programmed in the I2C_OAR1 or I2C_OAR2 register with OA2MSK=0.

OA2 is enabled by setting the OA2EN bit in the I2C_OAR2 register.

- The General Call address is enabled by setting the GCEN bit in the I2C_CR1 register.

When the I2C is selected by one of its enabled addresses, the ADDR interrupt status flag is set, and an interrupt is generated if the ADDRIE bit is set.

By default, the slave uses its clock stretching capability, which means that it stretches the SCL signal at low level when needed, in order to perform software actions. If the master does not support clock stretching, the I2C must be configured with NOSTRETCH=1 in the I2C_CR1 register.

After receiving an ADDR interrupt, if several addresses are enabled the user must read the ADDCODE[6:0] bits in the I2C_ISR register in order to check which address matched. DIR flag must also be checked in order to know the transfer direction.

Slave clock stretching (NOSTRETCH = 0)

In default mode, the I2C slave stretches the SCL clock in the following situations:

- When the ADDR flag is set: the received address matches with one of the enabled slave addresses. This stretch is released when the ADDR flag is cleared by software setting the ADDRCF bit.
- In transmission, if the previous data transmission is completed and no new data is written in I2C_TXDR register, or if the first data byte is not written when the ADDR flag is cleared (TXE=1). This stretch is released when the data is written to the I2C_TXDR register.
- In reception when the I2C_RXDR register is not read yet and a new data reception is completed. This stretch is released when I2C_RXDR is read.
- When TCR = 1 in Slave Byte Control mode, reload mode (SBC=1 and RELOAD=1), meaning that the last data byte has been transferred. This stretch is released when then TCR is cleared by writing a non-zero value in the NBYTES[7:0] field.
- After SCL falling edge detection, the I2C stretches SCL low during $[(SDADEL+SCLDEL+1) \times (PRESC+1) + 1] \times t_{I2CCLK}$.

Slave without clock stretching (NOSTRETCH = 1)

When NOSTRETCH = 1 in the I2C_CR1 register, the I2C slave does not stretch the SCL signal.

- The SCL clock is not stretched while the ADDR flag is set.
- In transmission, the data must be written in the I2C_TXDR register before the first SCL pulse corresponding to its transfer occurs. If not, an underrun occurs, the OVR flag is set in the I2C_ISR register and an interrupt is generated if the ERRIE bit is set in the I2C_CR1 register. The OVR flag is also set when the first data transmission starts and the STOPF bit is still set (has not been cleared). Therefore, if the user clears the STOPF flag of the previous transfer only after writing the first data to be transmitted in the next transfer, he ensures that the OVR status is provided, even for the first data to be transmitted.
- In reception, the data must be read from the I2C_RXDR register before the 9th SCL pulse (ACK pulse) of the next data byte occurs. If not an overrun occurs, the OVR flag is set in the I2C_ISR register and an interrupt is generated if the ERRIE bit is set in the I2C_CR1 register.

Slave Byte Control mode

In order to allow byte ACK control in slave reception mode, Slave Byte Control mode must be enabled by setting the SBC bit in the I2C_CR1 register. This is required to be compliant with SMBus standards.

Reload mode must be selected in order to allow byte ACK control in slave reception mode (RELOAD=1). To get control of each byte, NBYTES must be initialized to 0x1 in the ADDR interrupt subroutine, and reloaded to 0x1 after each received byte. When the byte is received, the TCR bit is set, stretching the SCL signal low between the 8th and 9th SCL pulses. The user can read the data from the I2C_RXDR register, and then decide to acknowledge it or not by configuring the ACK bit in the I2C_CR2 register. The SCL stretch is released by programming NBYTES to a non-zero value: the acknowledge or not-acknowledge is sent and next byte can be received.

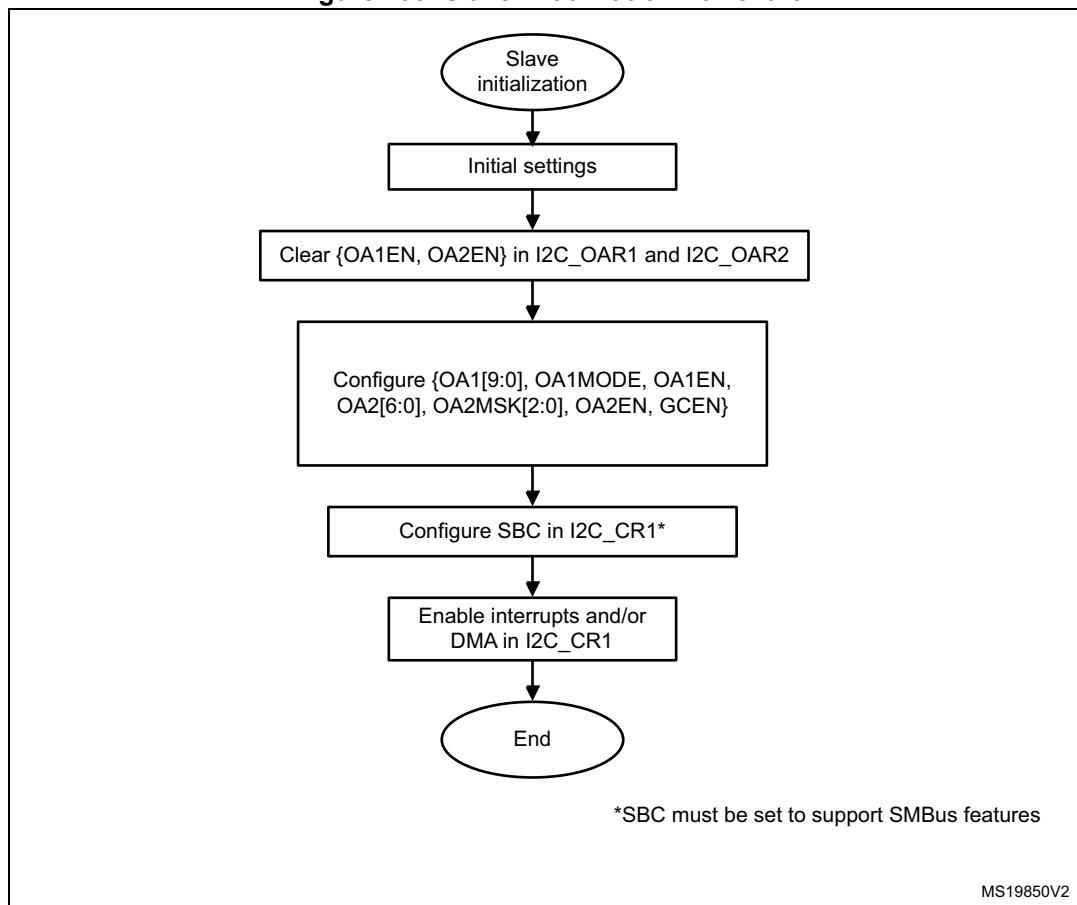
NBYTES can be loaded with a value greater than 0x1, and in this case, the reception flow is continuous during NBYTES data reception.

Note: *The SBC bit must be configured when the I2C is disabled, or when the slave is not addressed, or when ADDR=1.*

The RELOAD bit value can be changed when ADDR=1, or when TCR=1.

Caution: Slave Byte Control mode is not compatible with NOSTRETCH mode. Setting SBC when NOSTRETCH=1 is not allowed.

Figure 293. Slave initialization flowchart



Slave transmitter

A transmit interrupt status (TXIS) is generated when the I2C_TXDR register becomes empty. An interrupt is generated if the TXIE bit is set in the I2C_CR1 register.

The TXIS bit is cleared when the I2C_TXDR register is written with the next data byte to be transmitted.

When a NACK is received, the NACKF bit is set in the I2C_ISR register and an interrupt is generated if the NACKIE bit is set in the I2C_CR1 register. The slave automatically releases the SCL and SDA lines in order to let the master perform a STOP or a RESTART condition. The TXIS bit is not set when a NACK is received.

When a STOP is received and the STOPIE bit is set in the I2C_CR1 register, the STOPF flag is set in the I2C_ISR register and an interrupt is generated. In most applications, the SBC bit is usually programmed to '0'. In this case, If TXE = 0 when the slave address is received (ADDR=1), the user can choose either to send the content of the I2C_TXDR register as the first data byte, or to flush the I2C_TXDR register by setting the TXE bit in order to program a new data byte.

In Slave Byte Control mode (SBC=1), the number of bytes to be transmitted must be programmed in NBYTES in the address match interrupt subroutine (ADDR=1). In this case, the number of TXIS events during the transfer corresponds to the value programmed in NBYTES.

Caution: When NOSTRETCH=1, the SCL clock is not stretched while the ADDR flag is set, so the user cannot flush the I2C_TXDR register content in the ADDR subroutine, in order to program the first data byte. The first data byte to be sent must be previously programmed in the I2C_TXDR register:

- This data can be the data written in the last TXIS event of the previous transmission message.
- If this data byte is not the one to be sent, the I2C_TXDR register can be flushed by setting the TXE bit in order to program a new data byte. The STOPF bit must be cleared only after these actions, in order to guarantee that they are executed before the first data transmission starts, following the address acknowledge.

If STOPF is still set when the first data transmission starts, an underrun error is generated (the OVR flag is set).

If a TXIS event is needed, (Transmit Interrupt or Transmit DMA request), the user must set the TXIS bit in addition to the TXE bit, in order to generate a TXIS event.

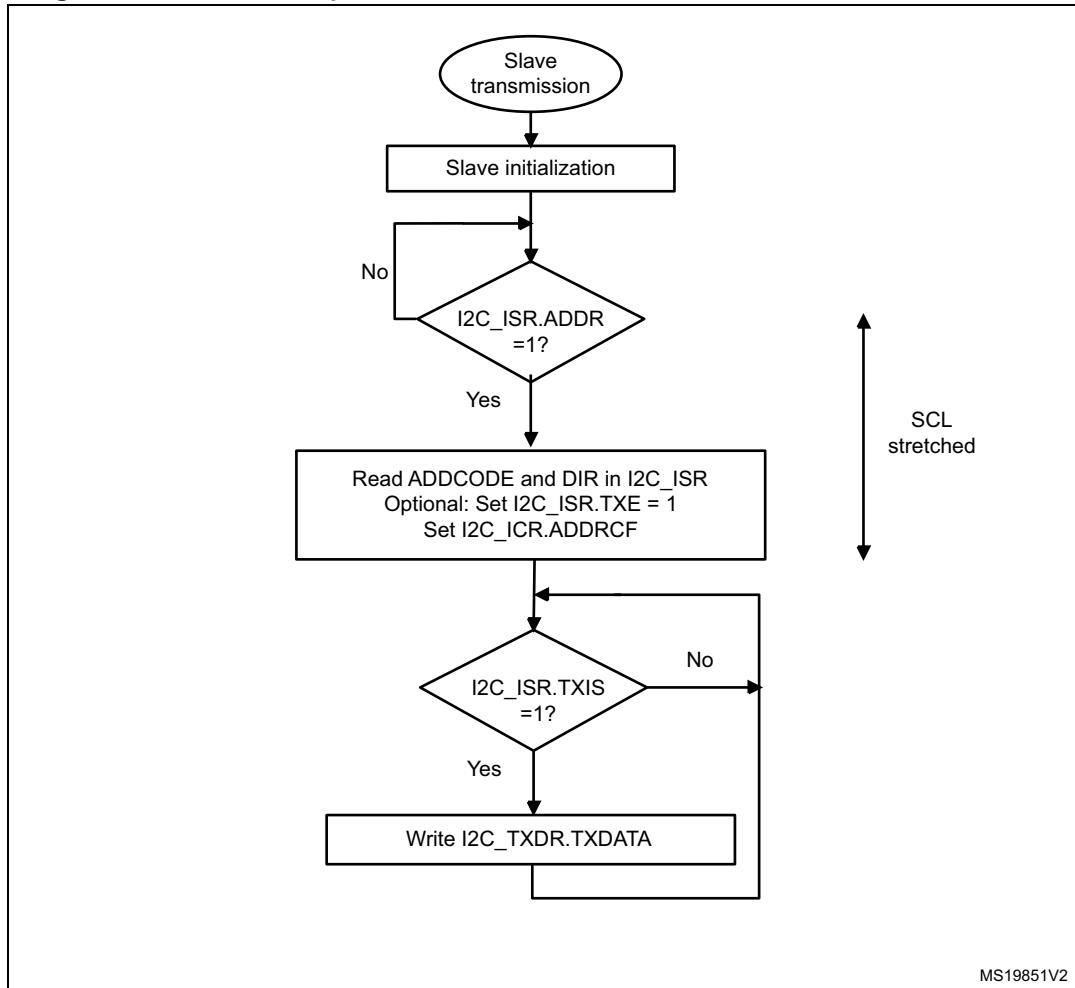
Figure 294. Transfer sequence flowchart for I2C slave transmitter, NOSTRETCH=0

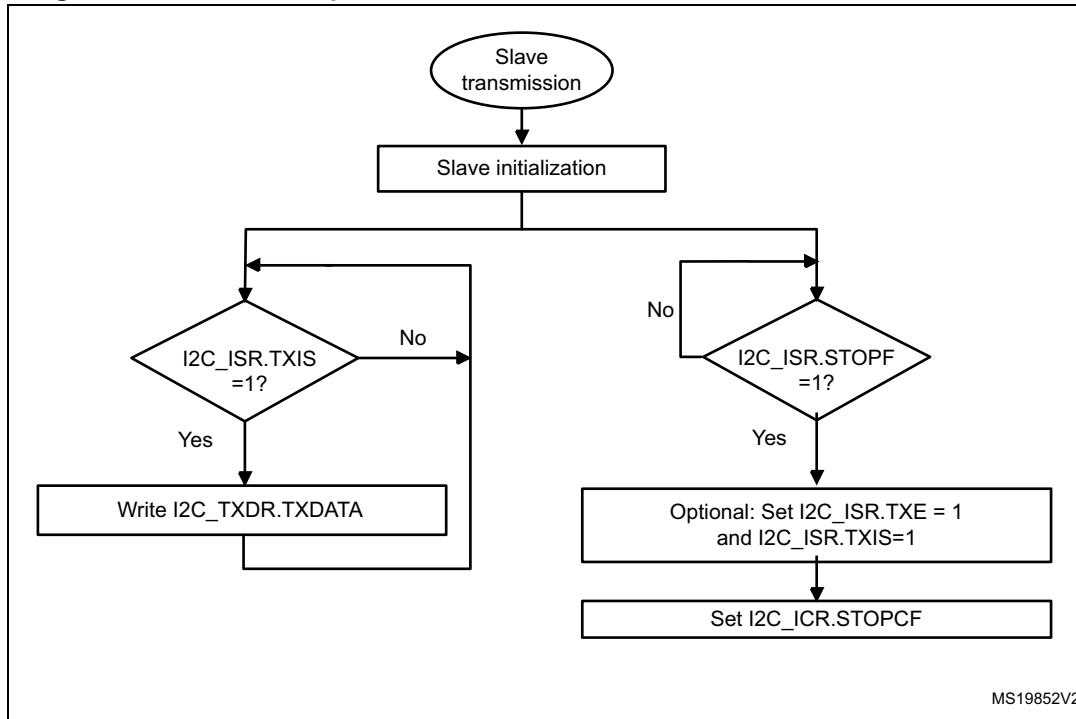
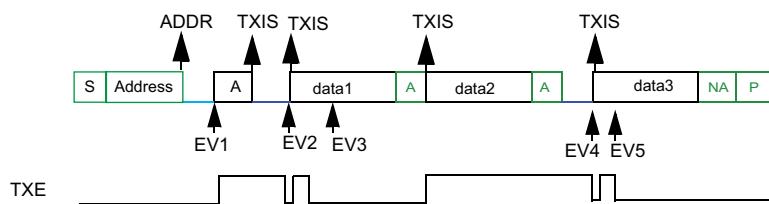
Figure 295. Transfer sequence flowchart for I2C slave transmitter, NOSTRETCH=1

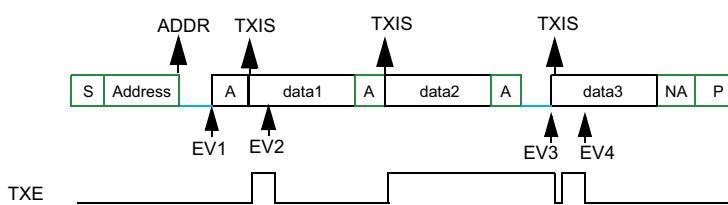
Figure 296. Transfer bus diagrams for I2C slave transmitter

Example I2C slave transmitter 3 bytes with 1st data flushed,
NOSTRETCH=0:



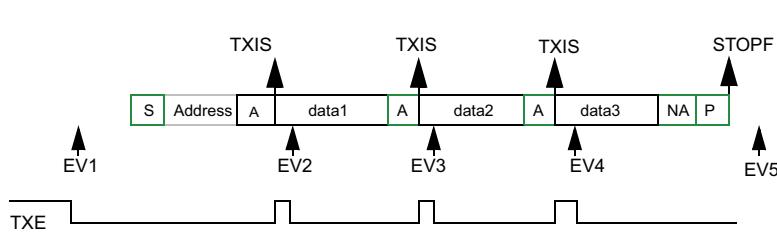
- EV1: ADDR ISR: check ADDCODE and DIR, set TXE, set ADDRCF
- EV2: TXIS ISR: wr data1
- EV3: TXIS ISR: wr data2
- EV4: TXIS ISR: wr data3
- EV5: TXIS ISR: wr data4 (not sent)

Example I2C slave transmitter 3 bytes without 1st data flush,
NOSTRETCH=0:



- EV1: ADDR ISR: check ADDCODE and DIR, set ADDRCF
- EV2: TXIS ISR: wr data2
- EV3: TXIS ISR: wr data3
- EV4: TXIS ISR: wr data4 (not sent)

Example I2C slave transmitter 3 bytes, NOSTRETCH=1:



- EV1: wr data1
- EV2: TXIS ISR: wr data2
- EV3: TXIS ISR: wr data3
- EV4: TXIS ISR: wr data4 (not sent)
- EV5: STOPF ISR: (optional: set TXE and TXIS), set STOPCF

legend:

- transmission
- reception
- SCL stretch

legend :

- transmission
- reception
- SCL stretch

legend:

- transmission
- reception
- SCL stretch

MS19853V1

Slave receiver

RXNE is set in I2C_ISR when the I2C_RXDR is full, and generates an interrupt if RXIE is set in I2C_CR1. RXNE is cleared when I2C_RXDR is read.

When a STOP is received and STOPIE is set in I2C_CR1, STOPF is set in I2C_ISR and an interrupt is generated.

Figure 297. Transfer sequence flowchart for slave receiver with NOSTRETCH=0

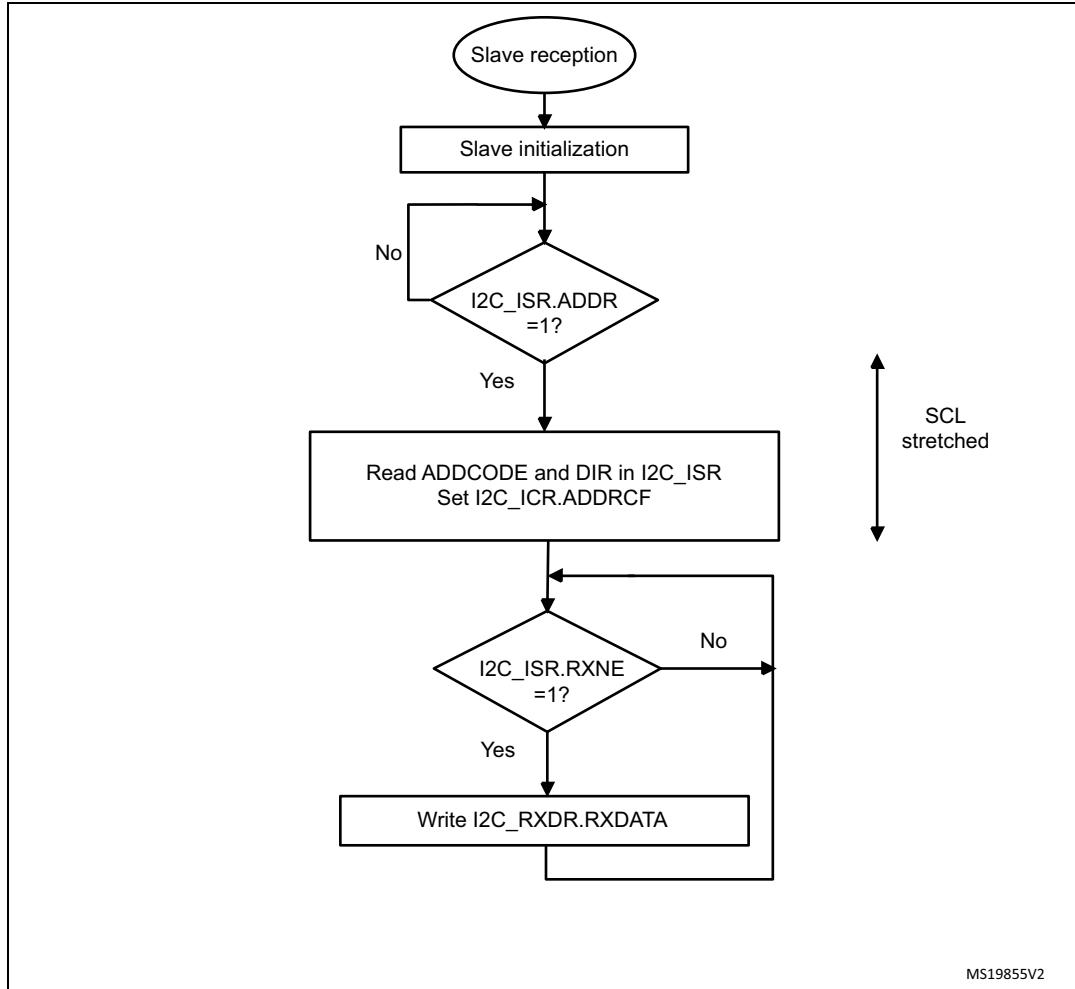
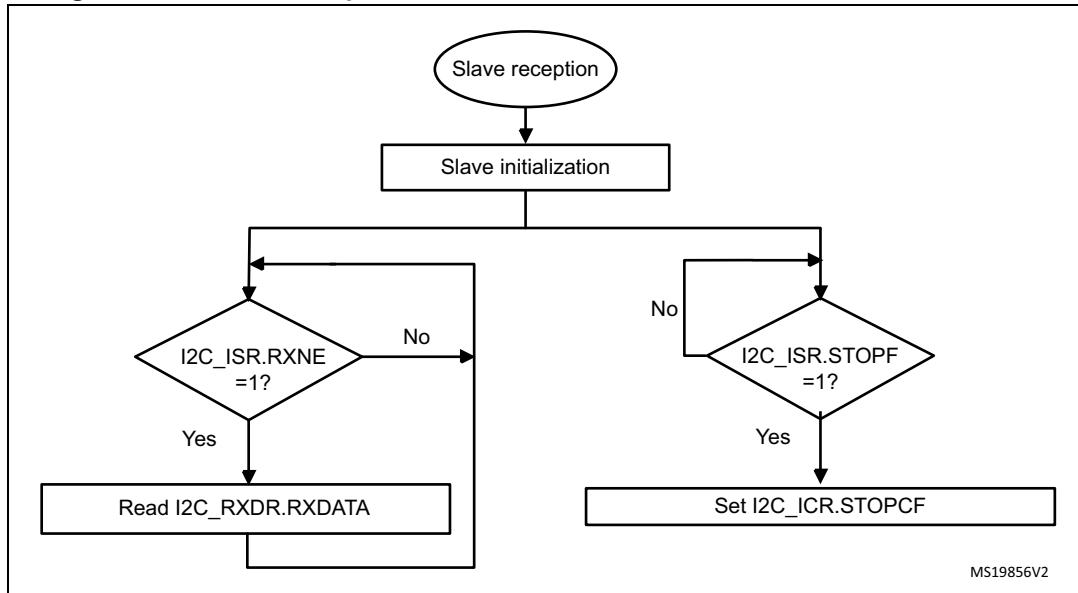
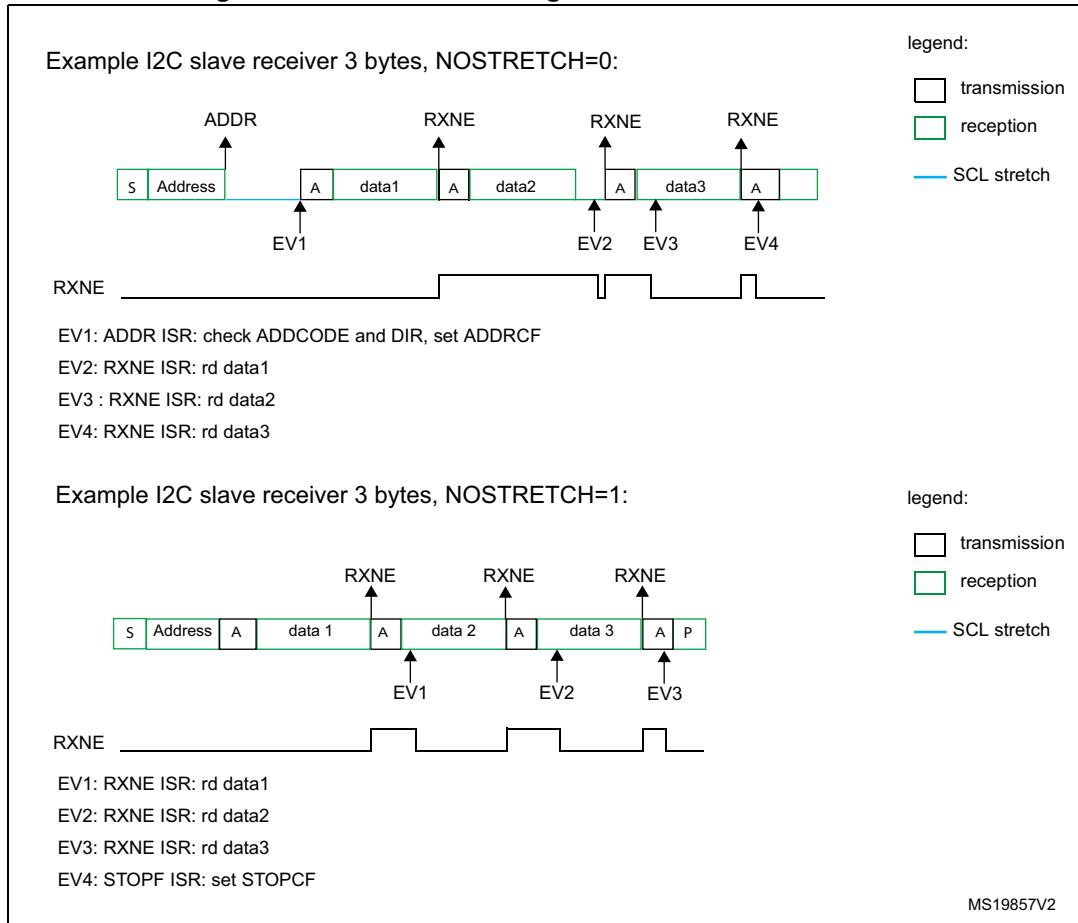


Figure 298. Transfer sequence flowchart for slave receiver with NOSTRETCH=1**Figure 299. Transfer bus diagrams for I2C slave receiver**

26.4.8 I2C master mode

I2C master initialization

Before enabling the peripheral, the I2C master clock must be configured by setting the SCLH and SCLL bits in the I2C_TIMINGR register.

The STM32CubeMX tool calculates and provides the I2C_TIMINGR content in the I2C Configuration window.

A clock synchronization mechanism is implemented in order to support multi-master environment and slave clock stretching.

In order to allow clock synchronization:

- The low level of the clock is counted using the SCLL counter, starting from the SCL low level internal detection.
- The high level of the clock is counted using the SCLH counter, starting from the SCL high level internal detection.

The I2C detects its own SCL low level after a t_{SYNC1} delay depending on the SCL falling edge, SCL input noise filters (analog + digital) and SCL synchronization to the I2CxCLK clock. The I2C releases SCL to high level once the SCLL counter reaches the value programmed in the SCLL[7:0] bits in the I2C_TIMINGR register.

The I2C detects its own SCL high level after a t_{SYNC2} delay depending on the SCL rising edge, SCL input noise filters (analog + digital) and SCL synchronization to I2CxCLK clock. The I2C ties SCL to low level once the SCLH counter is reached reaches the value programmed in the SCLH[7:0] bits in the I2C_TIMINGR register.

Consequently the master clock period is:

$$t_{SCL} = t_{SYNC1} + t_{SYNC2} + \{[(SCLH+1) + (SCLL+1)] \times (PRESC+1) \times t_{I2CCLK}\}$$

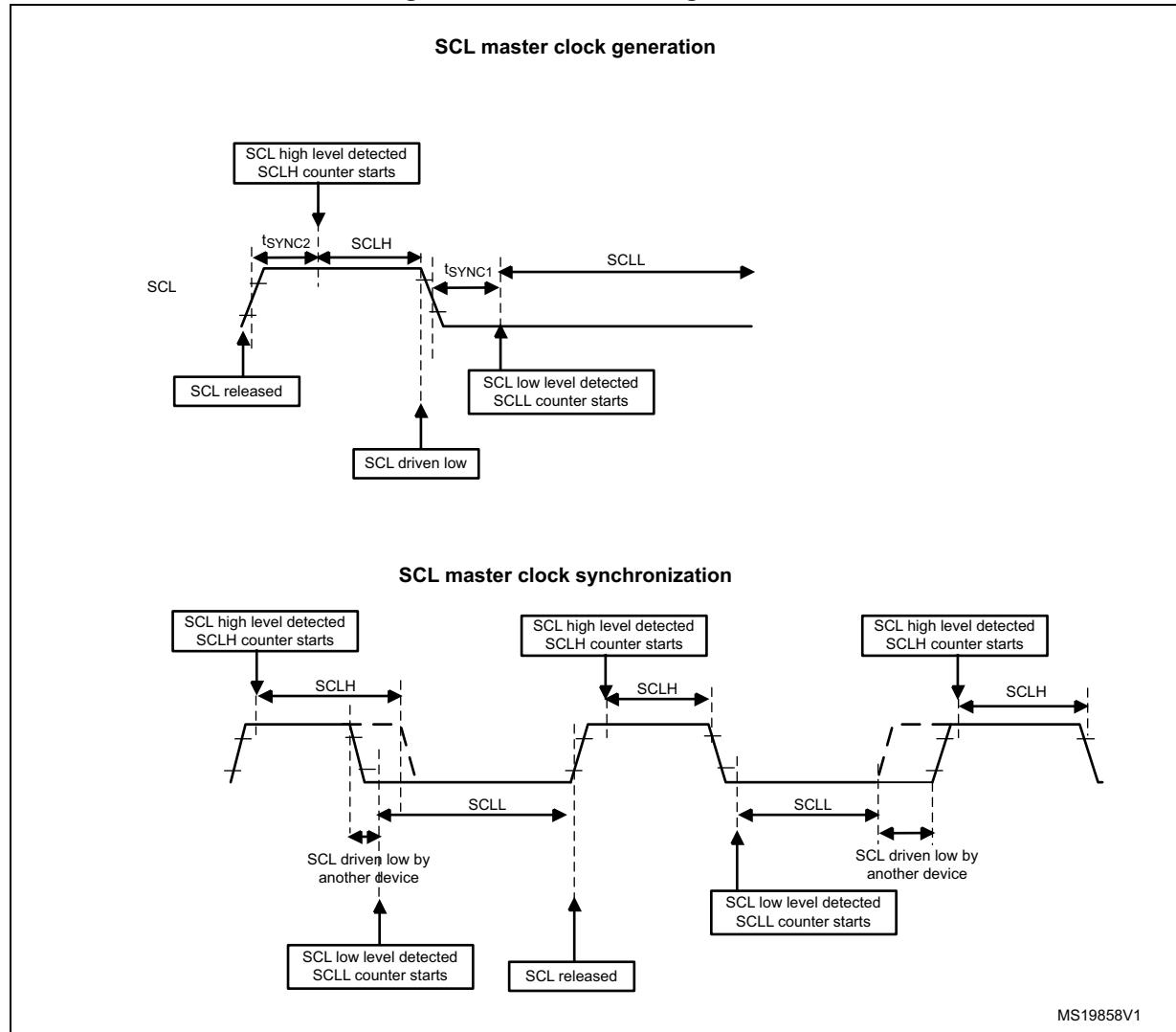
The duration of t_{SYNC1} depends on these parameters:

- SCL falling slope
- When enabled, input delay induced by the analog filter.
- When enabled, input delay induced by the digital filter: DNF $\times t_{I2CCLK}$
- Delay due to SCL synchronization with I2CCLK clock (2 to 3 I2CCLK periods)

The duration of t_{SYNC2} depends on these parameters:

- SCL rising slope
- When enabled, input delay induced by the analog filter.
- When enabled, input delay induced by the digital filter: DNF $\times t_{I2CCLK}$
- Delay due to SCL synchronization with I2CCLK clock (2 to 3 I2CCLK periods)

Figure 300. Master clock generation



Caution: In order to be I²C or SMBus compliant, the master clock must respect the timings given below:

Table 148. I²C-SMBUS specification clock timings

Symbol	Parameter	Standard-mode (Sm)		Fast-mode (Fm)		Fast-mode Plus (Fm+)		SMBUS		Unit
		Min	Max	Min	Max	Min	Max	Min	Max	
f _{SCL}	SCL clock frequency	-	100	-	400	-	1000	-	100	kHz
t _{HOLD:STA}	Hold time (repeated) START condition	4.0	-	0.6	-	0.26	-	4.0	-	μs
t _{SU:STA}	Set-up time for a repeated START condition	4.7	-	0.6	-	0.26	-	4.7	-	μs
t _{SU:STO}	Set-up time for STOP condition	4.0	-	0.6	-	0.26	-	4.0	-	μs
t _{BUF}	Bus free time between a STOP and START condition	4.7	-	1.3	-	0.5	-	4.7	-	μs
t _{LOW}	Low period of the SCL clock	4.7	-	1.3	-	0.5	-	4.7	-	μs
t _{HIGH}	Period of the SCL clock	4.0	-	0.6	-	0.26	-	4.0	50	μs
t _r	Rise time of both SDA and SCL signals	-	1000	-	300	-	120	-	1000	ns
t _f	Fall time of both SDA and SCL signals	-	300	-	300	-	120	-	300	ns

Note: SCLL is also used to generate the t_{BUF} and t_{SU:STA} timings.

SCLH is also used to generate the t_{HOLD:STA} and t_{SU:STO} timings.

Refer to [Section 26.4.9: I2C_TIMINGR register configuration examples](#) for examples of I2C_TIMINGR settings vs. I2CCLK frequency.

Master communication initialization (address phase)

In order to initiate the communication, the user must program the following parameters for the addressed slave in the I2C_CR2 register:

- Addressing mode (7-bit or 10-bit): ADD10
- Slave address to be sent: SADD[9:0]
- Transfer direction: RD_WRN
- In case of 10-bit address read: HEAD10R bit. HEAD10R must be configure to indicate if the complete address sequence must be sent, or only the header in case of a direction change.
- The number of bytes to be transferred: NBYTES[7:0]. If the number of bytes is equal to or greater than 255 bytes, NBYTES[7:0] must initially be filled with 0xFF.

The user must then set the START bit in I2C_CR2 register. Changing all the above bits is not allowed when START bit is set.

Then the master automatically sends the START condition followed by the slave address as soon as it detects that the bus is free (BUSY = 0) and after a delay of t_{BUF}.

In case of an arbitration loss, the master automatically switches back to slave mode and can acknowledge its own address if it is addressed as a slave.

Note: The START bit is reset by hardware when the slave address has been sent on the bus, whatever the received acknowledge value. The START bit is also reset by hardware if an arbitration loss occurs.

In 10-bit addressing mode, when the Slave Address first 7 bits is NACKed by the slave, the

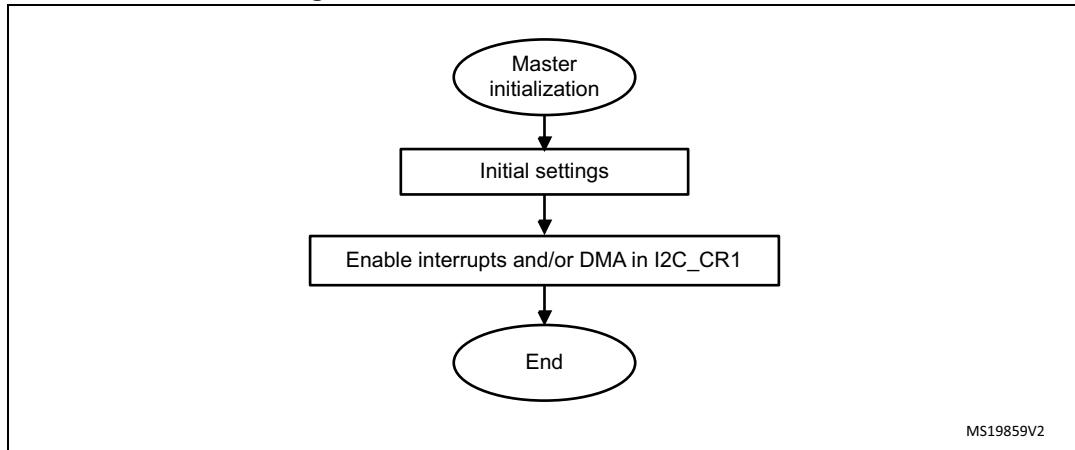
master re-launches automatically the slave address transmission until ACK is received. In this case ADDRCF must be set if a NACK is received from the slave, in order to stop sending the slave address.

If the I2C is addressed as a slave (ADDR=1) while the START bit is set, the I2C switches to slave mode and the START bit is cleared, when the ADDRCF bit is set.

Note:

The same procedure is applied for a Repeated Start condition. In this case BUSY=1.

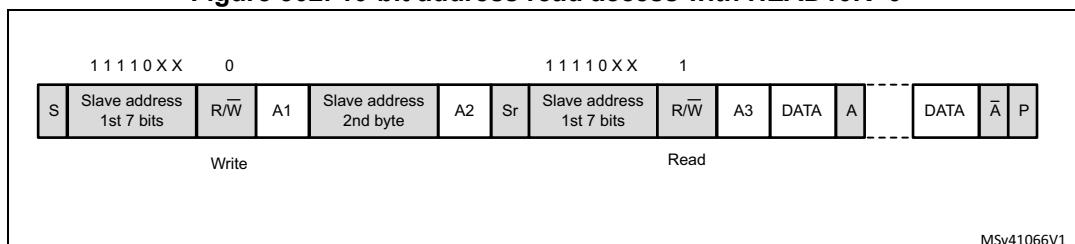
Figure 301. Master initialization flowchart



Initialization of a master receiver addressing a 10-bit address slave

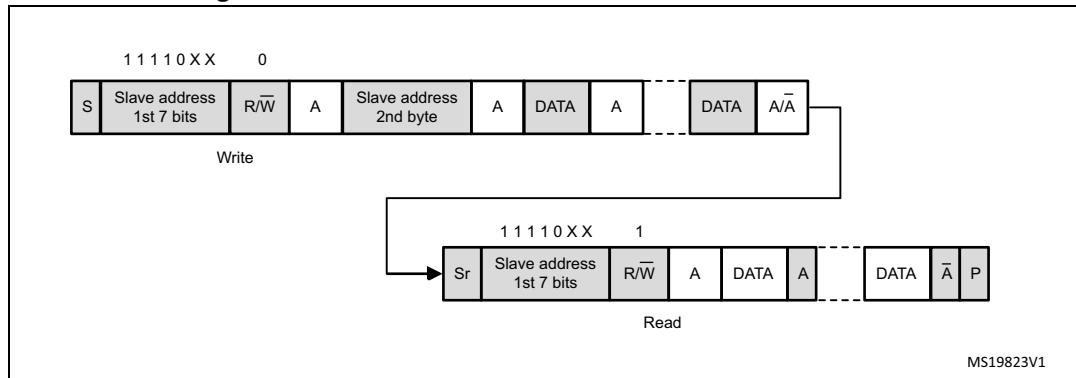
- If the slave address is in 10-bit format, the user can choose to send the complete read sequence by clearing the HEAD10R bit in the I2C_CR2 register. In this case the master automatically sends the following complete sequence after the START bit is set:
(Re)Start + Slave address 10-bit header Write + Slave address 2nd byte + REStart + Slave address 10-bit header Read

Figure 302. 10-bit address read access with HEAD10R=0



- If the master addresses a 10-bit address slave, transmits data to this slave and then reads data from the same slave, a master transmission flow must be done first. Then a repeated start is set with the 10 bit slave address configured with HEAD10R=1. In this case the master sends this sequence: ReStart + Slave address 10-bit header Read.

Figure 303. 10-bit address read access with HEAD10R=1



Master transmitter

In the case of a write transfer, the TXIS flag is set after each byte transmission, after the 9th SCL pulse when an ACK is received.

A TXIS event generates an interrupt if the TXIE bit is set in the I2C_CR1 register. The flag is cleared when the I2C_TXDR register is written with the next data byte to be transmitted.

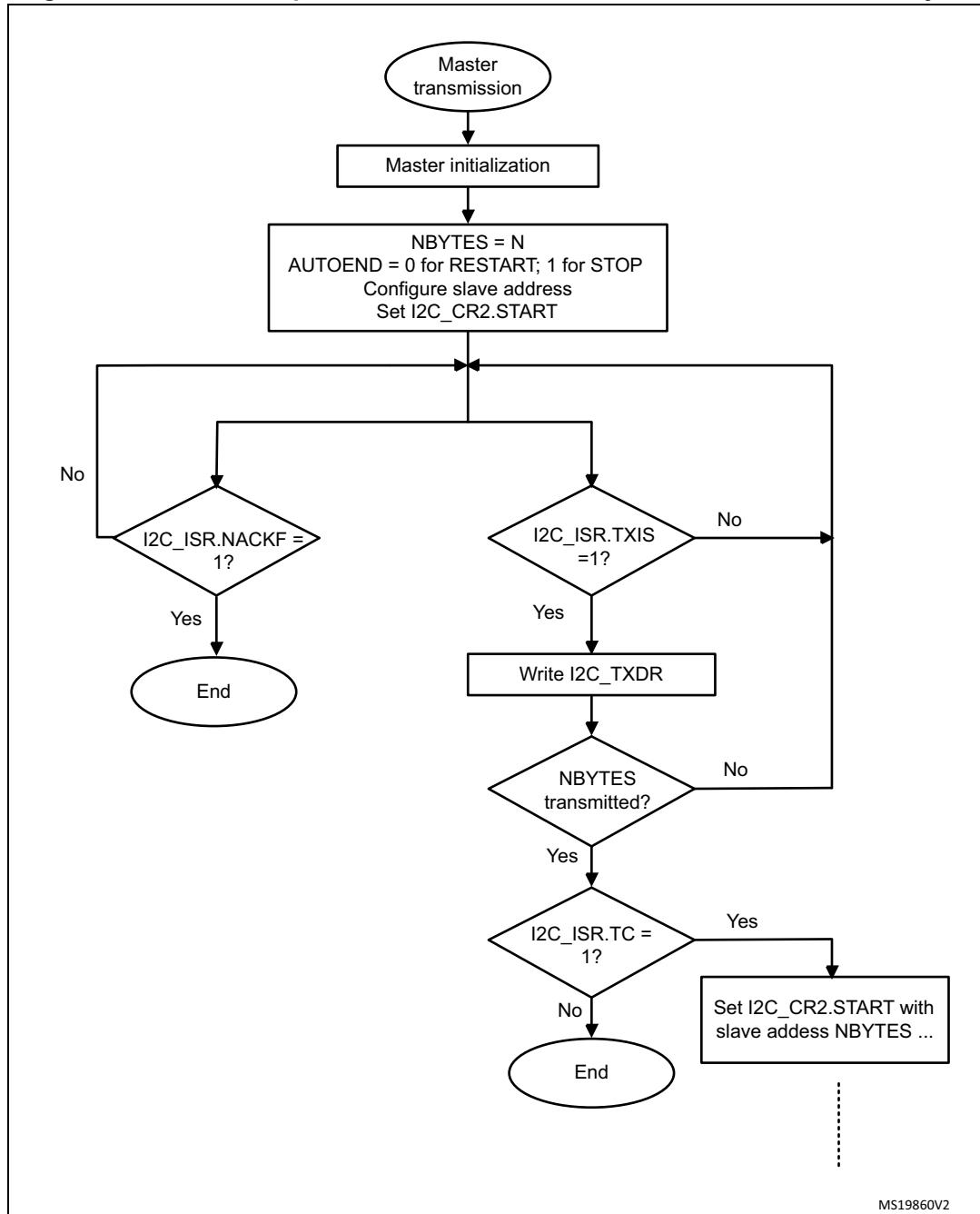
The number of TXIS events during the transfer corresponds to the value programmed in NBYTES[7:0]. If the total number of data bytes to be sent is greater than 255, reload mode must be selected by setting the RELOAD bit in the I2C_CR2 register. In this case, when NBYTES data have been transferred, the TCR flag is set and the SCL line is stretched low until NBYTES[7:0] is written to a non-zero value.

The TXIS flag is not set when a NACK is received.

- When RELOAD=0 and NBYTES data have been transferred:
 - In automatic end mode (AUTOEND=1), a STOP is automatically sent.
 - In software end mode (AUTOEND=0), the TC flag is set and the SCL line is stretched low in order to perform software actions:

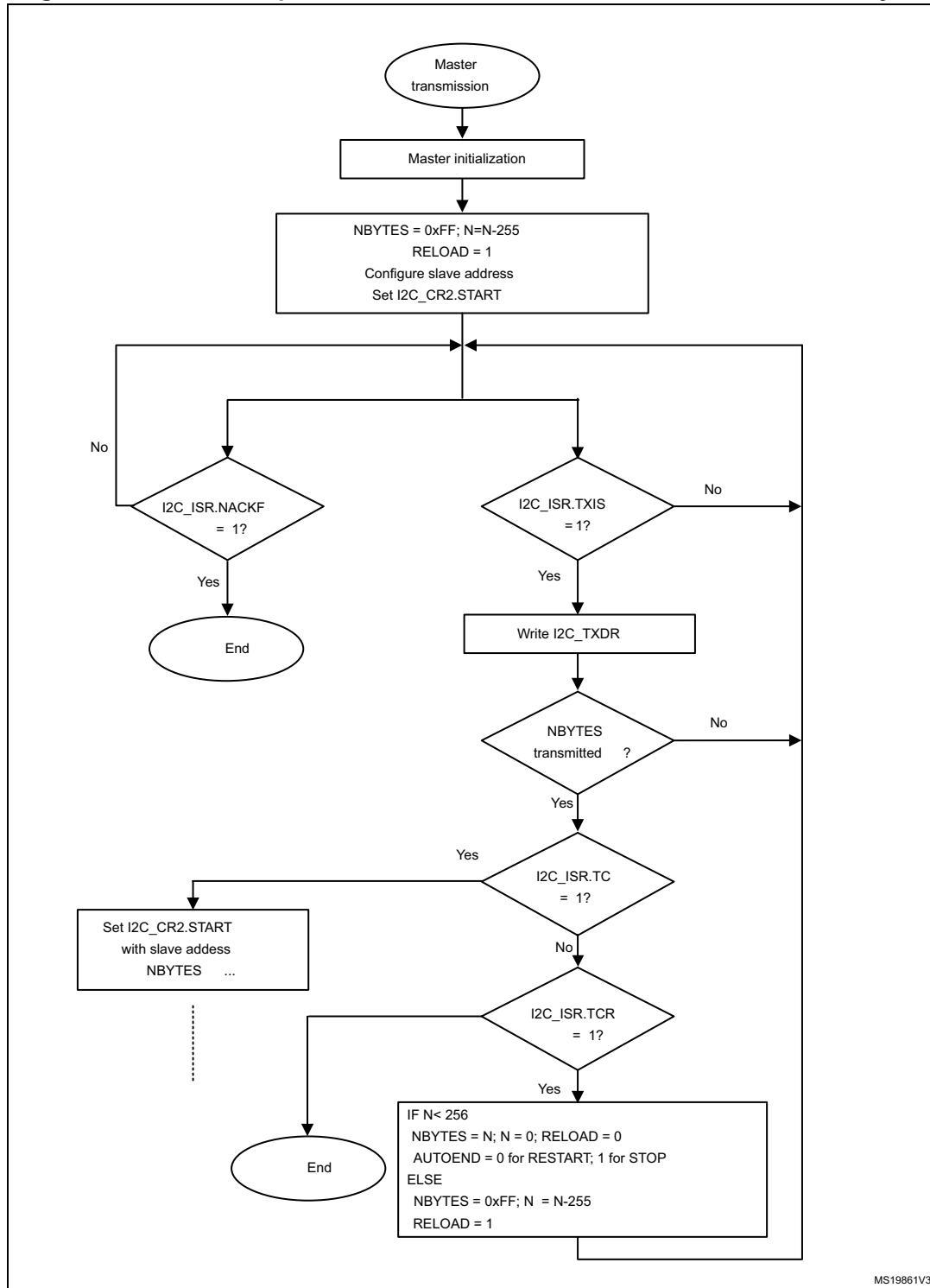
A RESTART condition can be requested by setting the START bit in the I2C_CR2 register with the proper slave address configuration, and number of bytes to be transferred. Setting the START bit clears the TC flag and the START condition is sent on the bus.

A STOP condition can be requested by setting the STOP bit in the I2C_CR2 register. Setting the STOP bit clears the TC flag and the STOP condition is sent on the bus.
- If a NACK is received: the TXIS flag is not set, and a STOP condition is automatically sent after the NACK reception. the NACKF flag is set in the I2C_ISR register, and an interrupt is generated if the NACKIE bit is set.

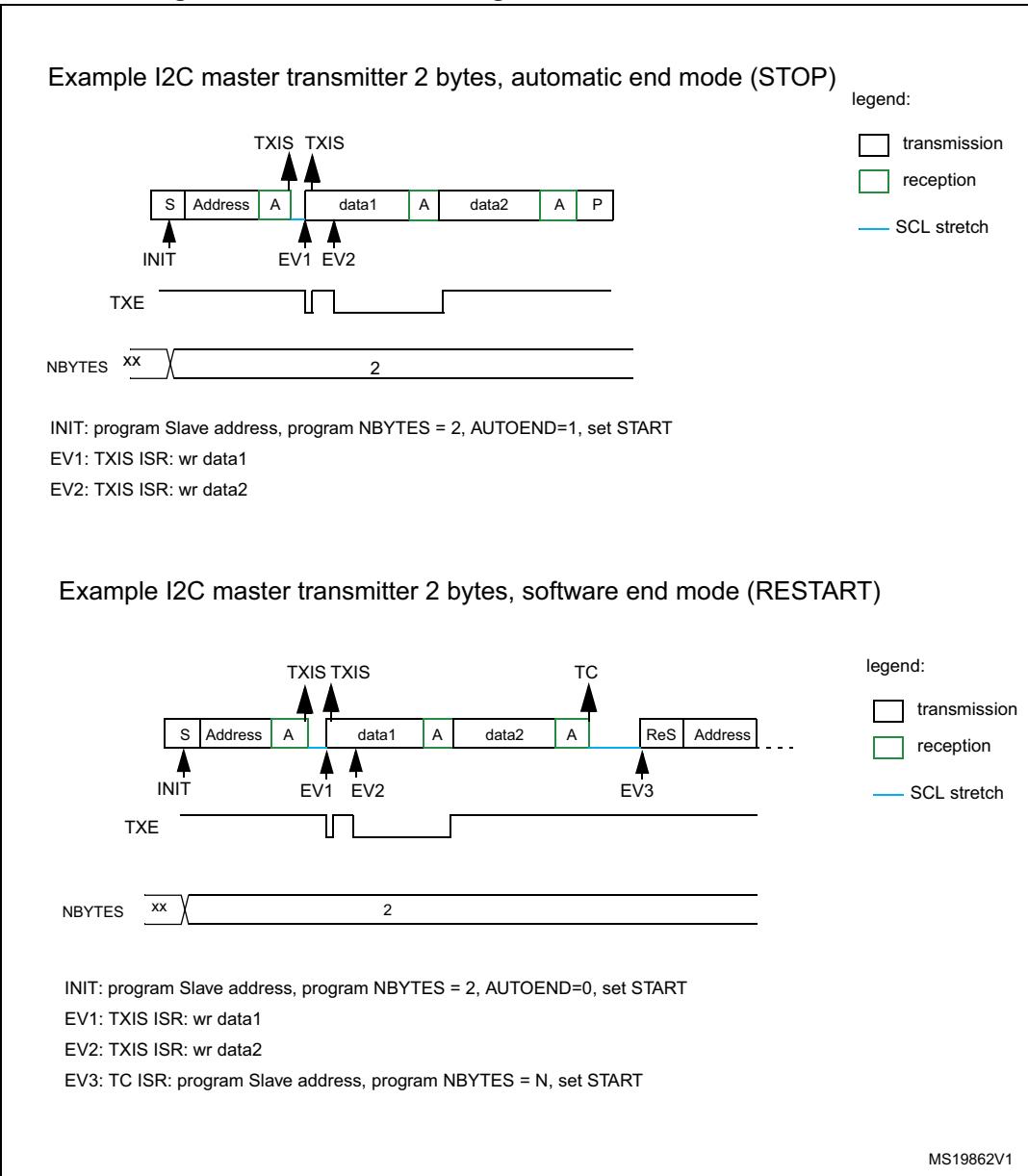
Figure 304. Transfer sequence flowchart for I2C master transmitter for $N \leq 255$ bytes

MS19860V2

Figure 305. Transfer sequence flowchart for I2C master transmitter for N>255 bytes



MS19861V3

Figure 306. Transfer bus diagrams for I2C master transmitter

Master receiver

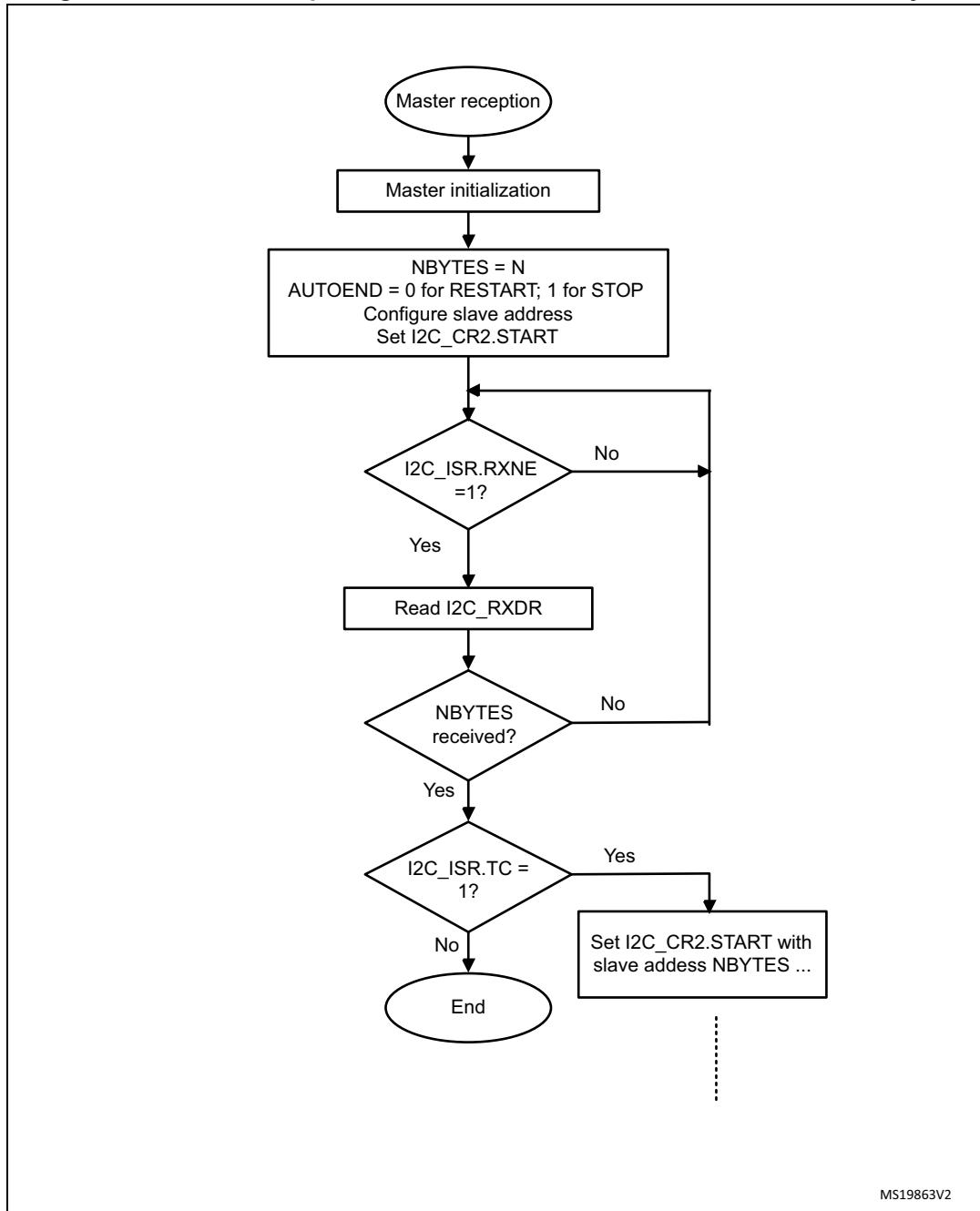
In the case of a read transfer, the RXNE flag is set after each byte reception, after the 8th SCL pulse. An RXNE event generates an interrupt if the RXIE bit is set in the I2C_CR1 register. The flag is cleared when I2C_RXDR is read.

If the total number of data bytes to be received is greater than 255, reload mode must be selected by setting the RELOAD bit in the I2C_CR2 register. In this case, when NBYTES[7:0] data have been transferred, the TCR flag is set and the SCL line is stretched low until NBYTES[7:0] is written to a non-zero value.

- When RELOAD=0 and NBYTES[7:0] data have been transferred:
 - In automatic end mode (AUTOEND=1), a NACK and a STOP are automatically sent after the last received byte.
 - In software end mode (AUTOEND=0), a NACK is automatically sent after the last received byte, the TC flag is set and the SCL line is stretched low in order to allow software actions:

A RESTART condition can be requested by setting the START bit in the I2C_CR2 register with the proper slave address configuration, and number of bytes to be transferred. Setting the START bit clears the TC flag and the START condition, followed by slave address, are sent on the bus.

A STOP condition can be requested by setting the STOP bit in the I2C_CR2 register. Setting the STOP bit clears the TC flag and the STOP condition is sent on the bus.

Figure 307. Transfer sequence flowchart for I2C master receiver for $N \leq 255$ bytes

MS19863V2

Figure 308. Transfer sequence flowchart for I2C master receiver for N >255 bytes

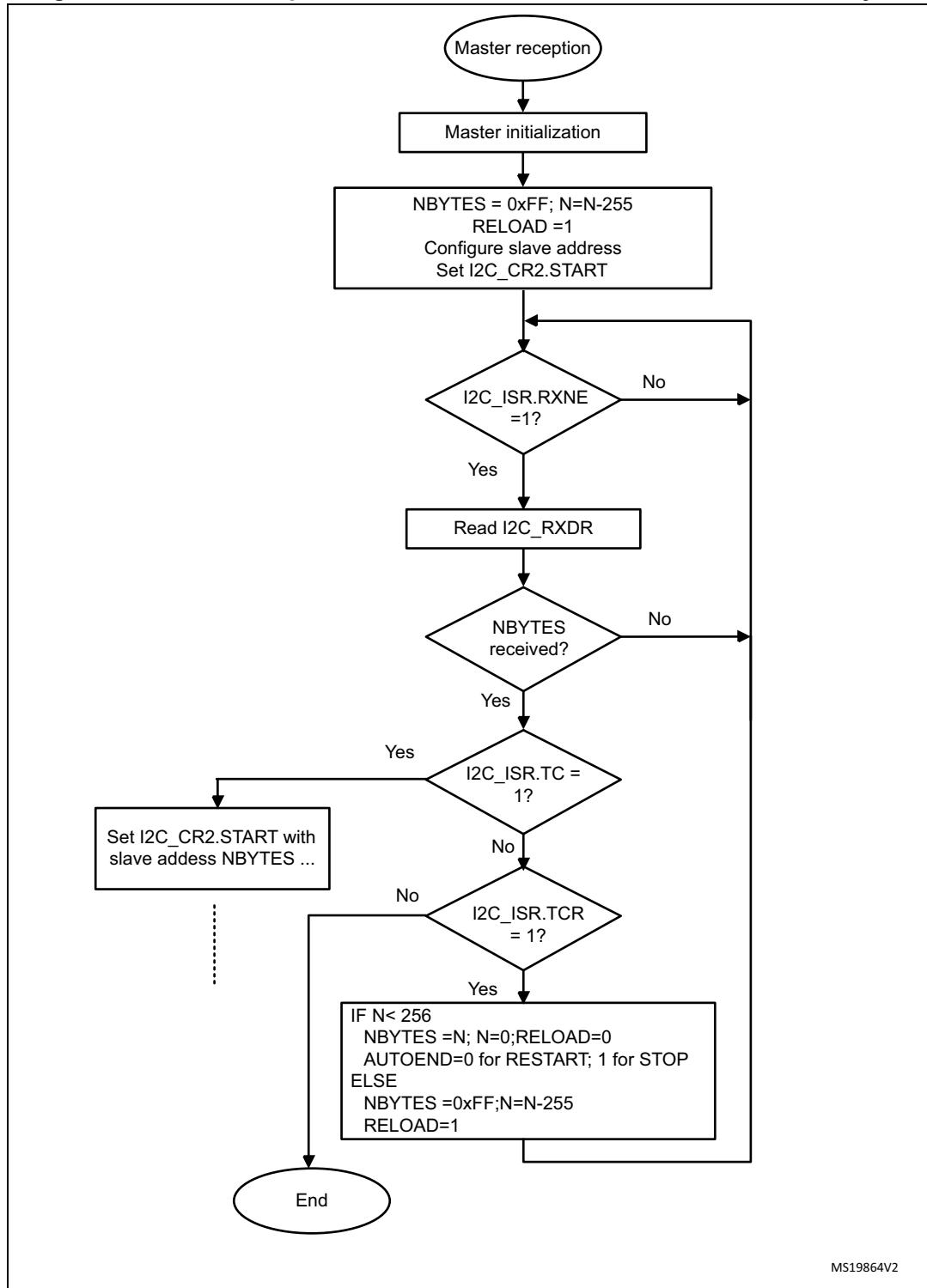
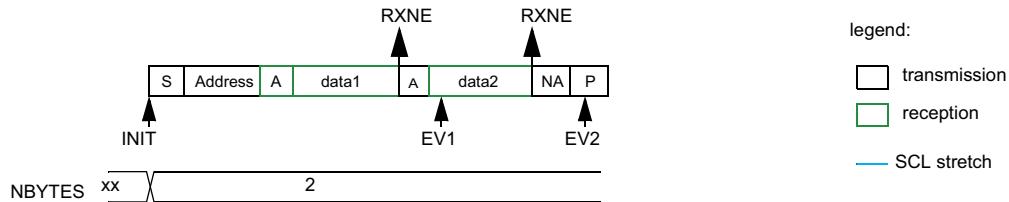


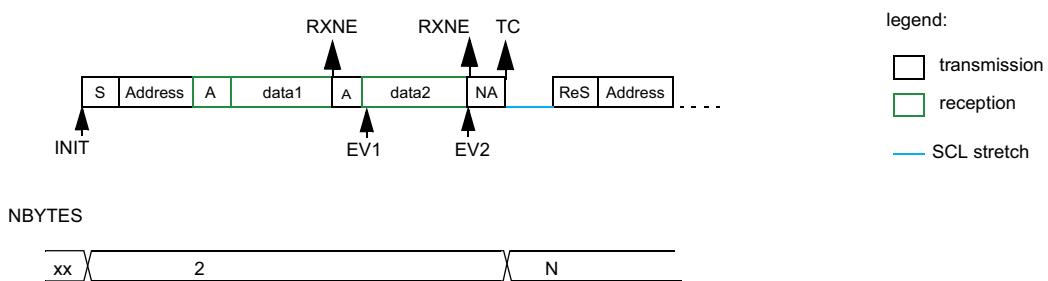
Figure 309. Transfer bus diagrams for I2C master receiver

Example I2C master receiver 2 bytes, automatic end mode (STOP)



INIT: program Slave address, program NBYTES = 2, AUTOEND=1, set START
EV1: RXNE ISR: rd data1
EV2: RXNE ISR: rd data2

Example I2C master receiver 2 bytes, software end mode (RESTART)



INIT: program Slave address, program NBYTES = 2, AUTOEND=0, set START
EV1: RXNE ISR: rd data1
EV2: RXNE ISR: read data2
EV3: TC ISR: program Slave address, program NBYTES = N, set START

MS19865V1

26.4.9 I2C_TIMINGR register configuration examples

The tables below provide examples of how to program the I2C_TIMINGR to obtain timings compliant with the I²C specification. In order to get more accurate configuration values, the STM32CubeMX tool (I2C Configuration window) must be used.

Table 149. Examples of timing settings for f_{I2CCCLK} = 8 MHz

Parameter	Standard-mode (Sm)		Fast-mode (Fm)	Fast-mode Plus (Fm+)
	10 kHz	100 kHz	400 kHz	500 kHz
PRESC	1	1	0	0
SCLL	0xC7	0x13	0x9	0x6
t _{SCLL}	200x250 ns = 50 µs	20x250 ns = 5.0 µs	10x125 ns = 1250 ns	7x125 ns = 875 ns
SCLH	0xC3	0xF	0x3	0x3
t _{SCLH}	196x250 ns = 49 µs	16x250 ns = 4.0 µs	4x125 ns = 500 ns	4x125 ns = 500 ns
t _{SCL} ⁽¹⁾	~100 µs ⁽²⁾	~10 µs ⁽²⁾	~2500 ns ⁽³⁾	~2000 ns ⁽⁴⁾
SDADEL	0x2	0x2	0x1	0x0
t _{SDADEL}	2x250 ns = 500 ns	2x250 ns = 500 ns	1x125 ns = 125 ns	0 ns
SCLDEL	0x4	0x4	0x3	0x1
t _{SCLDEL}	5x250 ns = 1250 ns	5x250 ns = 1250 ns	4x125 ns = 500 ns	2x125 ns = 250 ns

1. SCL period t_{SCL} is greater than t_{SCLL} + t_{SCLH} due to SCL internal detection delay. Values provided for t_{SCL} are examples only.

2. t_{SYNC1} + t_{SYNC2} minimum value is 4 × t_{I2CCCLK} = 500 ns. Example with t_{SYNC1} + t_{SYNC2} = 1000 ns.

3. t_{SYNC1} + t_{SYNC2} minimum value is 4 × t_{I2CCCLK} = 500 ns. Example with t_{SYNC1} + t_{SYNC2} = 750 ns.

4. t_{SYNC1} + t_{SYNC2} minimum value is 4 × t_{I2CCCLK} = 500 ns. Example with t_{SYNC1} + t_{SYNC2} = 655 ns.

Table 150. Examples of timings settings for f_{I2CCCLK} = 16 MHz

Parameter	Standard-mode (Sm)		Fast-mode (Fm)	Fast-mode Plus (Fm+)
	10 kHz	100 kHz	400 kHz	1000 kHz
PRESC	3	3	1	0
SCLL	0xC7	0x13	0x9	0x4
t _{SCLL}	200 × 250 ns = 50 µs	20 × 250 ns = 5.0 µs	10 × 125 ns = 1250 ns	5 × 62.5 ns = 312.5 ns
SCLH	0xC3	0xF	0x3	0x2
t _{SCLH}	196 × 250 ns = 49 µs	16 × 250 ns = 4.0 µs	4 × 125 ns = 500 ns	3 × 62.5 ns = 187.5 ns
t _{SCL} ⁽¹⁾	~100 µs ⁽²⁾	~10 µs ⁽²⁾	~2500 ns ⁽³⁾	~1000 ns ⁽⁴⁾
SDADEL	0x2	0x2	0x2	0x0
t _{SDADEL}	2 × 250 ns = 500 ns	2 × 250 ns = 500 ns	2 × 125 ns = 250 ns	0 ns
SCLDEL	0x4	0x4	0x3	0x2
t _{SCLDEL}	5 × 250 ns = 1250 ns	5 × 250 ns = 1250 ns	4 × 125 ns = 500 ns	3 × 62.5 ns = 187.5 ns

1. SCL period t_{SCL} is greater than t_{SCLL} + t_{SCLH} due to SCL internal detection delay. Values provided for t_{SCL} are examples only.

2. $t_{SYNC1} + t_{SYNC2}$ minimum value is $4 \times t_{I2CCLK} = 250$ ns. Example with $t_{SYNC1} + t_{SYNC2} = 1000$ ns.
3. $t_{SYNC1} + t_{SYNC2}$ minimum value is $4 \times t_{I2CCLK} = 250$ ns. Example with $t_{SYNC1} + t_{SYNC2} = 750$ ns.
4. $t_{SYNC1} + t_{SYNC2}$ minimum value is $4 \times t_{I2CCLK} = 250$ ns. Example with $t_{SYNC1} + t_{SYNC2} = 500$ ns.

Table 151. Examples of timings settings for $f_{I2CCLK} = 48$ MHz

Parameter	Standard-mode (Sm)		Fast-mode (Fm)	Fast-mode Plus (Fm+)
	10 kHz	100 kHz	400 kHz	1000 kHz
PRESC	0xB	0xB	5	5
SCLL	0xC7	0x13	0x9	0x3
t_{SCLL}	200×250 ns = 50 μ s	20×250 ns = 5.0 μ s	10×125 ns = 1250 ns	4×125 ns = 500 ns
SCLH	0xC3	0xF	0x3	0x1
t_{SCLH}	196×250 ns = 49 μ s	16×250 ns = 4.0 μ s	4×125 ns = 500 ns	2×125 ns = 250 ns
t_{SCL} ⁽¹⁾	~ 100 μ s ⁽²⁾	~ 10 μ s ⁽²⁾	~ 2500 ns ⁽³⁾	~ 875 ns ⁽⁴⁾
SDADEL	0x2	0x2	0x3	0x0
t_{SDADEL}	2×250 ns = 500 ns	2×250 ns = 500 ns	3×125 ns = 375 ns	0 ns
SCLDEL	0x4	0x4	0x3	0x1
t_{SCLDEL}	5×250 ns = 1250 ns	5×250 ns = 1250 ns	4×125 ns = 500 ns	2×125 ns = 250 ns

1. The SCL period t_{SCL} is greater than $t_{SCLL} + t_{SCLH}$ due to the SCL internal detection delay. Values provided for t_{SCL} are only examples.
2. $t_{SYNC1} + t_{SYNC2}$ minimum value is $4 \times t_{I2CCLK} = 83.3$ ns. Example with $t_{SYNC1} + t_{SYNC2} = 1000$ ns
3. $t_{SYNC1} + t_{SYNC2}$ minimum value is $4 \times t_{I2CCLK} = 83.3$ ns. Example with $t_{SYNC1} + t_{SYNC2} = 750$ ns
4. $t_{SYNC1} + t_{SYNC2}$ minimum value is $4 \times t_{I2CCLK} = 83.3$ ns. Example with $t_{SYNC1} + t_{SYNC2} = 250$ ns

26.4.10 SMBus specific features

This section is relevant only when SMBus feature is supported. Refer to [Section 26.3: I²C implementation](#).

Introduction

The System Management Bus (SMBus) is a two-wire interface through which various devices can communicate with each other and with the rest of the system. It is based on I²C principles of operation. SMBus provides a control bus for system and power management related tasks.

This peripheral is compatible with the SMBUS specification (<http://smbus.org>).

The System Management Bus Specification refers to three types of devices.

- A slave is a device that receives or responds to a command.
- A master is a device that issues commands, generates the clocks and terminates the transfer.
- A host is a specialized master that provides the main interface to the system's CPU. A host must be a master-slave and must support the SMBus host notify protocol. Only one host is allowed in a system.

This peripheral can be configured as master or slave device, and also as a host.

Bus protocols

There are eleven possible command protocols for any given device. A device may use any or all of the eleven protocols to communicate. The protocols are Quick Command, Send Byte, Receive Byte, Write Byte, Write Word, Read Byte, Read Word, Process Call, Block Read, Block Write and Block Write-Block Read Process Call. These protocols should be implemented by the user software.

For more details of these protocols, refer to SMBus specification (<http://smbus.org>).

Address resolution protocol (ARP)

SMBus slave address conflicts can be resolved by dynamically assigning a new unique address to each slave device. In order to provide a mechanism to isolate each device for the purpose of address assignment each device must implement a unique device identifier (UDID). This 128-bit number is implemented by software.

This peripheral supports the Address Resolution Protocol (ARP). The SMBus Device Default Address (0b1100 001) is enabled by setting SMBDEN bit in I2C_CR1 register. The ARP commands should be implemented by the user software.

Arbitration is also performed in slave mode for ARP support.

For more details of the SMBus Address Resolution Protocol, refer to SMBus specification (<http://smbus.org>).

Received Command and Data acknowledge control

A SMBus receiver must be able to NACK each received command or data. In order to allow the ACK control in slave mode, the Slave Byte Control mode must be enabled by setting SBC bit in I2C_CR1 register. Refer to *Slave Byte Control mode on page 833* for more details.

Host Notify protocol

This peripheral supports the Host Notify protocol by setting the SMBHEN bit in the I2C_CR1 register. In this case the host acknowledges the SMBus Host address (0b0001 000).

When this protocol is used, the device acts as a master and the host as a slave.

SMBus alert

The SMBus ALERT optional signal is supported. A slave-only device can signal the host through the SMBALERT# pin that it wants to talk. The host processes the interrupt and simultaneously accesses all SMBALERT# devices through the Alert Response Address (0b0001 100). Only the device(s) which pulled SMBALERT# low acknowledges the Alert Response Address.

When configured as a slave device(SMBHEN=0), the SMBA pin is pulled low by setting the ALERTEN bit in the I2C_CR1 register. The Alert Response Address is enabled at the same time.

When configured as a host (SMBHEN=1), the ALERT flag is set in the I2C_ISR register when a falling edge is detected on the SMBA pin and ALERTEN=1. An interrupt is generated if the ERRIE bit is set in the I2C_CR1 register. When ALERTEN=0, the ALERT line is considered high even if the external SMBA pin is low.

If the SMBus ALERT pin is not needed, the SMBA pin can be used as a standard GPIO if ALERTEN=0.

Packet error checking

A packet error checking mechanism has been introduced in the SMBus specification to improve reliability and communication robustness. Packet Error Checking is implemented by appending a Packet Error Code (PEC) at the end of each message transfer. The PEC is calculated by using the $C(x) = x_8 + x^2 + x + 1$ CRC-8 polynomial on all the message bytes (including addresses and read/write bits).

The peripheral embeds a hardware PEC calculator and allows to send a Not Acknowledge automatically when the received byte does not match with the hardware calculated PEC.

Timeouts

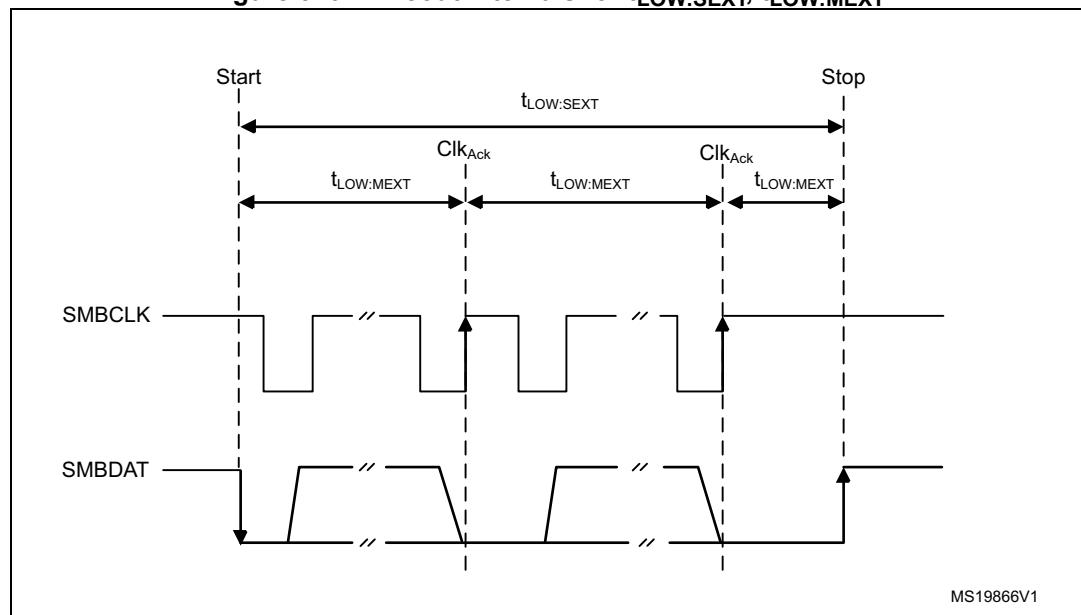
This peripheral embeds hardware timers in order to be compliant with the 3 timeouts defined in SMBus specification.

Table 152. SMBus timeout specifications

Symbol	Parameter	Limits		Unit
		Min	Max	
$t_{TIMEOUT}$	Detect clock low timeout	25	35	ms
$t_{LOW:SEXT}^{(1)}$	Cumulative clock low extend time (slave device)	-	25	ms
$t_{LOW:MEXT}^{(2)}$	Cumulative clock low extend time (master device)	-	10	ms

1. $t_{LOW:SEXT}$ is the cumulative time a given slave device is allowed to extend the clock cycles in one message from the initial START to the STOP. It is possible that, another slave device or the master also extends the clock causing the combined clock low extend time to be greater than $t_{LOW:SEXT}$. Therefore, this parameter is measured with the slave device as the sole target of a full-speed master.
2. $t_{LOW:MEXT}$ is the cumulative time a master device is allowed to extend its clock cycles within each byte of a message as defined from START-to-ACK, ACK-to-ACK, or ACK-to-STOP. It is possible that a slave device or another master also extends the clock causing the combined clock low time to be greater than $t_{LOW:MEXT}$ on a given byte. Therefore, this parameter is measured with a full speed slave device as the sole target of the master.

Figure 310. Timeout intervals for $t_{LOW:SEXT}$, $t_{LOW:MEXT}$



Bus idle detection

A master can assume that the bus is free if it detects that the clock and data signals have been high for t_{IDLE} greater than $t_{HIGH,MAX}$. (refer to [Table 146: I2C-SMBUS specification data setup and hold times](#))

This timing parameter covers the condition where a master has been dynamically added to the bus and may not have detected a state transition on the SMBCLK or SMBDAT lines. In this case, the master must wait long enough to ensure that a transfer is not currently in progress. The peripheral supports a hardware bus idle detection.

26.4.11 SMBus initialization

This section is relevant only when SMBus feature is supported. Refer to [Section 26.3: I2C implementation](#).

In addition to I2C initialization, some other specific initialization must be done in order to perform SMBus communication:

Received Command and Data Acknowledge control (Slave mode)

A SMBus receiver must be able to NACK each received command or data. In order to allow ACK control in slave mode, the Slave Byte Control mode must be enabled by setting the SBC bit in the I2C_CR1 register. Refer to [Slave Byte Control mode on page 833](#) for more details.

Specific address (Slave mode)

The specific SMBus addresses must be enabled if needed. Refer to [Bus idle detection on page 856](#) for more details.

- The SMBus Device Default address (0b1100 001) is enabled by setting the SMBDEN bit in the I2C_CR1 register.
- The SMBus Host address (0b0001 000) is enabled by setting the SMBHEN bit in the I2C_CR1 register.
- The Alert Response Address (0b0001100) is enabled by setting the ALERTEN bit in the I2C_CR1 register.

Packet error checking

PEC calculation is enabled by setting the PECEN bit in the I2C_CR1 register. Then the PEC transfer is managed with the help of a hardware byte counter: NBYTES[7:0] in the I2C_CR2 register. The PECEN bit must be configured before enabling the I2C.

The PEC transfer is managed with the hardware byte counter, so the SBC bit must be set when interfacing the SMBus in slave mode. The PEC is transferred after NBYTES-1 data have been transferred when the PECPBYTE bit is set and the RELOAD bit is cleared. If RELOAD is set, PECPBYTE has no effect.

Caution: Changing the PECEN configuration is not allowed when the I2C is enabled.

Table 153. SMBUS with PEC configuration

Mode	SBC bit	RELOAD bit	AUTOEND bit	PECBYTE bit
Master Tx/Rx NBYTES + PEC+ STOP	x	0	1	1
Master Tx/Rx NBYTES + PEC + ReSTART	x	0	0	1
Slave Tx/Rx with PEC	1	0	x	1

Timeout detection

The timeout detection is enabled by setting the TIMOUTEN and TEXTEN bits in the I2C_TIMEOUTTR register. The timers must be programmed in such a way that they detect a timeout before the maximum time given in the SMBus specification.

- $t_{TIMEOUT}$ check

In order to enable the $t_{TIMEOUT}$ check, the 12-bit TIMEOUTA[11:0] bits must be programmed with the timer reload value in order to check the $t_{TIMEOUT}$ parameter. The TIDLE bit must be configured to '0' in order to detect the SCL low level timeout.

Then the timer is enabled by setting the TIMOUTEN in the I2C_TIMEOUTTR register.

If SCL is tied low for a time greater than $(TIMEOUTA+1) \times 2048 \times t_{I2CCLK}$, the TIMEOUT flag is set in the I2C_ISR register.

Refer to [Table 154: Examples of TIMEOUTA settings for various I2CCLK frequencies \(max \$t_{TIMEOUT} = 25\$ ms\)](#).

Caution: Changing the TIMEOUTA[11:0] bits and TIDLE bit configuration is not allowed when the TIMOUTEN bit is set.

- $t_{LOW:SEXT}$ and $t_{LOW:MEXT}$ check

Depending on if the peripheral is configured as a master or as a slave, The 12-bit TIMEOUTB timer must be configured in order to check $t_{LOW:SEXT}$ for a slave and $t_{LOW:MEXT}$ for a master. As the standard specifies only a maximum, the user can choose the same value for the both.

Then the timer is enabled by setting the TEXTEN bit in the I2C_TIMEOUTTR register.

If the SMBus peripheral performs a cumulative SCL stretch for a time greater than $(TIMEOUTB+1) \times 2048 \times t_{I2CCLK}$, and in the timeout interval described in [Bus idle detection on page 856](#) section, the TIMEOUT flag is set in the I2C_ISR register.

Refer to [Table 155: Examples of TIMEOUTB settings for various I2CCLK frequencies](#)

Caution: Changing the TIMEOUTB configuration is not allowed when the TEXTEN bit is set.

Bus Idle detection

In order to enable the t_{IDLE} check, the 12-bit TIMEOUTA[11:0] field must be programmed with the timer reload value in order to obtain the t_{IDLE} parameter. The TIDLE bit must be configured to '1' in order to detect both SCL and SDA high level timeout.

Then the timer is enabled by setting the TIMOUTEN bit in the I2C_TIMEOUTTR register.

If both the SCL and SDA lines remain high for a time greater than $(TIMEOUTA+1) \times 4 \times t_{I2CCLK}$, the TIMEOUT flag is set in the I2C_ISR register.

Refer to [Table 156: Examples of TIMEOUTA settings for various I2CCLK frequencies \(max \$t_{IDLE} = 50\$ \$\mu\$ s\)](#)

Caution: Changing the TIMEOUTA and TIDLE configuration is not allowed when the TIMEOUTEN is set.

26.4.12 SMBus: I2C_TIMEOUTR register configuration examples

This section is relevant only when SMBus feature is supported. Refer to [Section 26.3: I2C implementation](#).

- Configuring the maximum duration of $t_{TIMEOUT}$ to 25 ms:

**Table 154. Examples of TIMEOUTA settings for various I2CCLK frequencies
(max $t_{TIMEOUT} = 25$ ms)**

f_{I2CCLK}	TIMEOUTA[11:0] bits	TIDLE bit	TIMEOUTEN bit	$t_{TIMEOUT}$
8 MHz	0x61	0	1	$98 \times 2048 \times 125$ ns = 25 ms
16 MHz	0xC3	0	1	$196 \times 2048 \times 62.5$ ns = 25 ms
32 MHz	0x186	0	1	$391 \times 2048 \times 31.25$ ns = 25 ms

- Configuring the maximum duration of $t_{LOW:SEXT}$ and $t_{LOW:MEXT}$ to 8 ms:

Table 155. Examples of TIMEOUTB settings for various I2CCLK frequencies

f_{I2CCLK}	TIMEOUTB[11:0] bits	TEXEN bit	$t_{LOW:EXT}$
8 MHz	0x1F	1	$32 \times 2048 \times 125$ ns = 8 ms
16 MHz	0x3F	1	$64 \times 2048 \times 62.5$ ns = 8 ms
32 MHz	0x7C	1	$125 \times 2048 \times 31.25$ ns = 8 ms

- Configuring the maximum duration of t_{IDLE} to 50 μ s

**Table 156. Examples of TIMEOUTA settings for various I2CCLK frequencies
(max $t_{IDLE} = 50$ μ s)**

f_{I2CCLK}	TIMEOUTA[11:0] bits	TIDLE bit	TIMEOUTEN bit	t_{TIDLE}
8 MHz	0x63	1	1	$100 \times 4 \times 125$ ns = 50 μ s
16 MHz	0xC7	1	1	$200 \times 4 \times 62.5$ ns = 50 μ s
32 MHz	0x18F	1	1	$400 \times 4 \times 31.25$ ns = 50 μ s

26.4.13 SMBus slave mode

This section is relevant only when SMBus feature is supported. Refer to [Section 26.3: I2C implementation](#).

In addition to I2C slave transfer management (refer to [Section 26.4.7: I2C slave mode](#)) some additional software flowcharts are provided to support SMBus.

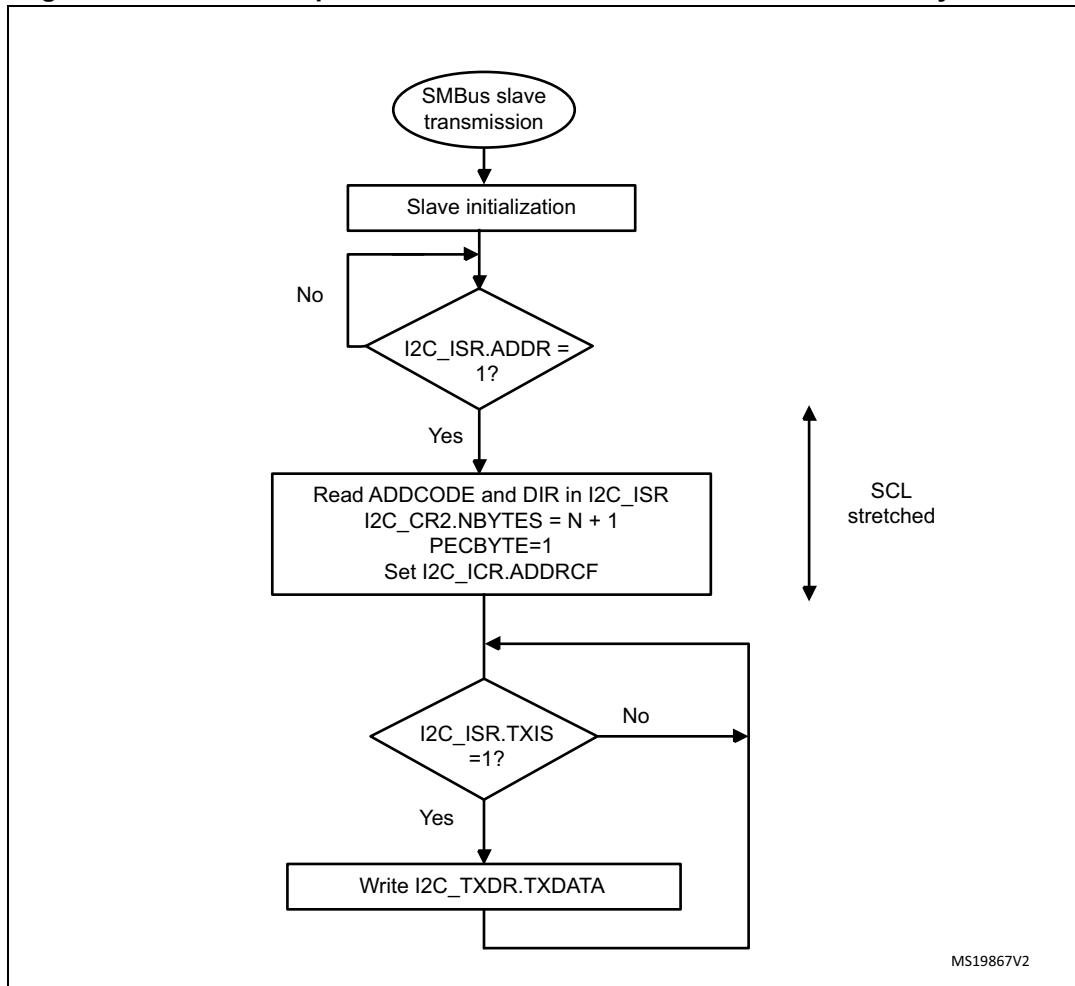
SMBus Slave transmitter

When the IP is used in SMBus, SBC must be programmed to '1' in order to allow the PEC transmission at the end of the programmed number of data bytes. When the PECPBYTE bit is set, the number of bytes programmed in NBYTES[7:0] includes the PEC transmission. In

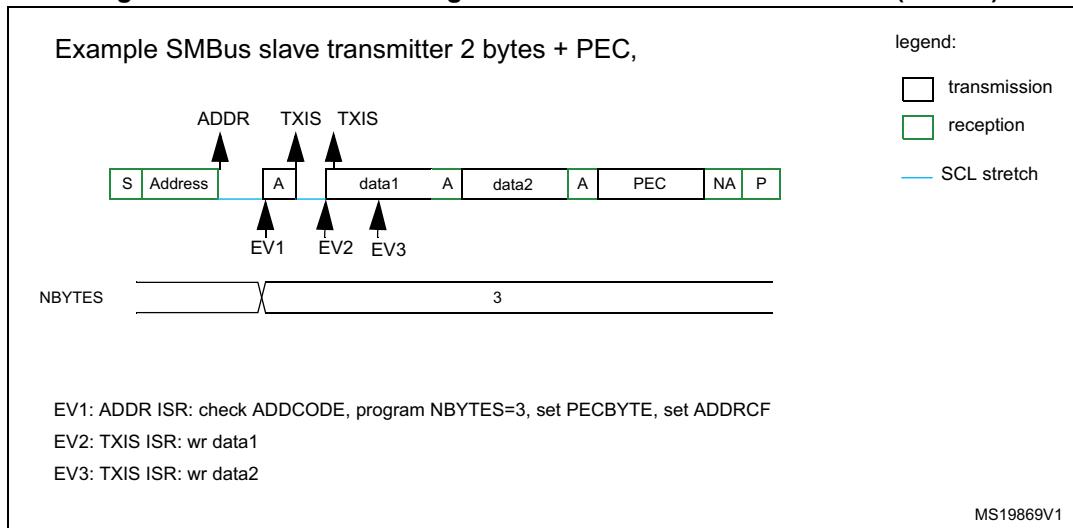
that case the total number of TXIS interrupts is NBYTES-1 and the content of the I2C_PECR register is automatically transmitted if the master requests an extra byte after the NBYTES-1 data transfer.

Caution: The PECBYTE bit has no effect when the RELOAD bit is set.

Figure 311. Transfer sequence flowchart for SMBus slave transmitter N bytes + PEC



MS19867V2

Figure 312. Transfer bus diagrams for SMBus slave transmitter (SBC=1)

SMBus Slave receiver

When the I2C is used in SMBus mode, SBC must be programmed to '1' in order to allow the PEC checking at the end of the programmed number of data bytes. In order to allow the ACK control of each byte, the reload mode must be selected (RELOAD=1). Refer to [Slave Byte Control mode on page 833](#) for more details.

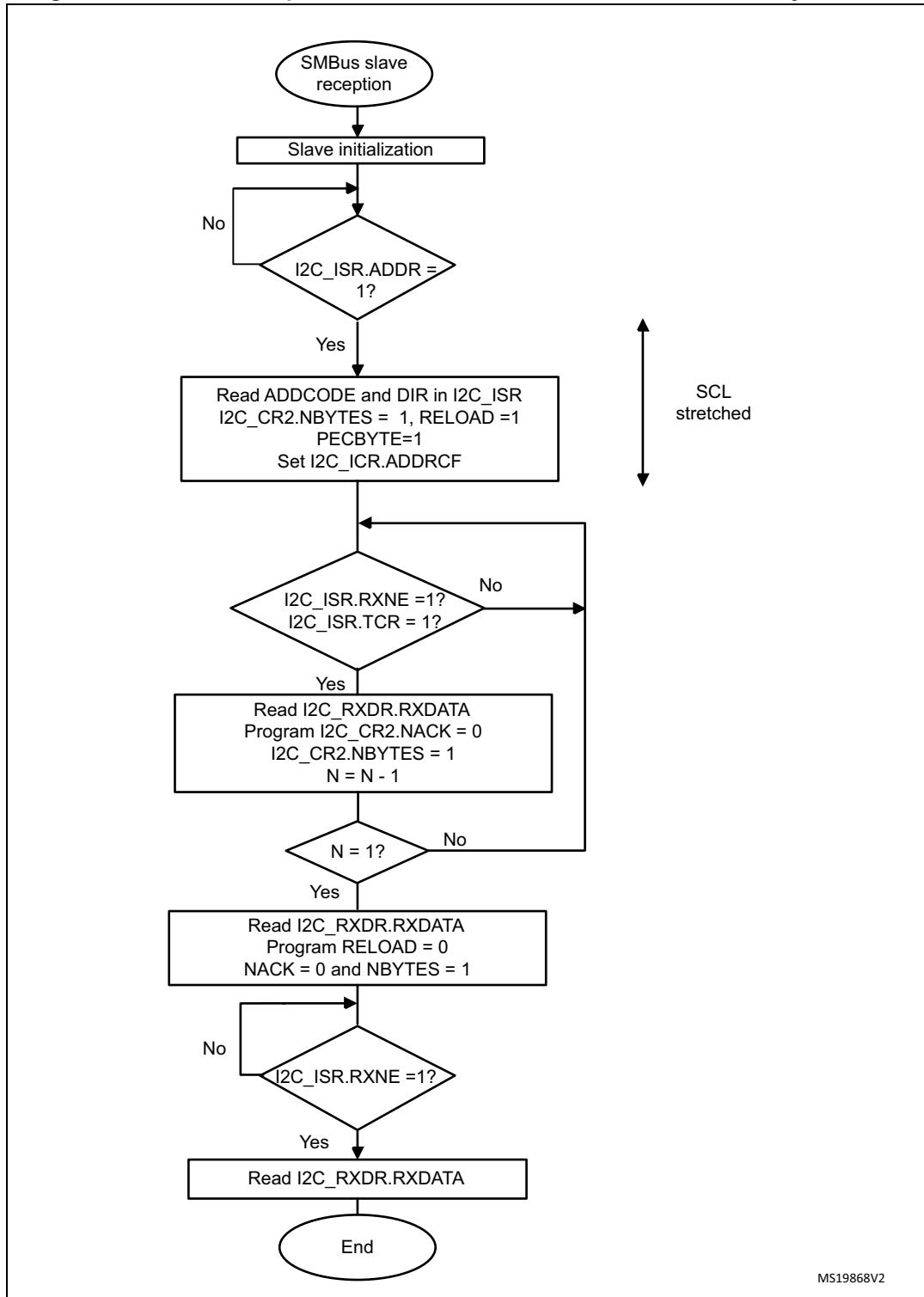
In order to check the PEC byte, the RELOAD bit must be cleared and the PECBYTE bit must be set. In this case, after NBYTES-1 data have been received, the next received byte is compared with the internal I2C_PECR register content. A NACK is automatically generated if the comparison does not match, and an ACK is automatically generated if the comparison matches, whatever the ACK bit value. Once the PEC byte is received, it is copied into the I2C_RXDR register like any other data, and the RXNE flag is set.

In the case of a PEC mismatch, the PECERR flag is set and an interrupt is generated if the ERRIE bit is set in the I2C_CR1 register.

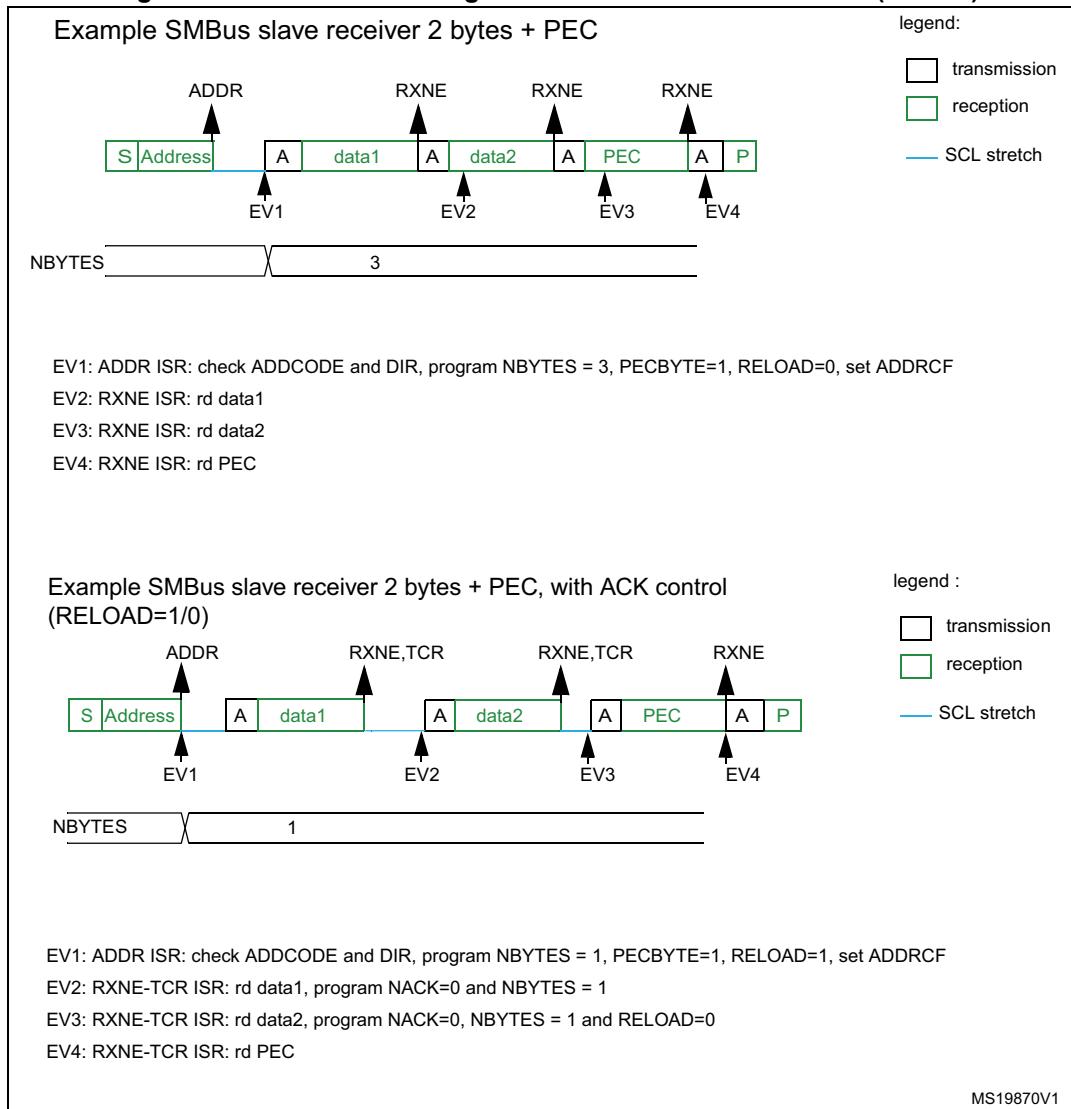
If no ACK software control is needed, the user can program PECBYTE=1 and, in the same write operation, program NBYTES with the number of bytes to be received in a continuous flow. After NBYTES-1 are received, the next received byte is checked as being the PEC.

Caution: The PECBYTE bit has no effect when the RELOAD bit is set.

Figure 313. Transfer sequence flowchart for SMBus slave receiver N Bytes + PEC



MS19868V2

Figure 314. Bus transfer diagrams for SMBus slave receiver (SBC=1)

This section is relevant only when SMBus feature is supported. Refer to [Section 26.3: I2C implementation](#).

In addition to I2C master transfer management (refer to [Section 26.4.8: I2C master mode](#)) some additional software flowcharts are provided to support SMBus.

SMBus Master transmitter

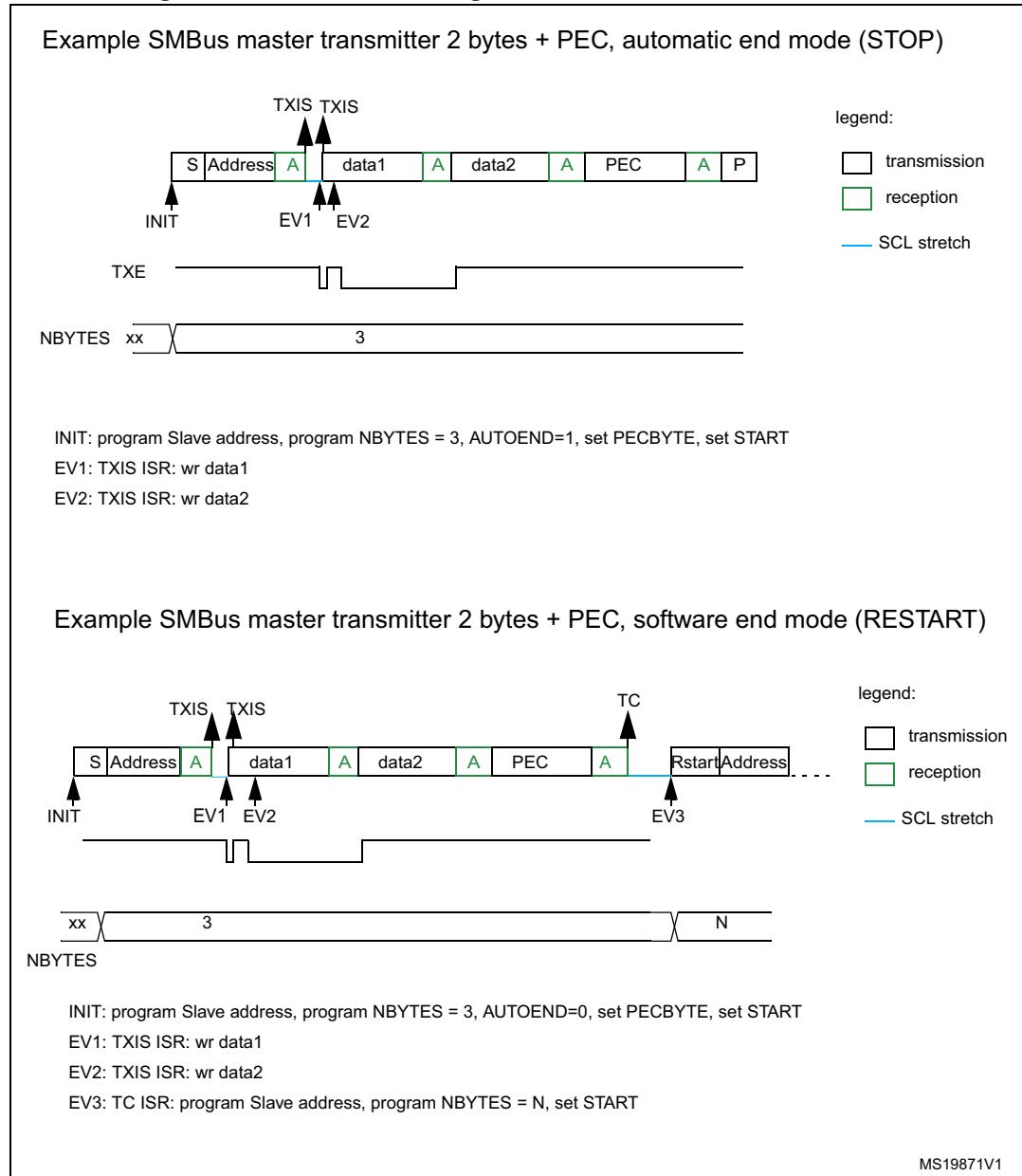
When the SMBus master wants to transmit the PEC, the PECBYTE bit must be set and the number of bytes must be programmed in the NBYTES[7:0] field, before setting the START bit. In this case the total number of TXIS interrupts is NBYTES-1. So if the PECBYTE bit is set when NBYTES=0x1, the content of the I2C_PECR register is automatically transmitted.

If the SMBus master wants to send a STOP condition after the PEC, automatic end mode must be selected (AUTOEND=1). In this case, the STOP condition automatically follows the PEC transmission.

When the SMBus master wants to send a RESTART condition after the PEC, software mode must be selected (AUTOEND=0). In this case, once NBYTES-1 have been transmitted, the I2C_PECR register content is transmitted and the TC flag is set after the PEC transmission, stretching the SCL line low. The RESTART condition must be programmed in the TC interrupt subroutine.

Caution: The PECBYTE bit has no effect when the RELOAD bit is set.

Figure 315. Bus transfer diagrams for SMBus master transmitter



MS19871V1

SMBus Master receiver

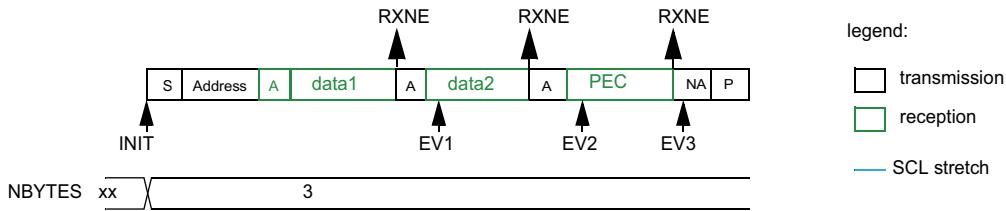
When the SMBus master wants to receive the PEC followed by a STOP at the end of the transfer, automatic end mode can be selected (AUTOEND=1). The PECBYTE bit must be set and the slave address must be programmed, before setting the START bit. In this case, after NBYTES-1 data have been received, the next received byte is automatically checked versus the I2C_PECR register content. A NACK response is given to the PEC byte, followed by a STOP condition.

When the SMBus master receiver wants to receive the PEC byte followed by a RESTART condition at the end of the transfer, software mode must be selected (AUTOEND=0). The PECBYTE bit must be set and the slave address must be programmed, before setting the START bit. In this case, after NBYTES-1 data have been received, the next received byte is automatically checked versus the I2C_PECR register content. The TC flag is set after the PEC byte reception, stretching the SCL line low. The RESTART condition can be programmed in the TC interrupt subroutine.

Caution: The PECBYTE bit has no effect when the RELOAD bit is set.

Figure 316. Bus transfer diagrams for SMBus master receiver

Example SMBus master receiver 2 bytes + PEC, automatic end mode (STOP)



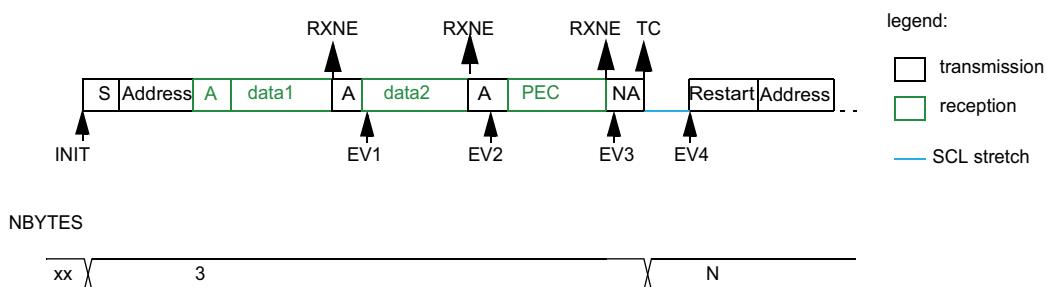
INIT: program Slave address, program NBYTES = 3, AUTOEND=1, set PECBYTE, set START

EV1: RXNE ISR: rd data1

EV2: RXNE ISR: rd data2

EV3: RXNE ISR: rd PEC

Example SMBus master receiver 2 bytes + PEC, software end mode (RESTART)



INIT: program Slave address, program NBYTES = 3, AUTOEND=0, set PECBYTE, set START

EV1: RXNE ISR: rd data1

EV2: RXNE ISR: rd data2

EV3: RXNE ISR: read PEC

EV4: TC ISR: program Slave address, program NBYTES = N, set START

MS19872V1

26.4.14 Error conditions

The following are the error conditions which may cause communication to fail.

Bus error (BERR)

A bus error is detected when a START or a STOP condition is detected and is not located after a multiple of 9 SCL clock pulses. A START or a STOP condition is detected when a SDA edge occurs while SCL is high.

The bus error flag is set only if the I2C is involved in the transfer as master or addressed slave (i.e not during the address phase in slave mode).

In case of a misplaced START or RESTART detection in slave mode, the I2C enters address recognition state like for a correct START condition.

When a bus error is detected, the BERR flag is set in the I2C_ISR register, and an interrupt is generated if the ERRIE bit is set in the I2C_CR1 register.

Arbitration lost (ARLO)

An arbitration loss is detected when a high level is sent on the SDA line, but a low level is sampled on the SCL rising edge.

- In master mode, arbitration loss is detected during the address phase, data phase and data acknowledge phase. In this case, the SDA and SCL lines are released, the START control bit is cleared by hardware and the master switches automatically to slave mode.
- In slave mode, arbitration loss is detected during data phase and data acknowledge phase. In this case, the transfer is stopped, and the SCL and SDA lines are released.

When an arbitration loss is detected, the ARLO flag is set in the I2C_ISR register, and an interrupt is generated if the ERRIE bit is set in the I2C_CR1 register.

Overrun/underrun error (OVR)

An overrun or underrun error is detected in slave mode when NOSTRETCH=1 and:

- In reception when a new byte is received and the RXDR register has not been read yet. The new received byte is lost, and a NACK is automatically sent as a response to the new byte.
- In transmission:
 - When STOPF=1 and the first data byte should be sent. The content of the I2C_TXDR register is sent if TXE=0, 0xFF if not.
 - When a new byte must be sent and the I2C_TXDR register has not been written yet, 0xFF is sent.

When an overrun or underrun error is detected, the OVR flag is set in the I2C_ISR register, and an interrupt is generated if the ERRIE bit is set in the I2C_CR1 register.

Packet Error Checking Error (PECERR)

This section is relevant only when the SMBus feature is supported. Refer to [Section 26.3: I2C implementation](#).

A PEC error is detected when the received PEC byte does not match with the I2C_PECR register content. A NACK is automatically sent after the wrong PEC reception.

When a PEC error is detected, the PECERR flag is set in the I2C_ISR register, and an interrupt is generated if the ERRIE bit is set in the I2C_CR1 register.

Timeout Error (TIMEOUT)

This section is relevant only when the SMBus feature is supported. Refer to [Section 26.3: I2C implementation](#).

A timeout error occurs for any of these conditions:

- TIDLE=0 and SCL remained low for the time defined in the TIMEOUTA[11:0] bits: this is used to detect a SMBus timeout.
- TIDLE=1 and both SDA and SCL remained high for the time defined in the TIMEOUTA [11:0] bits: this is used to detect a bus idle condition.
- Master cumulative clock low extend time reached the time defined in the TIMEOUTB[11:0] bits (SMBus $t_{LOW:MEXT}$ parameter)
- Slave cumulative clock low extend time reached the time defined in TIMEOUTB[11:0] bits (SMBus $t_{LOW:SEXT}$ parameter)

When a timeout violation is detected in master mode, a STOP condition is automatically sent.

When a timeout violation is detected in slave mode, SDA and SCL lines are automatically released.

When a timeout error is detected, the TIMEOUT flag is set in the I2C_ISR register, and an interrupt is generated if the ERRIE bit is set in the I2C_CR1 register.

Alert (ALERT)

This section is relevant only when the SMBus feature is supported. Refer to [Section 26.3: I2C implementation](#).

The ALERT flag is set when the I2C interface is configured as a Host (SMBHEN=1), the alert pin detection is enabled (ALERTEN=1) and a falling edge is detected on the SMBA pin. An interrupt is generated if the ERRIE bit is set in the I2C_CR1 register.

26.4.15 DMA requests

Transmission using DMA

DMA (Direct Memory Access) can be enabled for transmission by setting the TXDMAEN bit in the I2C_CR1 register. Data is loaded from an SRAM area configured using the DMA peripheral (see [Section 8: Direct memory access controller \(DMA\) on page 217](#)) to the I2C_TXDR register whenever the TXIS bit is set.

Only the data are transferred with DMA.

- In master mode: the initialization, the slave address, direction, number of bytes and START bit are programmed by software (the transmitted slave address cannot be transferred with DMA). When all data are transferred using DMA, the DMA must be initialized before setting the START bit. The end of transfer is managed with the

NBYTES counter. Refer to [Master transmitter on page 844](#).

- In slave mode:
 - With NOSTRETCH=0, when all data are transferred using DMA, the DMA must be initialized before the address match event, or in ADDR interrupt subroutine, before clearing ADDR.
 - With NOSTRETCH=1, the DMA must be initialized before the address match event.
- For instances supporting SMBus: the PEC transfer is managed with NBYTES counter. Refer to [SMBus Slave transmitter on page 858](#) and [SMBus Master transmitter on page 862](#).

Note: If DMA is used for transmission, the TXIE bit does not need to be enabled.

Reception using DMA

DMA (Direct Memory Access) can be enabled for reception by setting the RXDMAEN bit in the I2C_CR1 register. Data is loaded from the I2C_RXDR register to an SRAM area configured using the DMA peripheral (refer to [Section 8: Direct memory access controller \(DMA\)](#)) whenever the RXNE bit is set. Only the data (including PEC) are transferred with DMA.

- In master mode, the initialization, the slave address, direction, number of bytes and START bit are programmed by software. When all data are transferred using DMA, the DMA must be initialized before setting the START bit. The end of transfer is managed with the NBYTES counter.
- In slave mode with NOSTRETCH=0, when all data are transferred using DMA, the DMA must be initialized before the address match event, or in the ADDR interrupt subroutine, before clearing the ADDR flag.
- If SMBus is supported (see [Section 26.3: I2C implementation](#)): the PEC transfer is managed with the NBYTES counter. Refer to [SMBus Slave receiver on page 860](#) and [SMBus Master receiver on page 864](#).

Note: If DMA is used for reception, the RXIE bit does not need to be enabled.

26.4.16 Debug mode

When the microcontroller enters debug mode (core halted), the SMBus timeout either continues to work normally or stops, depending on the DBG_I2Cx_STOP configuration bits in the DBG module.

26.5 I2C low-power modes

Table 157. Effect of low-power modes on the I2C

Mode	Description
Sleep	No effect I2C interrupts cause the device to exit the Sleep mode.
Stop	The contents of I2C registers are kept.
Standby	The I2C peripheral is powered down and must be reinitialized after exiting Standby.

26.6 I2C interrupts

The table below gives the list of I2C interrupt requests.

Table 158. I2C Interrupt requests

Interrupt acronym	Interrupt event	Event flag	Enable control bit	Interrupt clear method	Exit the Sleep mode	Exit the Stop mode
I2C	I2C_EV	Receive buffer not empty	RXNE	RXIE	Read I2C_RXDR register	Yes No
		Transmit buffer interrupt status	TXIS	TXIE	Write I2C_TXDR register	
		Stop detection interrupt flag	STOPF	STOPIE	Write STOPCF=1	
		Transfer Complete Reload	TCR	TCIE	Write I2C_CR2 with NBYTES[7:0] ≠ 0	
		Transfer complete	TC		Write START=1 or STOP=1	
		Address matched	ADDR	ADDRIE	Write ADDRCF=1	
		NACK reception	NACKF	NACKIE	Write NACKCF=1	
I2C	I2C_ER	Bus error	BERR	ERRIE	Write BERRCF=1	Yes No
		Arbitration loss	ARLO		Write ARLOCF=1	
		Overrun/Underrun	OVR		Write OVRCF=1	
		PEC error	PECERR		Write PECERRCF=1	
		Timeout/t _{LOW} error	TIMEOUT		Write TIMEOUTCF=1	
		SMBus Alert	ALERT		Write ALERTCF=1	

26.7 I2C registers

Refer to [Section 1.2 on page 53](#) for a list of abbreviations used in register descriptions.

The peripheral registers are accessed by words (32-bit).

26.7.1 I2C2 control register 1 (I2C_CR1)

Address offset: 0x00

Reset value: 0x0000 0000

Access: No wait states, except if a write access occurs while a write access to this register is ongoing. In this case, wait states are inserted in the second write access until the previous

one is completed. The latency of the second write access can be up to $2 \times \text{PCLK1} + 6 \times \text{I2CCLK}$.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PECEN	ALERT EN	SMBD EN	SMBH EN	GCEN	Res.	NOSTR ETCH	SBC
								rw	rw	rw	rw	rw		rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RXDMA EN	TXDMA EN	Res.	ANF OFF	DNF[3:0]				ERRIE	TCIE	STOP IE	NACK IE	ADDR IE	RXIE	TXIE	PE
rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bit 23 **PECEN**: PEC enable

- 0: PEC calculation disabled
- 1: PEC calculation enabled

Note: If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'. Refer to [Section 26.3: I2C implementation](#).

Bit 22 **ALERTEN**: SMBus alert enable

Device mode (SMBHEN=0):

- 0: Releases SMBA pin high and Alert Response Address Header disabled: 0001100x followed by NACK.
- 1: Drives SMBA pin low and Alert Response Address Header enables: 0001100x followed by ACK.

Host mode (SMBHEN=1):

- 0: SMBus Alert pin (SMBA) not supported.
- 1: SMBus Alert pin (SMBA) supported.

Note: When ALERTEN=0, the SMBA pin can be used as a standard GPIO.

If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'. Refer to [Section 26.3: I2C implementation](#).

Bit 21 **SMBDEN**: SMBus Device Default address enable

- 0: Device default address disabled. Address 0b1100001x is NACKed.
- 1: Device default address enabled. Address 0b1100001x is ACKed.

Note: If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'. Refer to [Section 26.3: I2C implementation](#).

Bit 20 **SMBHEN**: SMBus Host address enable

- 0: Host address disabled. Address 0b0001000x is NACKed.
- 1: Host address enabled. Address 0b0001000x is ACKed.

Note: If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'. Refer to [Section 26.3: I2C implementation](#).

Bit 19 **GCEN**: General call enable

- 0: General call disabled. Address 0b00000000 is NACKed.
- 1: General call enabled. Address 0b00000000 is ACKed.

Bit 18 Reserved, must be kept at reset value.

Bit 17 NOSTRETCH: Clock stretching disable

This bit is used to disable clock stretching in slave mode. It must be kept cleared in master mode.

- 0: Clock stretching enabled
- 1: Clock stretching disabled

Note: This bit can only be programmed when the I2C is disabled (PE = 0).

Bit 16 SBC: Slave byte control

This bit is used to enable hardware byte control in slave mode.

- 0: Slave byte control disabled
- 1: Slave byte control enabled

Bit 15 RXDMAEN: DMA reception requests enable

- 0: DMA mode disabled for reception
- 1: DMA mode enabled for reception

Bit 14 TXDMAEN: DMA transmission requests enable

- 0: DMA mode disabled for transmission
- 1: DMA mode enabled for transmission

Bit 13 Reserved, must be kept at reset value.

Bit 12 ANFOFF: Analog noise filter OFF

- 0: Analog noise filter enabled
- 1: Analog noise filter disabled

Note: This bit can only be programmed when the I2C is disabled (PE = 0).

Bits 11:8 DNF[3:0]: Digital noise filter

These bits are used to configure the digital noise filter on SDA and SCL input. The digital filter, filters spikes with a length of up to $DNF[3:0] * t_{I2CCLK}$

0000: Digital filter disabled

0001: Digital filter enabled and filtering capability up to 1 t_{I2CCLK}

.....
1111: digital filter enabled and filtering capability up to 15 t_{I2CCLK}

Note: If the analog filter is also enabled, the digital filter is added to the analog filter.

This filter can only be programmed when the I2C is disabled (PE = 0).

Bit 7 ERRIE: Error interrupts enable

- 0: Error detection interrupts disabled
- 1: Error detection interrupts enabled

Note: Any of these errors generate an interrupt:

Arbitration Loss (ARLO)

Bus Error detection (BERR)

Overrun/Underrun (OVR)

Timeout detection (TIMEOUT)

PEC error detection (PECERR)

Alert pin event detection (ALERT)

Bit 6 TCIE: Transfer Complete interrupt enable

- 0: Transfer Complete interrupt disabled
- 1: Transfer Complete interrupt enabled

Note: Any of these events generate an interrupt:

Transfer Complete (TC)

Transfer Complete Reload (TCR)

Bit 5 **STOPIE**: Stop detection Interrupt enable

- 0: Stop detection (STOPF) interrupt disabled
- 1: Stop detection (STOPF) interrupt enabled

Bit 4 **NACKIE**: Not acknowledge received Interrupt enable

- 0: Not acknowledge (NACKF) received interrupts disabled
- 1: Not acknowledge (NACKF) received interrupts enabled

Bit 3 **ADDRIE**: Address match Interrupt enable (slave only)

- 0: Address match (ADDR) interrupts disabled
- 1: Address match (ADDR) interrupts enabled

Bit 2 **RXIE**: RX Interrupt enable

- 0: Receive (RXNE) interrupt disabled
- 1: Receive (RXNE) interrupt enabled

Bit 1 **TXIE**: TX Interrupt enable

- 0: Transmit (TXIS) interrupt disabled
- 1: Transmit (TXIS) interrupt enabled

Bit 0 **PE**: Peripheral enable

- 0: Peripheral disable
- 1: Peripheral enable

Note: When PE=0, the I2C SCL and SDA lines are released. Internal state machines and status bits are put back to their reset value. When cleared, PE must be kept low for at least 3 APB clock cycles.

26.7.2 I2C2 control register 2 (I2C_CR2)

Address offset: 0x0C4

Reset value: 0x0000 0000

Access: No wait states, except if a write access occurs while a write access to this register is ongoing. In this case, wait states are inserted in the second write access until the previous one is completed. The latency of the second write access can be up to 2 x PCLK1 + 6 x I2CCLK.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	PEC BYTE	AUTOE ND	RE LOAD	NBYTES[7:0]							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NACK	STOP	START	HEAD1 OR	ADD10	RD_	WRN	SADD[9:0]								
rs	rs	rs	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bit 26 **PECBYTE**: Packet error checking byte

This bit is set by software, and cleared by hardware when the PEC is transferred, or when a STOP condition or an Address matched is received, also when PE=0.

0: No PEC transfer.

1: PEC transmission/reception is requested

Note: Writing '0' to this bit has no effect.

This bit has no effect when RELOAD is set.

This bit has no effect in slave mode when SBC=0.

If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'. Refer to [Section 26.3: I2C implementation](#).

Bit 25 **AUTOEND**: Automatic end mode (master mode)

This bit is set and cleared by software.

0: software end mode: TC flag is set when NBYTES data are transferred, stretching SCL low.
1: Automatic end mode: a STOP condition is automatically sent when NBYTES data are transferred.

Note: This bit has no effect in slave mode or when the RELOAD bit is set.

Bit 24 **RELOAD**: NBYTES reload mode

This bit is set and cleared by software.

0: The transfer is completed after the NBYTES data transfer (STOP or RESTART follows).
1: The transfer is not completed after the NBYTES data transfer (NBYTES is reloaded). TCR flag is set when NBYTES data are transferred, stretching SCL low.

Bits 23:16 **NBYTES[7:0]**: Number of bytes

The number of bytes to be transmitted/received is programmed there. This field is don't care in slave mode with SBC=0.

Note: Changing these bits when the START bit is set is not allowed.

Bit 15 **NACK**: NACK generation (slave mode)

The bit is set by software, cleared by hardware when the NACK is sent, or when a STOP condition or an Address matched is received, or when PE=0.

0: an ACK is sent after current received byte.

1: a NACK is sent after current received byte.

Note: Writing '0' to this bit has no effect.

This bit is used in slave mode only: in master receiver mode, NACK is automatically generated after last byte preceding STOP or RESTART condition, whatever the NACK bit value.

When an overrun occurs in slave receiver NOSTRETCH mode, a NACK is automatically generated whatever the NACK bit value.

When hardware PEC checking is enabled (PECBYTE=1), the PEC acknowledge value does not depend on the NACK value.

Bit 14 **STOP**: Stop generation (master mode)

The bit is set by software, cleared by hardware when a STOP condition is detected, or when PE = 0.

In Master Mode:

0: No Stop generation.

1: Stop generation after current byte transfer.

Note: Writing '0' to this bit has no effect.

Bit 13 **START**: Start generation

This bit is set by software, and cleared by hardware after the Start followed by the address sequence is sent, by an arbitration loss, by a timeout error detection, or when PE = 0. It can also be cleared by software by writing '1' to the ADDRCF bit in the I2C_ICR register.

0: No Start generation.

1: Restart/Start generation:

If the I2C is already in master mode with AUTOEND = 0, setting this bit generates a Repeated Start condition when RELOAD=0, after the end of the NBYTES transfer.

Otherwise setting this bit generates a START condition once the bus is free.

Note: Writing '0' to this bit has no effect.

The START bit can be set even if the bus is BUSY or I2C is in slave mode.

This bit has no effect when RELOAD is set.

Bit 12 **HEAD10R**: 10-bit address header only read direction (master receiver mode)

0: The master sends the complete 10 bit slave address read sequence: Start + 2 bytes 10bit address in write direction + Restart + 1st 7 bits of the 10 bit address in read direction.

1: The master only sends the 1st 7 bits of the 10 bit address, followed by Read direction.

Note: Changing this bit when the START bit is set is not allowed.

Bit 11 **ADD10**: 10-bit addressing mode (master mode)

0: The master operates in 7-bit addressing mode,

1: The master operates in 10-bit addressing mode

Note: Changing this bit when the START bit is set is not allowed.

Bit 10 **RD_WRN**: Transfer direction (master mode)

0: Master requests a write transfer.

1: Master requests a read transfer.

Note: Changing this bit when the START bit is set is not allowed.

Bits 9:8 **SADD[9:8]**: Slave address bit 9:8 (master mode)

In 7-bit addressing mode (ADD10 = 0):

These bits are don't care

In 10-bit addressing mode (ADD10 = 1):

These bits should be written with bits 9:8 of the slave address to be sent

Note: Changing these bits when the START bit is set is not allowed.

Bits 7:1 **SADD[7:1]**: Slave address bit 7:1 (master mode)

In 7-bit addressing mode (ADD10 = 0):

These bits should be written with the 7-bit slave address to be sent

In 10-bit addressing mode (ADD10 = 1):

These bits should be written with bits 7:1 of the slave address to be sent.

Note: Changing these bits when the START bit is set is not allowed.

Bit 0 **SADD0**: Slave address bit 0 (master mode)

In 7-bit addressing mode (ADD10 = 0):

This bit is don't care

In 10-bit addressing mode (ADD10 = 1):

This bit should be written with bit 0 of the slave address to be sent

Note: Changing these bits when the START bit is set is not allowed.

26.7.3 I2C2 own address 1 register (I2C_OAR1)

Address offset: 0x08

Reset value: 0x0000 0000

Access: No wait states, except if a write access occurs while a write access to this register is ongoing. In this case, wait states are inserted in the second write access until the previous one is completed. The latency of the second write access can be up to 2 x PCLK1 + 6 x I2CCLK.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OA1EN	Res.	Res.	Res.	Res.	OA1 MODE	OA1[9:8]									OA1[0]
rw					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bit 15 **OA1EN**: Own Address 1 enable

- 0: Own address 1 disabled. The received slave address OA1 is NACKed.
- 1: Own address 1 enabled. The received slave address OA1 is ACKed.

Bits 14:11 Reserved, must be kept at reset value.

Bit 10 **OA1MODE**: Own Address 1 10-bit mode

- 0: Own address 1 is a 7-bit address.
- 1: Own address 1 is a 10-bit address.

Note: This bit can be written only when OA1EN=0.

Bits 9:8 **OA1[9:8]**: Interface address

- 7-bit addressing mode: do not care
- 10-bit addressing mode: bits 9:8 of address

Note: These bits can be written only when OA1EN=0.

Bits 7:1 **OA1[7:1]**: Interface address

- 7-bit addressing mode: 7-bit address
- 10-bit addressing mode: bits 7:1 of 10-bit address

Note: These bits can be written only when OA1EN=0.

Bit 0 **OA1[0]**: Interface address

- 7-bit addressing mode: do not care
- 10-bit addressing mode: bit 0 of address

Note: This bit can be written only when OA1EN=0.

26.7.4 I2C2 own address 2 register (I2C_OAR2)

Address offset: 0x0C

Reset value: 0x0000 0000

Access: No wait states, except if a write access occurs while a write access to this register is ongoing. In this case, wait states are inserted in the second write access until the previous one is completed. The latency of the second write access can be up to $2 \times \text{PCLK1} + 6 \times \text{I2CCLK}$.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OA2EN	Res.	Res.	Res.	Res.	OA2MSK[2:0]			OA2[7:1]							
rw					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bit 15 **OA2EN**: Own Address 2 enable

- 0: Own address 2 disabled. The received slave address OA2 is NACKed.
- 1: Own address 2 enabled. The received slave address OA2 is ACKed.

Bits 14:11 Reserved, must be kept at reset value.

Bits 10:8 **OA2MSK[2:0]**: Own Address 2 masks

- 000: No mask
- 001: OA2[1] is masked and don't care. Only OA2[7:2] are compared.
- 010: OA2[2:1] are masked and don't care. Only OA2[7:3] are compared.
- 011: OA2[3:1] are masked and don't care. Only OA2[7:4] are compared.
- 100: OA2[4:1] are masked and don't care. Only OA2[7:5] are compared.
- 101: OA2[5:1] are masked and don't care. Only OA2[7:6] are compared.
- 110: OA2[6:1] are masked and don't care. Only OA2[7] is compared.
- 111: OA2[7:1] are masked and don't care. No comparison is done, and all (except reserved) 7-bit received addresses are acknowledged.

Note: These bits can be written only when OA2EN=0.

As soon as OA2MSK is not equal to 0, the reserved I2C addresses (0b0000xxx and 0b1111xxx) are not acknowledged even if the comparison matches.

Bits 7:1 **OA2[7:1]**: Interface address

7-bit addressing mode: 7-bit address

Note: These bits can be written only when OA2EN=0.

Bit 0 Reserved, must be kept at reset value.

26.7.5 I2C2 timing register (I2C_TIMINGR)

Address offset: 0x10

Reset value: 0x0000 0000

Access: No wait states

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

Bits 31:28 **PRESC[3:0]**: Timing prescaler

This field is used to prescale I2CCLK in order to generate the clock period t_{PRESC} used for data setup and hold counters (refer to [I2C timings on page 825](#)) and for SCL high and low level counters (refer to [I2C master initialization on page 840](#)).

$$t_{PRESC} = (\text{PRESC}+1) \times t_{I2CCLK}$$

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

Bits 23:20 **SCLDEL[3:0]**: Data setup time

This field is used to generate a delay t_{SCLDEL} between SDA edge and SCL rising edge. In master mode and in slave mode with NOSTRETCH = 0, the SCL line is stretched low during t_{SCLDEL} .

$$t_{SCLDEL} = (\text{SCLDEL}+1) \times t_{PRESC}$$

Note: t_{SCLDEL} is used to generate $t_{SU:DAT}$ timing.

Bits 19:16 **SDADEL[3:0]**: Data hold time

This field is used to generate the delay t_{SDADEL} between SCL falling edge and SDA edge. In master mode and in slave mode with NOSTRETCH = 0, the SCL line is stretched low during t_{SDADEL} .

$$t_{SDADEL} = \text{SDADEL} \times t_{PRESC}$$

Note: t_{SDADEL} is used to generate $t_{HD:DAT}$ timing.

Bits 15:8 **SCLH[7:0]**: SCL high period (master mode)

This field is used to generate the SCL high period in master mode.

$$t_{SCLH} = (\text{SCLH}+1) \times t_{PRESC}$$

Note: t_{SCLH} is also used to generate $t_{SU:STO}$ and $t_{HD:STA}$ timing.

Bits 7:0 **SCLL[7:0]**: SCL low period (master mode)

This field is used to generate the SCL low period in master mode.

$$t_{SCLL} = (\text{SCLL}+1) \times t_{PRESC}$$

Note: t_{SCLL} is also used to generate t_{BUF} and $t_{SU:STA}$ timings.

Note: This register must be configured when the I2C is disabled ($PE = 0$).

Note: The STM32CubeMX tool calculates and provides the I2C_TIMINGR content in the I2C Configuration window.

26.7.6 I2C2 timeout register (I2C_TIMEOUTTR)

Address offset: 0x14

Reset value: 0x0000 0000

Access: No wait states, except if a write access occurs while a write access to this register is ongoing. In this case, wait states are inserted in the second write access until the previous one is completed. The latency of the second write access can be up to $2 \times \text{PCLK1} + 6 \times \text{I2CCLK}$.

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

Bit 31 **TEXTEN**: Extended clock timeout enable

0: Extended clock timeout detection is disabled

1: Extended clock timeout detection is enabled. When a cumulative SCL stretch for more than $t_{\text{LOW:EXT}}$ is done by the I2C interface, a timeout error is detected (TIMEOUT=1).

Bits 30:28 Reserved, must be kept at reset value.

Bits 27:16 **TIMEOUTB[11:0]**: Bus timeout B

This field is used to configure the cumulative clock extension timeout:

In master mode, the master cumulative clock low extend time ($t_{\text{LOW:MEXT}}$) is detected

In slave mode, the slave cumulative clock low extend time ($t_{\text{LOW:SEXT}}$) is detected

$t_{\text{LOW:EXT}} = (\text{TIMEOUTB}+1) \times 2048 \times t_{\text{I2CCLK}}$

Note: These bits can be written only when TEXTEN=0.

Bit 15 **TIMOUTEN**: Clock timeout enable

0: SCL timeout detection is disabled

1: SCL timeout detection is enabled: when SCL is low for more than t_{TIMEOUT} (TIDLE=0) or high for more than t_{IDLE} (TIDLE=1), a timeout error is detected (TIMEOUT=1).

Bits 14:13 Reserved, must be kept at reset value.

Bit 12 **TIDLE**: Idle clock timeout detection

0: TIMEOUTA is used to detect SCL low timeout

1: TIMEOUTA is used to detect both SCL and SDA high timeout (bus idle condition)

Note: This bit can be written only when TIMOUTEN=0.

Bits 11:0 **TIMEOUTA[11:0]**: Bus Timeout A

This field is used to configure:

The SCL low timeout condition t_{TIMEOUT} when TIDLE=0

$t_{\text{TIMEOUT}} = (\text{TIMEOUTA}+1) \times 2048 \times t_{\text{I2CCLK}}$

The bus idle condition (both SCL and SDA high) when TIDLE=1

$t_{\text{IDLE}} = (\text{TIMEOUTA}+1) \times 4 \times t_{\text{I2CCLK}}$

Note: These bits can be written only when TIMOUTEN=0.

Note: If the SMBus feature is not supported, this register is reserved and forced by hardware to "0x00000000". Refer to [Section 26.3: I2C implementation](#).

26.7.7 I2C2 interrupt and status register (I2C_ISR)

Address offset: 0x18

Reset value: 0x0000 0001

Access: No wait states

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16							
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ADDCODE[6:0]														DIR
								r	r	r	r	r	r	r	r							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0							
BUSY	Res.	ALERT	TIME OUT	PEC ERR	OVR	ARLO	BERR	TCR	TC	STOPF	NACKF	ADDR	RXNE	TXIS	TXE							
r		r	r	r	r	r	r	r	r	r	r	r	r	rs	rs							

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

Bits 23:17 ADDCODE[6:0]: Address match code (Slave mode)

These bits are updated with the received address when an address match event occurs (ADDR = 1).

In the case of a 10-bit address, ADDCODE provides the 10-bit header followed by the 2 MSBs of the address.

Bit 16 DIR: Transfer direction (Slave mode)

This flag is updated when an address match event occurs (ADDR=1).

0: Write transfer, slave enters receiver mode.

1: Read transfer, slave enters transmitter mode.

Bit 15 BUSY: Bus busy

This flag indicates that a communication is in progress on the bus. It is set by hardware when a START condition is detected. It is cleared by hardware when a STOP condition is detected, or when PE=0.

Bit 14 Reserved, must be kept at reset value.

Bit 13 ALERT: SMBus alert

This flag is set by hardware when SMBHEN=1 (SMBus host configuration), ALERTEN=1 and a SMBALERT event (falling edge) is detected on SMBA pin. It is cleared by software by setting the ALERTCF bit.

Note: This bit is cleared by hardware when PE=0.

If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'.

Refer to Section 26.3: I2C implementation.

Bit 12 TIMEOUT: Timeout or t_{LOW} detection flag

This flag is set by hardware when a timeout or extended clock timeout occurred. It is cleared by software by setting the TIMEOUTCF bit.

Note: This bit is cleared by hardware when PE=0.

If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'.

Refer to Section 26.3: I2C implementation.

Bit 11 **PECERR**: PEC Error in reception

This flag is set by hardware when the received PEC does not match with the PEC register content. A NACK is automatically sent after the wrong PEC reception. It is cleared by software by setting the PECCF bit.

Note: This bit is cleared by hardware when PE=0.

If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'. Refer to Section 26.3: I2C implementation.

Bit 10 **OVR**: Overrun/Underrun (slave mode)

This flag is set by hardware in slave mode with NOSTRETCH=1, when an overrun/underrun error occurs. It is cleared by software by setting the OVRCF bit.

Note: This bit is cleared by hardware when PE=0.

Bit 9 **ARLO**: Arbitration lost

This flag is set by hardware in case of arbitration loss. It is cleared by software by setting the ARLOCF bit.

Note: This bit is cleared by hardware when PE=0.

Bit 8 **BERR**: Bus error

This flag is set by hardware when a misplaced Start or STOP condition is detected whereas the peripheral is involved in the transfer. The flag is not set during the address phase in slave mode. It is cleared by software by setting the BERRCF bit.

Note: This bit is cleared by hardware when PE=0.

Bit 7 **TCR**: Transfer Complete Reload

This flag is set by hardware when RELOAD=1 and NBYTES data have been transferred. It is cleared by software when NBYTES is written to a non-zero value.

Note: This bit is cleared by hardware when PE=0.

This flag is only for master mode, or for slave mode when the SBC bit is set.

Bit 6 **TC**: Transfer Complete (master mode)

This flag is set by hardware when RELOAD=0, AUTOEND=0 and NBYTES data have been transferred. It is cleared by software when START bit or STOP bit is set.

Note: This bit is cleared by hardware when PE=0.

Bit 5 **STOPF**: Stop detection flag

This flag is set by hardware when a STOP condition is detected on the bus and the peripheral is involved in this transfer:

- either as a master, provided that the STOP condition is generated by the peripheral.
- or as a slave, provided that the peripheral has been addressed previously during this transfer.

It is cleared by software by setting the STOPCF bit.

Note: This bit is cleared by hardware when PE=0.

Bit 4 **NACKF**: Not Acknowledge received flag

This flag is set by hardware when a NACK is received after a byte transmission. It is cleared by software by setting the NACKCF bit.

Note: This bit is cleared by hardware when PE=0.

Bit 3 **ADDR**: Address matched (slave mode)

This bit is set by hardware as soon as the received slave address matched with one of the enabled slave addresses. It is cleared by software by setting the ADDRCF bit.

Note: This bit is cleared by hardware when PE=0.

Bit 2 **RXNE**: Receive data register not empty (receivers)

This bit is set by hardware when the received data is copied into the I2C_RXDR register, and is ready to be read. It is cleared when I2C_RXDR is read.

Note: This bit is cleared by hardware when PE=0.

Bit 1 **TXIS**: Transmit interrupt status (transmitters)

This bit is set by hardware when the I2C_TXDR register is empty and the data to be transmitted must be written in the I2C_TXDR register. It is cleared when the next data to be sent is written in the I2C_TXDR register.

This bit can be written to '1' by software when NOSTRETCH=1 only, in order to generate a TXIS event (interrupt if TXIE=1 or DMA request if TXDMAEN=1).

Note: This bit is cleared by hardware when PE=0.

Bit 0 **TXE**: Transmit data register empty (transmitters)

This bit is set by hardware when the I2C_TXDR register is empty. It is cleared when the next data to be sent is written in the I2C_TXDR register.

This bit can be written to '1' by software in order to flush the transmit data register I2C_TXDR.

Note: This bit is set by hardware when PE=0.

26.7.8 I2C2 interrupt clear register (I2C_ICR)(

Address offset: 0x1C

Reset value: 0x0000 0000

Access: No wait states

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	ALERT CF	TIM OUTCF	PECCF	OVRCF	ARLO CF	BERR CF	Res.	Res.	STOP CF	NACK CF	ADDR CF	Res.	Res.	Res.
		w	w	w	w	w	w			w	w	w			

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

Bit 13 **ALERTCF**: Alert flag clear

Writing 1 to this bit clears the ALERT flag in the I2C_ISR register.

Note: If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'. Refer to Section 26.3: I2C implementation.

Bit 12 **TIMOUTCF**: Timeout detection flag clear

Writing 1 to this bit clears the TIMEOUT flag in the I2C_ISR register.

Note: If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'. Refer to Section 26.3: I2C implementation.

Bit 11 **PECCF**: PEC Error flag clear

Writing 1 to this bit clears the PECERR flag in the I2C_ISR register.

Note: If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'. Refer to Section 26.3: I2C implementation.

- Bit 10 **OVRCF**: Overrun/Underrun flag clear
Writing 1 to this bit clears the OVR flag in the I2C_ISR register.
- Bit 9 **ARLOCF**: Arbitration lost flag clear
Writing 1 to this bit clears the ARLO flag in the I2C_ISR register.
- Bit 8 **BERRCF**: Bus error flag clear
Writing 1 to this bit clears the BERRF flag in the I2C_ISR register.
- Bits 7:6 Reserved, must be kept at reset value.
- Bit 5 **STOPCF**: STOP detection flag clear
Writing 1 to this bit clears the STOPF flag in the I2C_ISR register.
- Bit 4 **NACKCF**: Not Acknowledge flag clear
Writing 1 to this bit clears the NACKF flag in I2C_ISR register.
- Bit 3 **ADDRCF**: Address matched flag clear
Writing 1 to this bit clears the ADDR flag in the I2C_ISR register. Writing 1 to this bit also clears the START bit in the I2C_CR2 register.
- Bits 2:0 Reserved, must be kept at reset value.

26.7.9 I2C2 PEC register (I2C_PECR)

Address offset: 0x20

Reset value: 0x0000 0000

Access: No wait states

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

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

Bits 7:0 **PEC[7:0]** Packet error checking register

This field contains the internal PEC when PECEN=1.

The PEC is cleared by hardware when PE=0.

Note: If the SMBus feature is not supported, this register is reserved and forced by hardware to “0x00000000”. Refer to [Section 26.3: I2C implementation](#).

26.7.10 I2C2 receive data register (I2C_RXDR)

Address offset: 0x24

Reset value: 0x0000 0000

Access: No wait states

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

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

Bits 7:0 **RXDATA[7:0]** 8-bit receive data

Data byte received from the I²C bus

26.7.11 I2C2 transmit data register (I2C_TXDR)

Address offset: 0x28

Reset value: 0x0000 0000

Access: No wait states

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

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

Bits 7:0 **TXDATA[7:0]** 8-bit transmit data

Data byte to be transmitted to the I²C bus

Note: These bits can be written only when TXE=1.

26.7.12 I2C register map

The table below provides the I2C register map and reset values.

Table 159. I2C register map and reset values

Offset	Register name	Reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x0	I2C_CR1		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PECEN	ALERTEN	SMBDEN	SMBHEN	GCEN	RES	NOSTRETCH	SBC	RXDMAEN	TXDMAEN	START	RES	ANOFF	ADD10	RD_WRN	DNF[3:0]	SADD[9:0]	OA1[9:0]	OA2[7:1]	OA2MS K [2:0]	PE	0			
	Reset value		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x4	I2C_CR2		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	NBYTES[7:0]																								
	Reset value		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	PECBYTE	0	AUTOEND	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x8	I2C_OAR1		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	NBYTES[7:0]																								
	Reset value		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0xC	I2C_OAR2		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SDADEL[3:0]	SCLDEL[3:0]	SDADEL[3:0]	SCLH[7:0]	SCLL[7:0]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	Reset value		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x10	I2C_TIMINGR		PRESC[3:0]	TEXTEN	0	0	0	0	0	0	TIMEOUTB[11:0]																								
	Reset value		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x14	I2C_TIMEOUTR		TIMEOUTB[11:0]	TIMEOUTA[11:0]	0	0	0	0	0	0	ADDCODE[6:0]	DIR	0	TIMOUTEN	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
	Reset value		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x18	I2C_ISR		ADDCODE[6:0]	TIMEOUTA[11:0]	0	0	0	0	0	0	ALERTCF	0	ALERT	0	TIDLE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	Reset value		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x1C	I2C_ICR		TIMEOUTA[11:0]	TIMEOUTA[11:0]	0	0	0	0	0	0	PECERR	0	PECFC	0	OVRFC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	Reset value		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x20	I2C_PECR		PECERR	PECFC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	Reset value		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x24	I2C_RXDR		PECFC	PECERR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Reset value		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RXDATA[7:0]																																			

Table 159. I2C register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x28	I2C_TXDR	Res.	TXDATA[7:0]																														
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

27 Universal synchronous receiver transmitter (USART) /universal asynchronous receiver transmitter (UART)

27.1 Introduction

The universal synchronous asynchronous receiver transmitter (USART) offers a flexible means of Full-duplex data exchange with external equipment requiring an industry standard NRZ asynchronous serial data format. The USART offers a very wide range of baud rates using a programmable baud rate generator.

It supports synchronous one-way communication and Half-duplex Single-wire communication, as well as multiprocessor communications. It also supports the LIN (Local Interconnect Network), Smartcard protocol and IrDA (Infrared Data Association) SIR ENDEC specifications and Modem operations (CTS/RTS).

High speed data communication is possible by using the DMA (direct memory access) for multibuffer configuration.

27.2 USART main features

- Full-duplex asynchronous communications
- NRZ standard format (mark/space)
- Configurable oversampling method by 16 or 8 to give flexibility between speed and clock tolerance
- A common programmable transmit and receive baud rate of up to 27 Mbit/s when USART clock source is the system clock frequency (Max is 216 MHz) and the oversampling by 8 is used
- Dual clock domain allowing:
 - Convenient baud rate programming independent from the PCLK reprogramming
- Auto baud rate detection
- Programmable data word length (7, 8 or 9 bits)
- Programmable data order with MSB-first or LSB-first shifting
- Configurable stop bits (1 or 2 stop bits)
- Synchronous mode and clock output for synchronous communications
- Single-wire Half-duplex communications
- Continuous communications using DMA
- Received/transmitted bytes are buffered in reserved SRAM using centralized DMA
- Separate enable bits for transmitter and receiver
- Separate signal polarity control for transmission and reception
- Swappable Tx/Rx pin configuration
- Hardware flow control for modem and RS-485 transceiver

- Communication control/error detection flags
- Parity control:
 - Transmits parity bit
 - Checks parity of received data byte
- Fourteen interrupt sources with flags
- Multiprocessor communications
 - The USART enters mute mode if the address does not match.
- Wakeup from mute mode (by idle line detection or address mark detection)

27.3 USART extended features

- LIN master synchronous break send capability and LIN slave break detection capability
 - 13-bit break generation and 10/11-bit break detection when USART is hardware configured for LIN
- IrDA SIR encoder decoder supporting 3/16 bit duration for normal mode
- Smartcard mode
 - Supports the T=0 and T=1 asynchronous protocols for smartcards as defined in the ISO/IEC 7816-3 standard
 - 0.5 and 1.5 stop bits for smartcard operation
- Support for ModBus communication
 - Timeout feature
 - CR/LF character recognition

27.4 USART implementation

Table 160. STM32F72xxx and STM32F73xxx USART features

USART modes/features ⁽¹⁾	USART1/USART2/ USART3/USART6	UART4/UART5/ UART7/UART8
Hardware flow control for modem	X	X
Continuous communication using DMA	X	X
Multiprocessor communication	X	X
Synchronous mode	X	-
Smartcard mode	X	-
Single-wire Half-duplex communication	X	X
IrDA SIR ENDEC block	X	X
LIN mode	X	X
Dual clock domain	X	X
Receiver timeout interrupt	X	X
Modbus communication	X	X
Auto baud rate detection	X	X
Driver Enable	X	X
USART data length	7 ⁽²⁾ , 8 and 9 bits	

1. X = supported.

2. In 7-bit data length mode, Smartcard mode, LIN master mode and Auto baud rate (0x7F and 0x55 frames) detection are not supported.

27.5 USART functional description

Any USART bidirectional communication requires a minimum of two pins: Receive data In (RX) and Transmit data Out (TX):

- **RX:** Receive data Input.

This is the serial data input. Oversampling techniques are used for data recovery by discriminating between valid incoming data and noise.

- **TX:** Transmit data Output.

When the transmitter is disabled, the output pin returns to its I/O port configuration. When the transmitter is enabled and nothing is to be transmitted, the TX pin is at high level. In Single-wire and Smartcard modes, this I/O is used to transmit and receive the data.

Serial data are transmitted and received through these pins in normal USART mode. The frames are comprised of:

- An Idle Line prior to transmission or reception
- A start bit
- A data word (7, 8 or 9 bits) least significant bit first
- 0.5, 1, 1.5, 2 stop bits indicating that the frame is complete
- The USART interface uses a baud rate generator
- A status register (USART_ISR)
- Receive and transmit data registers (USART_RDR, USART_TDR)
- A baud rate register (USART_BRR)
- A guard-time register (USART_GTPR) in case of Smartcard mode.

Refer to [Section 27.8: USART registers on page 929](#) for the definitions of each bit.

The following pin is required to interface in synchronous mode and Smartcard mode:

- **CK:** Clock output. This pin outputs the transmitter data clock for synchronous transmission corresponding to SPI master mode (no clock pulses on start bit and stop bit, and a software option to send a clock pulse on the last data bit). In parallel, data can be received synchronously on RX. This can be used to control peripherals that have shift registers. The clock phase and polarity are software programmable. In Smartcard mode, CK output can provide the clock to the smartcard.

The following pins are required in RS232 Hardware flow control mode:

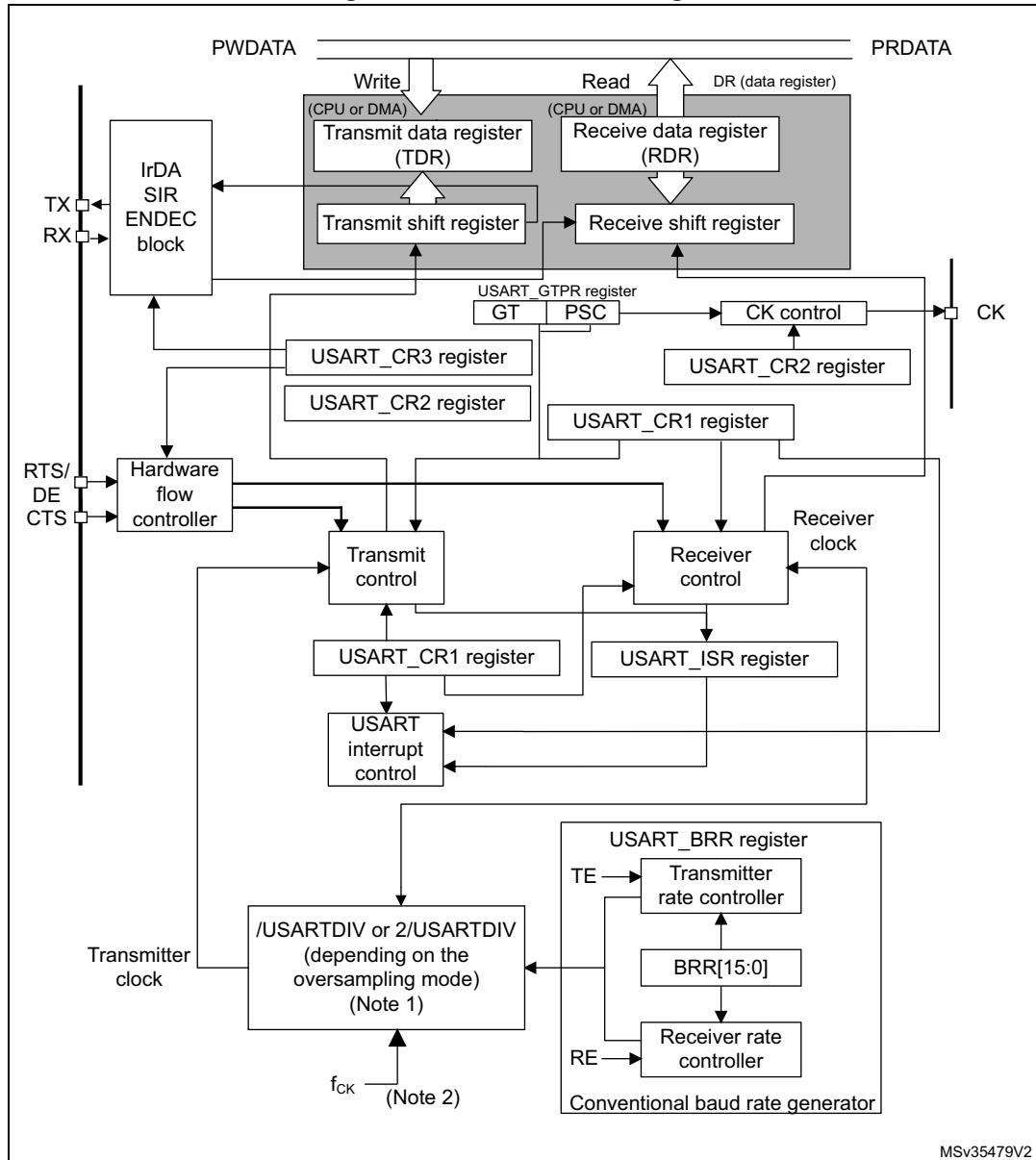
- **CTS:** Clear To Send blocks the data transmission at the end of the current transfer when high
- **RTS:** Request to send indicates that the USART is ready to receive data (when low).

The following pin is required in RS485 Hardware control mode:

- **DE:** Driver Enable activates the transmission mode of the external transceiver.

Note: *DE and RTS share the same pin.*

Figure 317. USART block diagram



MSv35479V2

1. For details on coding USARTDIV in the USART_BRR register, refer to [Section 27.5.4: USART baud rate generation](#).
2. f_{CK} can be f_{LSE} , f_{HSI} , f_{PCLK} , f_{SYS} .

27.5.1 USART character description

The word length can be selected as being either 7 or 8 or 9 bits by programming the M[1:0] bits in the USART_CR1 register (see [Figure 318](#)).

- 7-bit character length: M[1:0] = 10
- 8-bit character length: M[1:0] = 00
- 9-bit character length: M[1:0] = 01

Note: *The 7-bit mode is supported only on some USARTs. In addition, not all modes are supported in 7-bit data length mode. Refer to [Section 27.4: USART implementation](#) for additional information.*

By default, the signal (TX or RX) is in low state during the start bit. It is in high state during the stop bit.

These values can be inverted, separately for each signal, through polarity configuration control.

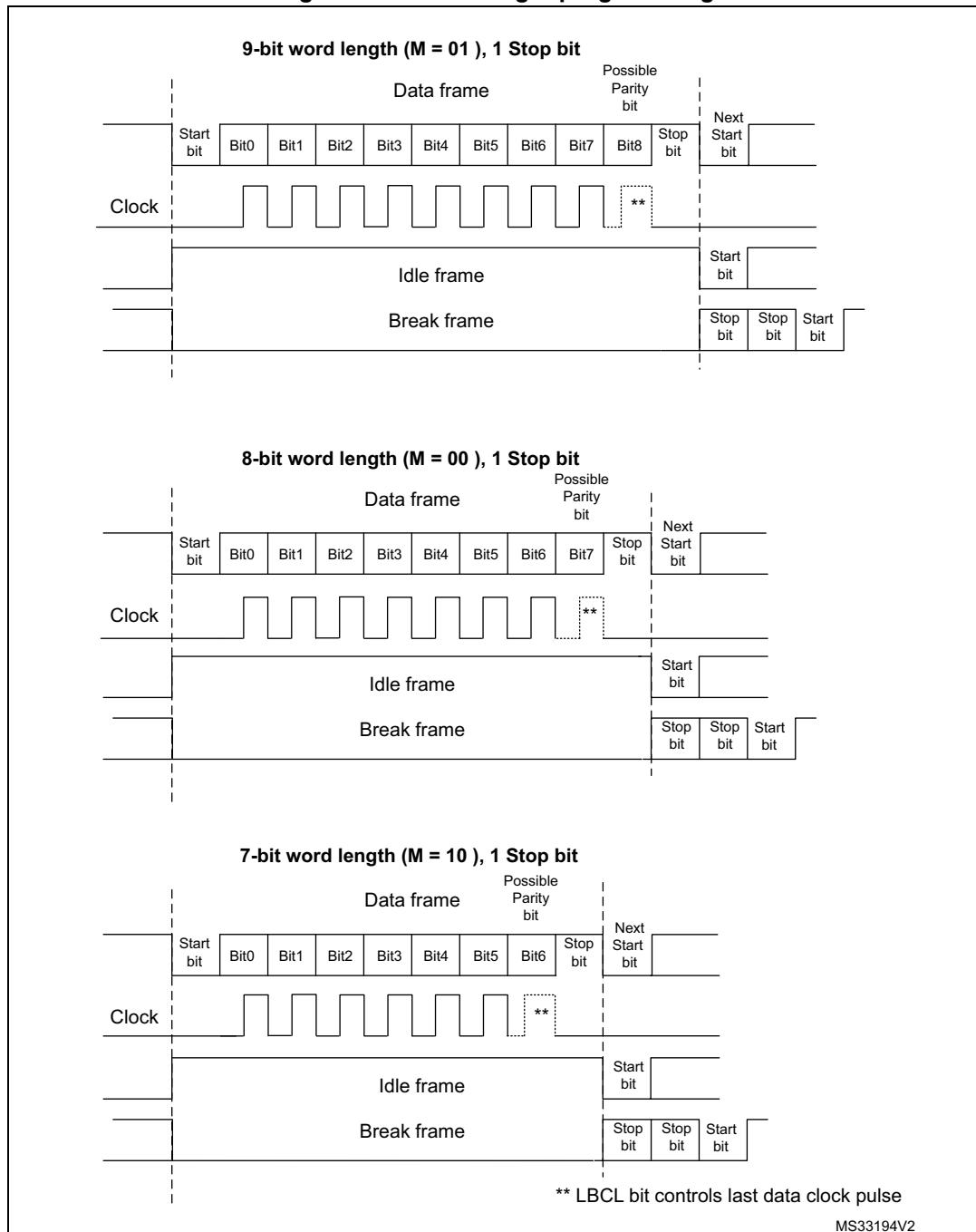
An **Idle character** is interpreted as an entire frame of “1”s (the number of “1”s includes the number of stop bits).

A **Break character** is interpreted on receiving “0”s for a frame period. At the end of the break frame, the transmitter inserts 2 stop bits.

Transmission and reception are driven by a common baud rate generator, the clock for each is generated when the enable bit is set respectively for the transmitter and receiver.

The details of each block is given below.

Figure 318. Word length programming



27.5.2 USART transmitter

The transmitter can send data words of either 7, 8 or 9 bits depending on the M bits status. The Transmit Enable bit (TE) must be set in order to activate the transmitter function. The data in the transmit shift register is output on the TX pin and the corresponding clock pulses are output on the CK pin.

Character transmission

During an USART transmission, data shifts out least significant bit first (default configuration) on the TX pin. In this mode, the USART_TDR register consists of a buffer (TDR) between the internal bus and the transmit shift register (see [Figure 317](#)).

Every character is preceded by a start bit which is a logic level low for one bit period. The character is terminated by a configurable number of stop bits.

The following stop bits are supported by USART: 0.5, 1, 1.5 and 2 stop bits.

Note: The TE bit must be set before writing the data to be transmitted to the USART_TDR.

The TE bit should not be reset during transmission of data. Resetting the TE bit during the transmission will corrupt the data on the TX pin as the baud rate counters will get frozen. The current data being transmitted will be lost.

An idle frame will be sent after the TE bit is enabled.

Configurable stop bits

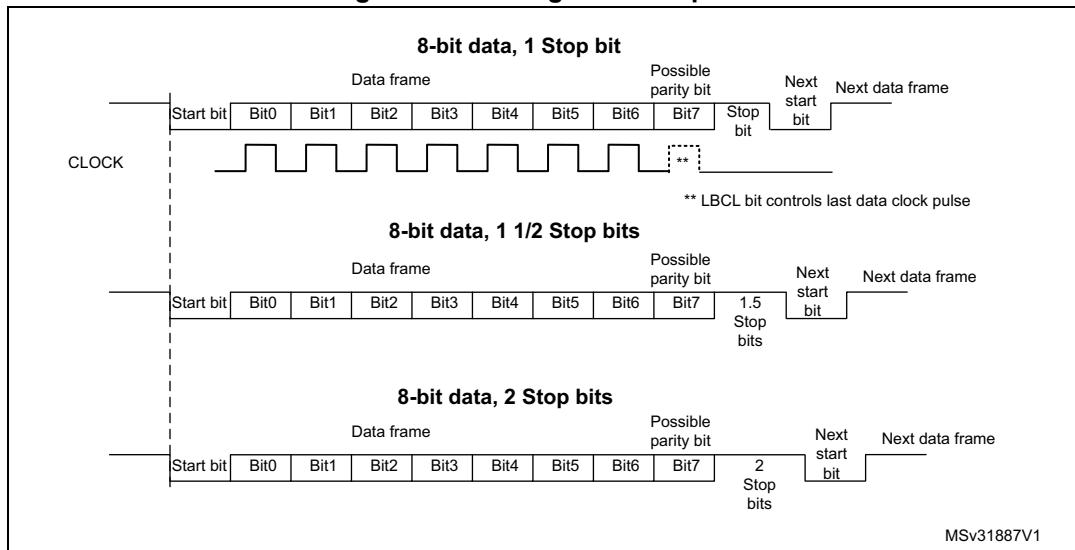
The number of stop bits to be transmitted with every character can be programmed in Control register 2, bits 13,12.

- **1 stop bit:** This is the default value of number of stop bits.
- **2 stop bits:** This will be supported by normal USART, Single-wire and Modem modes.
- **1.5 stop bits:** To be used in Smartcard mode.
- **0.5 stop bit:** To be used when receiving data in Smartcard mode.

An idle frame transmission will include the stop bits.

A break transmission will be 10 low bits (when M[1:0] = 00) or 11 low bits (when M[1:0] = 01) or 9 low bits (when M[1:0] = 10) followed by 2 stop bits (see [Figure 319](#)). It is not possible to transmit long breaks (break of length greater than 9/10/11 low bits).

Figure 319. Configurable stop bits



Character transmission procedure

1. Program the M bits in USART_CR1 to define the word length.
2. Select the desired baud rate using the USART_BRR register.
3. Program the number of stop bits in USART_CR2.
4. Enable the USART by writing the UE bit in USART_CR1 register to 1.
5. Select DMA enable (DMAT) in USART_CR3 if multibuffer communication is to take place. Configure the DMA register as explained in multibuffer communication.
6. Set the TE bit in USART_CR1 to send an idle frame as first transmission.
7. Write the data to send in the USART_TDR register (this clears the TXE bit). Repeat this for each data to be transmitted in case of single buffer.
8. After writing the last data into the USART_TDR register, wait until TC=1. This indicates that the transmission of the last frame is complete. This is required for instance when the USART is disabled or enters the Halt mode to avoid corrupting the last transmission.

Single byte communication

Clearing the TXE bit is always performed by a write to the transmit data register.

The TXE bit is set by hardware and it indicates:

- The data has been moved from the USART_TDR register to the shift register and the data transmission has started.
- The USART_TDR register is empty.
- The next data can be written in the USART_TDR register without overwriting the previous data.

This flag generates an interrupt if the TXEIE bit is set.

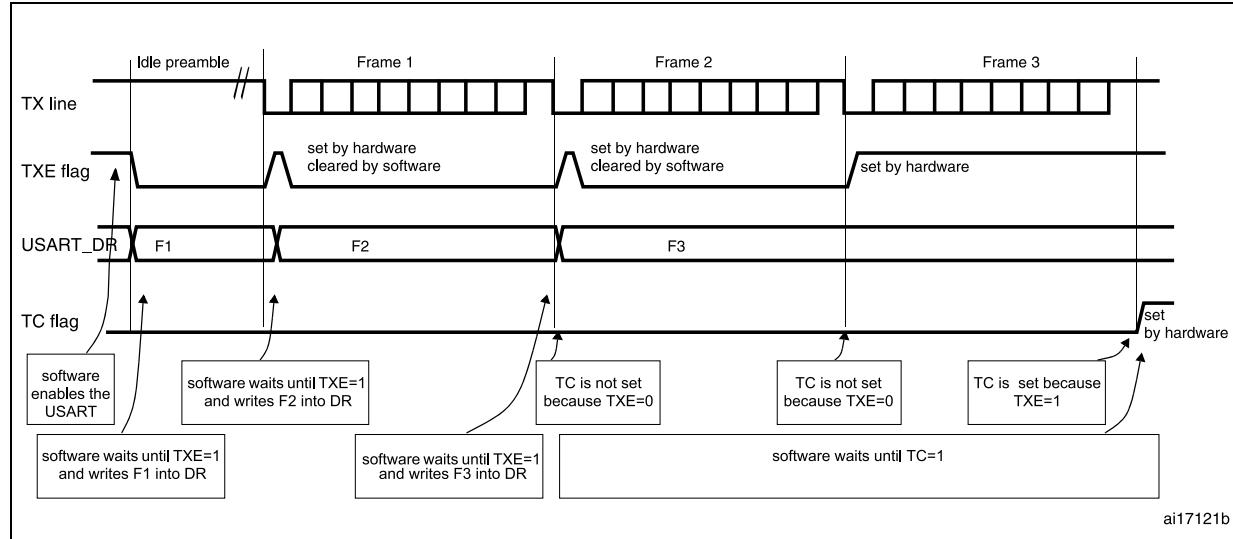
When a transmission is taking place, a write instruction to the USART_TDR register stores the data in the TDR register; next, the data is copied in the shift register at the end of the currently ongoing transmission.

When no transmission is taking place, a write instruction to the USART_TDR register places the data in the shift register, the data transmission starts, and the TXE bit is set.

If a frame is transmitted (after the stop bit) and the TXE bit is set, the TC bit goes high. An interrupt is generated if the TCIE bit is set in the USART_CR1 register.

After writing the last data in the USART_TDR register, it is mandatory to wait for TC=1 before disabling the USART or causing the microcontroller to enter the low-power mode (see [Figure 320: TC/TXE behavior when transmitting](#)).

Figure 320. TC/TXE behavior when transmitting



Break characters

Setting the SBKRQ bit transmits a break character. The break frame length depends on the M bits (see [Figure 318](#)).

If a '1' is written to the SBKRQ bit, a break character is sent on the TX line after completing the current character transmission. The SBKF bit is set by the write operation and it is reset by hardware when the break character is completed (during the stop bits after the break character). The USART inserts a logic 1 signal (STOP) for the duration of 2 bits at the end of the break frame to guarantee the recognition of the start bit of the next frame.

In the case the application needs to send the break character following all previously inserted data, including the ones not yet transmitted, the software should wait for the TXE flag assertion before setting the SBKRQ bit.

Idle characters

Setting the TE bit drives the USART to send an idle frame before the first data frame.

27.5.3 USART receiver

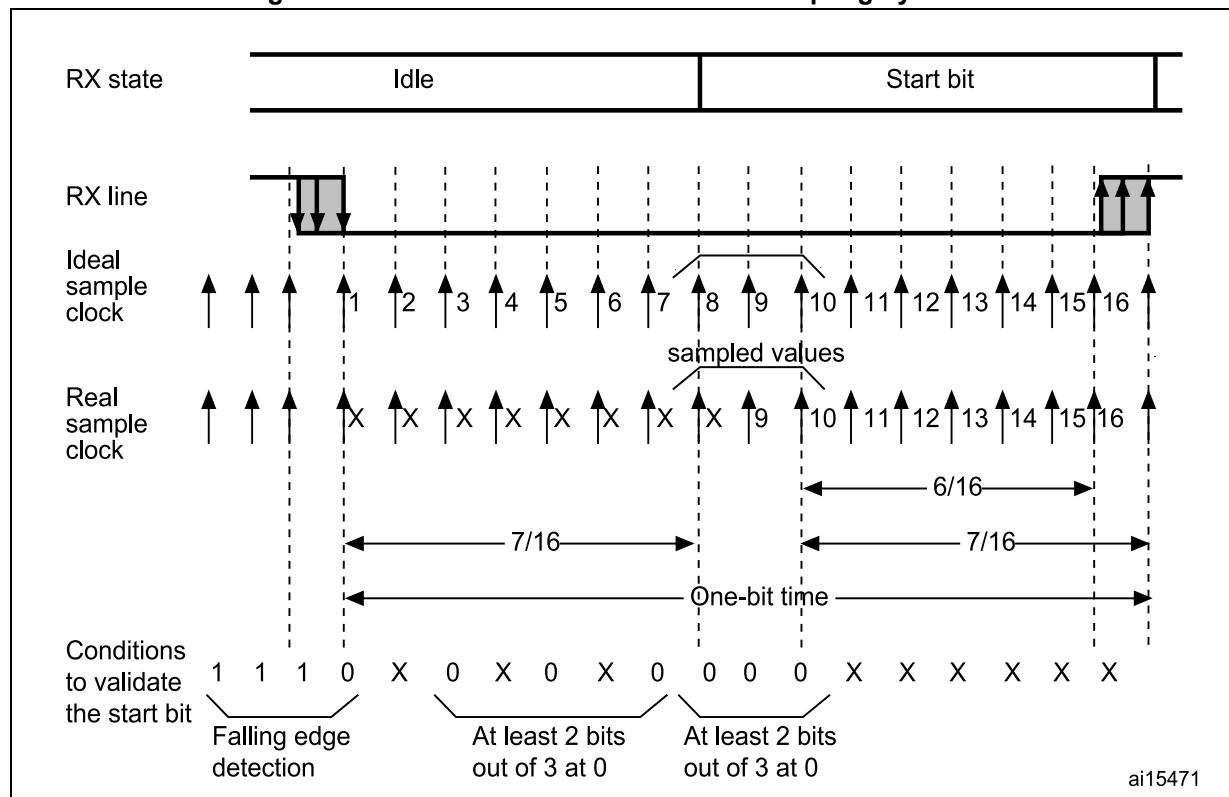
The USART can receive data words of either 7, 8 or 9 bits depending on the M bits in the USART_CR1 register.

Start bit detection

The start bit detection sequence is the same when oversampling by 16 or by 8.

In the USART, the start bit is detected when a specific sequence of samples is recognized. This sequence is: 1 1 1 0 X 0 X 0X 0X 0 X 0X 0.

Figure 321. Start bit detection when oversampling by 16 or 8



Note: If the sequence is not complete, the start bit detection aborts and the receiver returns to the idle state (no flag is set), where it waits for a falling edge.

The start bit is confirmed (RXNE flag set, interrupt generated if RXNEIE=1) if the 3 sampled bits are at 0 (first sampling on the 3rd, 5th and 7th bits finds the 3 bits at 0 and second sampling on the 8th, 9th and 10th bits also finds the 3 bits at 0).

The start bit is validated (RXNE flag set, interrupt generated if RXNEIE=1) but the NF noise flag is set if,

- for both samplings, 2 out of the 3 sampled bits are at 0 (sampling on the 3rd, 5th and 7th bits and sampling on the 8th, 9th and 10th bits)
- or
- for one of the samplings (sampling on the 3rd, 5th and 7th bits or sampling on the 8th, 9th and 10th bits), 2 out of the 3 bits are found at 0.

If neither conditions a. or b. are met, the start detection aborts and the receiver returns to the idle state (no flag is set).

Character reception

During an USART reception, data shifts in least significant bit first (default configuration) through the RX pin. In this mode, the USART_RDR register consists of a buffer (RDR) between the internal bus and the receive shift register.

Character reception procedure

1. Program the M bits in USART_CR1 to define the word length.
2. Select the desired baud rate using the baud rate register USART_BRR
3. Program the number of stop bits in USART_CR2.
4. Enable the USART by writing the UE bit in USART_CR1 register to 1.
5. Select DMA enable (DMAR) in USART_CR3 if multibuffer communication is to take place. Configure the DMA register as explained in multibuffer communication.
6. Set the RE bit USART_CR1. This enables the receiver which begins searching for a start bit.

When a character is received:

- The RXNE bit is set to indicate that the content of the shift register is transferred to the RDR. In other words, data has been received and can be read (as well as its associated error flags).
- An interrupt is generated if the RXNEIE bit is set.
- The error flags can be set if a frame error, noise or an overrun error has been detected during reception. PE flag can also be set with RXNE.
- In multibuffer, RXNE is set after every byte received and is cleared by the DMA read of the Receive data Register.
- In single buffer mode, clearing the RXNE bit is performed by a software read to the USART_RDR register. The RXNE flag can also be cleared by writing 1 to the RXFRQ in the USART_RQR register. The RXNE bit must be cleared before the end of the reception of the next character to avoid an overrun error.

Break character

When a break character is received, the USART handles it as a framing error.

Idle character

When an idle frame is detected, there is the same procedure as for a received data character plus an interrupt if the IDLEIE bit is set.

Overrun error

An overrun error occurs when a character is received when RXNE has not been reset. Data can not be transferred from the shift register to the RDR register until the RXNE bit is cleared.

The RXNE flag is set after every byte received. An overrun error occurs if RXNE flag is set when the next data is received or the previous DMA request has not been serviced. When an overrun error occurs:

- The ORE bit is set.
- The RDR content will not be lost. The previous data is available when a read to USART_RDR is performed.
- The shift register 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 EIE bit is set.
- The ORE bit is reset by setting the ORECF bit in the ICR register.

Note:

The ORE bit, when set, indicates that at least 1 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.

Selecting the clock source and the proper oversampling method

The choice of the clock source is done through the Clock Control system (see Section Reset and clock control (RCC))). The clock source must be chosen before enabling the USART (by setting the UE bit).

The clock source frequency is f_{CK} .

When the dual clock domain is supported, the clock source can be one of the following sources: PCLK (default), LSE, HSI or SYSCLK.

The communication speed range (specially the maximum communication speed) is also determined by the clock source.

The receiver implements different user-configurable oversampling techniques for data recovery by discriminating between valid incoming data and noise. This allows a trade-off between the maximum communication speed and noise/clock inaccuracy immunity.

The oversampling method can be selected by programming the OVER8 bit in the USART_CR1 register and can be either 16 or 8 times the baud rate clock ([Figure 322](#) and [Figure 323](#)).

Depending on the application:

- Select oversampling by 8 (OVER8=1) to achieve higher speed (up to $f_{CK}/8$). In this case the maximum receiver tolerance to clock deviation is reduced (refer to [Section 27.5.5: Tolerance of the USART receiver to clock deviation on page 904](#))
- Select oversampling by 16 (OVER8=0) to increase the tolerance of the receiver to clock deviations. In this case, the maximum speed is limited to maximum $f_{CK}/16$ where f_{CK} is the clock source frequency.

Programming the ONEBIT bit in the USART_CR3 register selects the method used to evaluate the logic level. There are two options:

- The majority vote of the three samples in the center of the received bit. In this case, when the 3 samples used for the majority vote are not equal, the NF bit is set
- A single sample in the center of the received bit

Depending on the application:

- select the three samples' majority vote method (ONEBIT=0) when operating in a noisy environment and reject the data when a noise is detected (refer to [Figure 161](#)) because this indicates that a glitch occurred during the sampling.
- select the single sample method (ONEBIT=1) when the line is noise-free to increase the receiver's tolerance to clock deviations (see [Section 27.5.5: Tolerance of the USART receiver to clock deviation on page 904](#)). In this case the NF bit will never be set.

When noise is detected in a frame:

- The NF bit is set at the rising edge of the RXNE bit.
- The invalid data is transferred from the Shift register to the USART_RDR register.
- No interrupt is generated in case of single byte communication. However this bit rises at the same time as the RXNE bit which itself generates an interrupt. In case of multibuffer communication an interrupt will be issued if the EIE bit is set in the USART_CR3 register.

The NF bit is reset by setting NFCF bit in ICR register.

Note: *Oversampling by 8 is not available in LIN, Smartcard and IrDA modes. In those modes, the OVER8 bit is forced to '0' by hardware.*

Figure 322. Data sampling when oversampling by 16

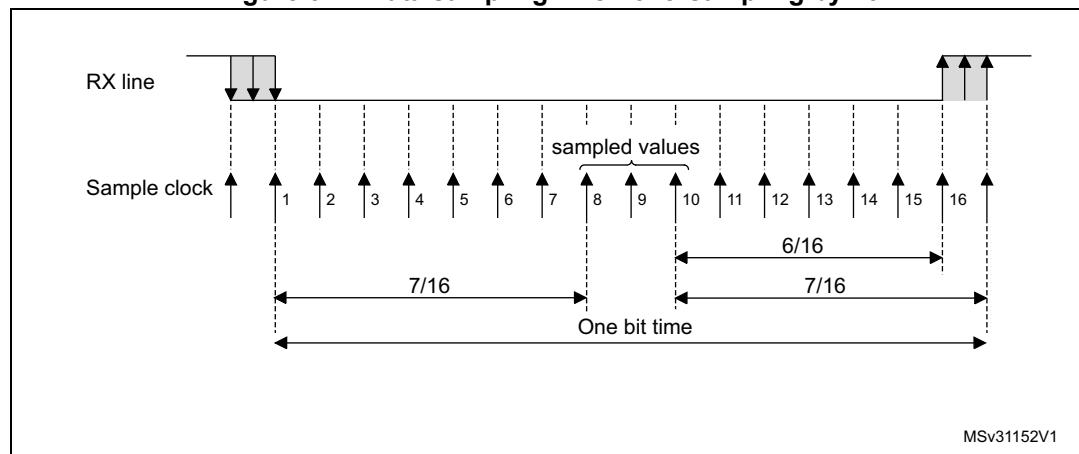


Figure 323. Data sampling when oversampling by 8

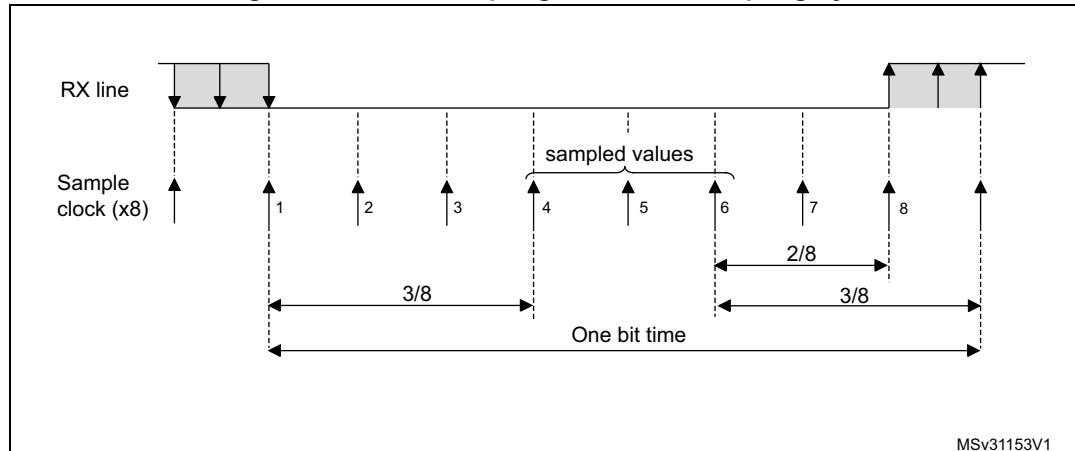


Table 161. Noise detection from sampled data

Sampled value	NE status	Received bit value
000	0	0
001	1	0
010	1	0
011	1	1
100	1	0
101	1	1
110	1	1
111	0	1

Framing error

A framing error is detected when the stop bit is not recognized on reception at the expected time, following either a de-synchronization or excessive noise.

When the framing error is detected:

- The FE bit is set by hardware
- The invalid data is transferred from the Shift register to the USART_RDR register.
- No interrupt is generated in case of single byte communication. However this bit rises at the same time as the RXNE bit which itself generates an interrupt. In case of multibuffer communication an interrupt will be issued if the EIE bit is set in the USART_CR3 register.

The FE bit is reset by writing 1 to the FECF in the USART_ICR register.

Configurable stop bits during reception

The number of stop bits to be received can be configured through the control bits of Control Register 2 - it can be either 1 or 2 in normal mode and 0.5 or 1.5 in Smartcard mode.

- **0.5 stop bit (reception in Smartcard mode):** No sampling is done for 0.5 stop bit. As a consequence, no framing error and no break frame can be detected when 0.5 stop bit is selected.
- **1 stop bit:** Sampling for 1 stop Bit is done on the 8th, 9th and 10th samples.
- **1.5 stop bits (Smartcard mode):** When transmitting in Smartcard mode, the device must check that the data is correctly sent. Thus the receiver block must be enabled (RE =1 in the USART_CR1 register) and the stop bit is checked to test if the smartcard has detected a parity error. In the event of a parity error, the smartcard forces the data signal low during the sampling - NACK signal-, which is flagged as a framing error. Then, the FE flag is set with the RXNE at the end of the 1.5 stop bits. Sampling for 1.5 stop bits is done on the 16th, 17th and 18th samples (1 baud clock period after the beginning of the stop bit). The 1.5 stop bits can be decomposed into 2 parts: one 0.5 baud clock period during which nothing happens, followed by 1 normal stop bit period during which sampling occurs halfway through. Refer to [Section 27.5.13: USART Smartcard mode on page 915](#) for more details.
- **2 stop bits:** Sampling for 2 stop bits is done on the 8th, 9th and 10th samples of the first stop bit. If a framing error is detected during the first stop bit the framing error flag 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.

27.5.4 USART baud rate generation

The baud rate for the receiver and transmitter (Rx and Tx) are both set to the same value as programmed in the USART_BRR register.

Equation 1: Baud rate for standard USART (SPI mode included) (OVER8 = 0 or 1)

In case of oversampling by 16, the equation is:

$$\text{Tx/Rx baud} = \frac{f_{CK}}{\text{USARTDIV}}$$

In case of oversampling by 8, the equation is:

$$\text{Tx/Rx baud} = \frac{2 \times f_{CK}}{\text{USARTDIV}}$$

Equation 2: Baud rate in Smartcard, LIN and IrDA modes (OVER8 = 0)

In Smartcard, LIN and IrDA modes, only Oversampling by 16 is supported:

$$\text{Tx/Rx baud} = \frac{f_{CK}}{\text{USARTDIV}}$$

USARTDIV is an unsigned fixed point number that is coded on the USART_BRR register.

- When OVER8 = 0, BRR = USARTDIV.
- When OVER8 = 1
 - BRR[2:0] = USARTDIV[3:0] shifted 1 bit to the right.
 - BRR[3] must be kept cleared.
 - BRR[15:4] = USARTDIV[15:4]

Note: *The baud counters are updated to the new value in the baud registers after a write operation to USART_BRR. Hence the baud rate register value should not be changed during communication.*

In case of oversampling by 16 or 8, USARTDIV must be greater than or equal to 16d.

How to derive USARTDIV from USART_BRR register values

Example 1

To obtain 9600 baud with $f_{CK} = 8$ MHz.

- In case of oversampling by 16:
USARTDIV = $8\ 000\ 000/9600$
BRR = USARTDIV = $833d = 0341h$
- In case of oversampling by 8:
USARTDIV = $2 * 8\ 000\ 000/9600$
USARTDIV = $1666,66$ ($1667d = 683h$)
BRR[3:0] = $3h >> 1 = 1h$
BRR = $0x681$

Example 2

To obtain 921.6 Kbaud with $f_{CK} = 48$ MHz.

- In case of oversampling by 16:
USARTDIV = $48\ 000\ 000/921\ 600$
BRR = USARTDIV = $52d = 34h$
- In case of oversampling by 8:
USARTDIV = $2 * 48\ 000\ 000/921\ 600$
USARTDIV = 104 ($104d = 68h$)
BRR[3:0] = USARTDIV[3:0] $>> 1 = 8h >> 1 = 4h$
BRR = $0x64$

**Table 162. Error calculation for programmed baud rates at $f_{CK} = 216$ MHz
in both cases of oversampling by 8 (OVER8 = 1)⁽¹⁾**

Desired baud rate (Bps)	Actual baud rate (Bps)	BRR	Error
9600	9600	AFC4	0.00000000
19200	19200	57E2	0.00000000
38400	38400	2BF1	0.00000000
57600	57600	1D46	0.00000000
115200	115200	EA3	0.00000000
230400	230400	751	0.00000000
460800	461538.461	3A4	0.160256293
921600	923076.923	1D2	0.001602564
13500000	13500000.000	20	0.00000000
27000000	27000000.000	10	0.00000000

1. The lower the CPU clock the lower the accuracy for a particular baud rate. The upper limit of the achievable baud rate can be fixed with these data.

**Table 163. Error calculation for programmed baud rates at $f_{CK} = 216$ MHz
in both cases of oversampling by 16 (OVER8 = 0)⁽¹⁾**

Desired baud rate (Bps)	Actual baud rate (Bps)	BRR	Error
9600	9600.000	57E4	0.00000000
19200	19200.000	2BF2	0.00000000
38400	38400.000	15F9	0.00000000
57600	57600.000	EA6	0.00000000
115200	115200.000	753	0.00000000
230400	230522.946	3A9	0.05336179
460800	461538.462	1D4	0.16025641
921600	923076.923	EA	0.16025641
4000000	4000000.000	36	0.00000000
6000000	6000000.000	24	0.00000000
10000000	10285714.286	15	2.85714286
13500000	13500000.000	10	0.00000000

1. The lower the CPU clock the lower the accuracy for a particular baud rate. The upper limit of the achievable baud rate can be fixed with these data.

27.5.5 Tolerance of the USART receiver to clock deviation

The asynchronous receiver of the USART works correctly only if the total clock system deviation is less than the tolerance of the USART receiver. The causes which contribute to the total deviation are:

- DTRA: Deviation due to the transmitter error (which also includes the deviation of the transmitter's local oscillator)
- DQUANT: Error due to the baud rate quantization of the receiver
- DREC: Deviation of the receiver's local oscillator
- DTCL: Deviation due to the transmission line (generally due to the transceivers which can introduce an asymmetry between the low-to-high transition timing and the high-to-low transition timing)

$$\text{DTRA} + \text{DQUANT} + \text{DREC} + \text{DTCL} < \text{USART receiver's tolerance}$$

The USART receiver can receive data correctly at up to the maximum tolerated deviation specified in [Table 164](#) and [Table 164](#) depending on the following choices:

- 9-, 10- or 11-bit character length defined by the M bits in the USART_CR1 register
- Oversampling by 8 or 16 defined by the OVER8 bit in the USART_CR1 register
- Bits BRR[3:0] of USART_BRR register are equal to or different from 0000.
- Use of 1 bit or 3 bits to sample the data, depending on the value of the ONEBIT bit in the USART_CR3 register.

Table 164. Tolerance of the USART receiver when BRR [3:0] = 0000

M bits	OVER8 bit = 0		OVER8 bit = 1	
	ONEBIT=0	ONEBIT=1	ONEBIT=0	ONEBIT=1
00	3.75%	4.375%	2.50%	3.75%
01	3.41%	3.97%	2.27%	3.41%
10	4.16%	4.86%	2.77%	4.16%

Table 165. Tolerance of the USART receiver when BRR [3:0] is different from 0000

M bits	OVER8 bit = 0		OVER8 bit = 1	
	ONEBIT=0	ONEBIT=1	ONEBIT=0	ONEBIT=1
00	3.33%	3.88%	2%	3%
01	3.03%	3.53%	1.82%	2.73%
10	3.7%	4.31%	2.22%	3.33%

Note:

The data specified in [Table 164](#) and [Table 165](#) may slightly differ in the special case when the received frames contain some Idle frames of exactly 10-bit durations when M bits = 00 (11-bit durations when M bits =01 or 9- bit durations when M bits = 10).

27.5.6 USART auto baud rate detection

The USART is able to detect and automatically set the USART_BRR register value based on the reception of one character. Automatic baud rate detection is useful under two circumstances:

- The communication speed of the system is not known in advance
- The system is using a relatively low accuracy clock source and this mechanism allows the correct baud rate to be obtained without measuring the clock deviation.

The clock source frequency must be compatible with the expected communication speed (when oversampling by 16, the baud rate is between $f_{CK}/65535$ and $f_{CK}/16$. when oversampling by 8, the baud rate is between $f_{CK}/65535$ and $f_{CK}/8$).

Before activating the auto baud rate detection, the auto baud rate detection mode must be chosen. There are various modes based on different character patterns.

They can be chosen through the ABRMOD[1:0] field in the USART_CR2 register. In these auto baud rate modes, the baud rate is measured several times during the synchronization data reception and each measurement is compared to the previous one.

These modes are:

- **Mode 0:** Any character starting with a bit at 1. In this case the USART measures the duration of the Start bit (falling edge to rising edge).
- **Mode 1:** Any character starting with a 10xx bit pattern. In this case, the USART measures the duration of the Start and of the 1st data bit. The measurement is done falling edge to falling edge, ensuring better accuracy in the case of slow signal slopes.
- **Mode 2:** A 0x7F character frame (it may be a 0x7F character in LSB first mode or a 0xFE in MSB first mode). In this case, the baud rate is updated first at the end of the start bit (BRs), then at the end of bit 6 (based on the measurement done from falling edge to falling edge: BR6). Bit 0 to bit 6 are sampled at BRs while further bits of the character are sampled at BR6.
- **Mode 3:** A 0x55 character frame. In this case, the baud rate is updated first at the end of the start bit (BRs), then at the end of bit 0 (based on the measurement done from falling edge to falling edge: BR0), and finally at the end of bit 6 (BR6). Bit 0 is sampled at BRs, bit 1 to bit 6 are sampled at BR0, and further bits of the character are sampled at BR6.

In parallel, another check is performed for each intermediate transition of RX line. An error is generated if the transitions on RX are not sufficiently synchronized with the receiver (the receiver being based on the baud rate calculated on bit 0).

Prior to activating auto baud rate detection, the USART_BRR register must be initialized by writing a non-zero baud rate value.

The automatic baud rate detection is activated by setting the ABREN bit in the USART_CR2 register. The USART will then wait for the first character on the RX line. The auto baud rate operation completion is indicated by the setting of the ABRF flag in the USART_ISR register. If the line is noisy, the correct baud rate detection cannot be guaranteed. In this case the BRR value may be corrupted and the ABRE error flag will be set. This also happens if the communication speed is not compatible with the automatic baud rate detection range (bit duration not between 16 and 65536 clock periods (oversampling by 16) and not between 8 and 65536 clock periods (oversampling by 8)).

The RXNE interrupt will signal the end of the operation.

At any later time, the auto baud rate detection may be relaunched by resetting the ABRF flag (by writing a 0).

Note: *If the USART is disabled (UE=0) during an auto baud rate operation, the BRR value may be corrupted.*

27.5.7 Multiprocessor communication using USART

In multiprocessor communication, the following bits are to be kept cleared:

- LINEN bit in the USART_CR2 register,
- HDSEL, IREN and SCEN bits in the USART_CR3 register.

It is possible to perform multiprocessor communication with the USART (with several USARTs connected in a network). For instance one of the USARTs can be the master, its TX output connected to the RX inputs of the other USARTs. The others are slaves, their respective TX outputs are logically ANDed together and connected to the RX input of the master.

In multiprocessor configurations it is often desirable that only the intended message recipient should actively receive the full message contents, thus reducing redundant USART service overhead for all non addressed receivers.

The non addressed devices may be placed in mute mode by means of the muting function. In order to use the mute mode feature, the MME bit must be set in the USART_CR1 register.

In mute mode:

- None of the reception status bits can be set.
- All the receive interrupts are inhibited.
- The RWU bit in USART_ISR register is set to 1. RWU can be controlled automatically by hardware or by software, through the MMRQ bit in the USART_RQR register, under certain conditions.

The USART can enter or exit from mute mode using one of two methods, depending on the WAKE bit in the USART_CR1 register:

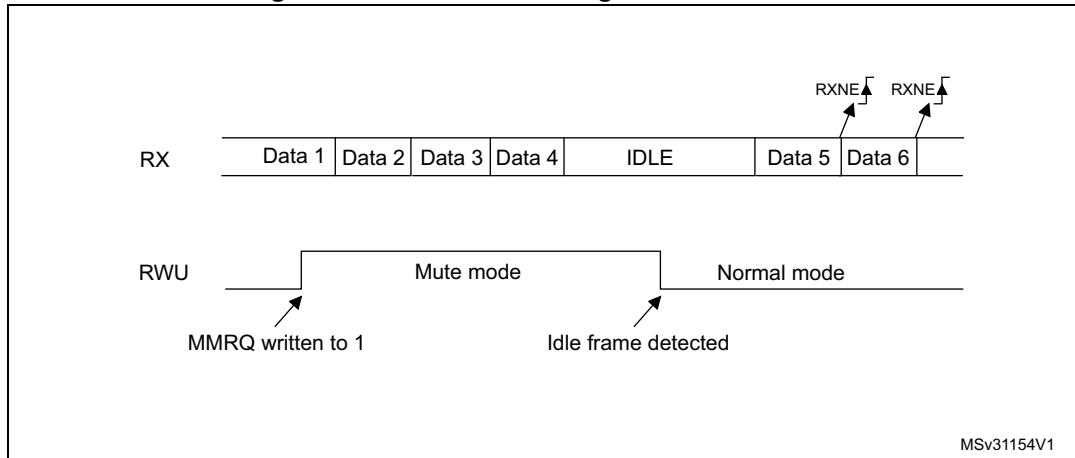
- Idle Line detection if the WAKE bit is reset,
- Address Mark detection if the WAKE bit is set.

Idle line detection (WAKE=0)

The USART enters mute mode when the MMRQ bit is written to 1 and the RWU is automatically set.

It wakes up when an Idle frame is detected. Then the RWU bit is cleared by hardware but the IDLE bit is not set in the USART_ISR register. An example of mute mode behavior using Idle line detection is given in [Figure 324](#).

Figure 324. Mute mode using Idle line detection



Note: If the MMRQ is set while the IDLE character has already elapsed, mute mode will not be entered (RWU is not set).

If the USART is activated while the line is IDLE, the idle state is detected after the duration of one IDLE frame (not only after the reception of one character frame).

4-bit/7-bit address mark detection (WAKE=1)

In this mode, bytes are recognized as addresses if their MSB is a '1' otherwise they are considered as data. In an address byte, the address of the targeted receiver is put in the 4 or 7 LSBs. The choice of 7 or 4-bit address detection is done using the ADDM7 bit. This 4-bit/7-bit word is compared by the receiver with its own address which is programmed in the ADD bits in the USART_CR2 register.

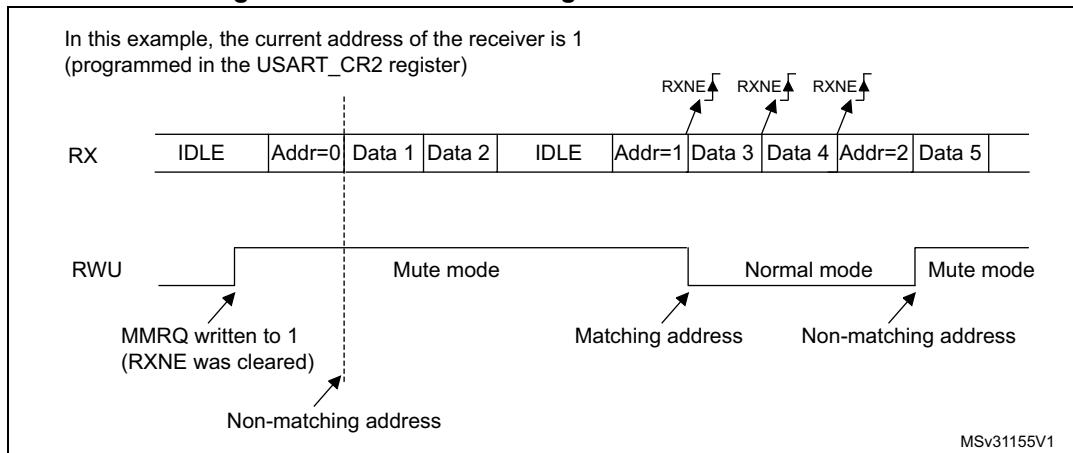
Note: In 7-bit and 9-bit data modes, address detection is done on 6-bit and 8-bit addresses (ADD[5:0] and ADD[7:0]) respectively.

The USART enters mute mode when an address character is received which does not match its programmed address. In this case, the RWU bit is set by hardware. The RXNE flag is not set for this address byte and no interrupt or DMA request is issued when the USART enters mute mode.

The USART also enters mute mode when the MMRQ bit is written to 1. The RWU bit is also automatically set in this case.

The USART exits from mute mode when an address character is received which matches the programmed address. Then the RWU bit is cleared and subsequent bytes are received normally. The RXNE bit is set for the address character since the RWU bit has been cleared.

An example of mute mode behavior using address mark detection is given in [Figure 325](#).

Figure 325. Mute mode using address mark detection

27.5.8 Modbus communication using USART

The USART offers basic support for the implementation of Modbus/RTU and Modbus/ASCII protocols. Modbus/RTU is a half duplex, block transfer protocol. The control part of the protocol (address recognition, block integrity control and command interpretation) must be implemented in software.

The USART offers basic support for the end of the block detection, without software overhead or other resources.

Modbus/RTU

In this mode, the end of one block is recognized by a “silence” (idle line) for more than 2 character times. This function is implemented through the programmable timeout function.

The timeout function and interrupt must be activated, through the RTOEN bit in the USART_CR2 register and the RTOIE in the USART_CR1 register. The value corresponding to a timeout of 2 character times (for example 22 x bit duration) must be programmed in the RTO register. When the receive line is idle for this duration, after the last stop bit is received, an interrupt is generated, informing the software that the current block reception is completed.

Modbus/ASCII

In this mode, the end of a block is recognized by a specific (CR/LF) character sequence. The USART manages this mechanism using the character match function.

By programming the LF ASCII code in the ADD[7:0] field and by activating the character match interrupt (CMIE=1), the software is informed when a LF has been received and can check the CR/LF in the DMA buffer.

27.5.9 USART parity control

Parity control (generation of parity bit in transmission and parity checking in reception) can be enabled by setting the PCE bit in the USART_CR1 register. Depending on the frame length defined by the M bits, the possible USART frame formats are as listed in [Table 166](#).

Table 166. Frame formats

M bits	PCE bit	USART frame ⁽¹⁾
00	0	SB 8-bit data STB
00	1	SB 7-bit data PB STB
01	0	SB 9-bit data STB
01	1	SB 8-bit data PB STB
10	0	SB 7-bit data STB
10	1	SB 6-bit data PB STB

- Legends: SB: start bit, STB: stop bit, PB: parity bit. In the data register, the PB is always taking the MSB position (9th, 8th or 7th, depending on the M bits value).

Even parity

The parity bit is calculated to obtain an even number of “1s” inside the frame of the 6, 7 or 8 LSB bits (depending on M bits values) and the parity bit.

As an example, if data=00110101, and 4 bits are set, then the parity bit 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 6, 7 or 8 LSB bits (depending on M bits values) and the parity bit.

As an example, if data=00110101 and 4 bits set, then the parity bit will be 1 if odd parity is selected (PS bit in USART_CR1 = 1).

Parity checking in reception

If the parity check fails, the PE flag is set in the USART_ISR register and an interrupt is generated if PEIE is set in the USART_CR1 register. The PE flag is cleared by software writing 1 to the PECF in the USART_ICR register.

Parity generation in transmission

If the PCE bit is set in USART_CR1, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit (even number of “1s” if even parity is selected (PS=0) or an odd number of “1s” if odd parity is selected (PS=1)).

27.5.10 USART LIN (local interconnection network) mode

This section is relevant only when LIN mode is supported. Please refer to [Section 27.4: USART implementation on page 888](#).

The LIN mode is selected by setting the LINEN bit in the USART_CR2 register. In LIN mode, the following bits must be kept cleared:

- STOP[1:0] and CLKEN in the USART_CR2 register,
- SCEN, HDSEL and IREN in the USART_CR3 register.

LIN transmission

The procedure explained in [Section 27.5.2: USART transmitter](#) has to be applied for LIN Master transmission. It must be the same as for normal USART transmission with the following differences:

- Clear the M bits to configure 8-bit word length.
- Set the LINEN bit to enter LIN mode. In this case, setting the SBKRQ bit sends 13 '0' bits as a break character. Then 2 bits of value '1' are sent to allow the next start detection.

LIN reception

When LIN mode is enabled, the break detection circuit is activated. The detection is totally independent from the normal USART receiver. A break can be detected whenever it occurs, during Idle state or during a frame.

When the receiver is enabled (RE=1 in USART_CR1), the circuit looks at the RX input for a start signal. The method for detecting start bits is the same when searching break characters or data. After a start bit has been detected, the circuit samples the next bits exactly like for the data (on the 8th, 9th and 10th samples). If 10 (when the LBDL = 0 in USART_CR2) or 11 (when LBDL=1 in USART_CR2) consecutive bits are detected as '0', and are followed by a delimiter character, the LBDF flag is set in USART_ISR. If the LBDIE bit=1, an interrupt is generated. Before validating the break, the delimiter is checked for as it signifies that the RX line has returned to a high level.

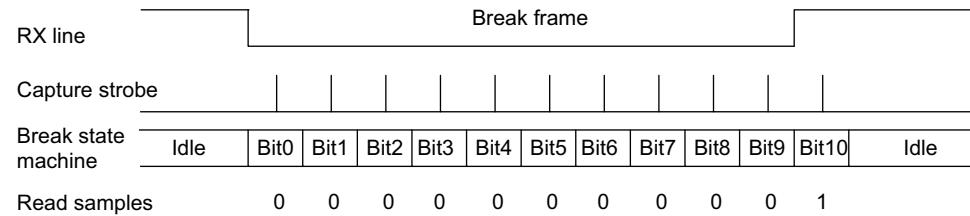
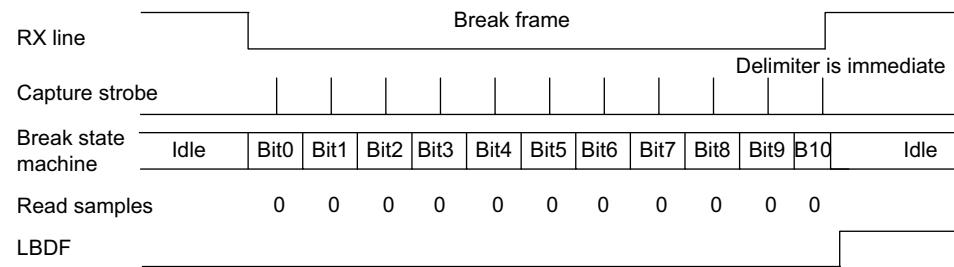
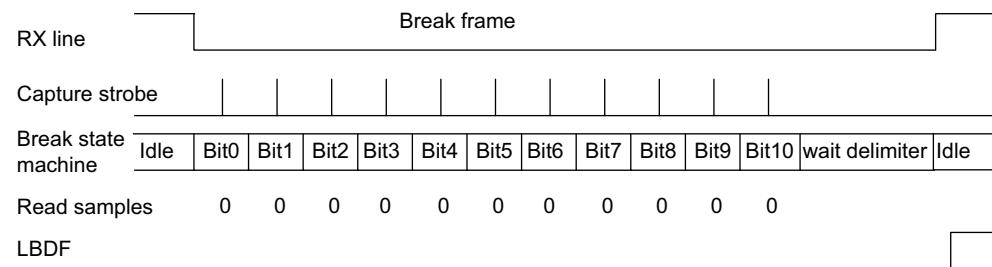
If a '1' is sampled before the 10 or 11 have occurred, the break detection circuit cancels the current detection and searches for a start bit again.

If the LIN mode is disabled (LINEN=0), the receiver continues working as normal USART, without taking into account the break detection.

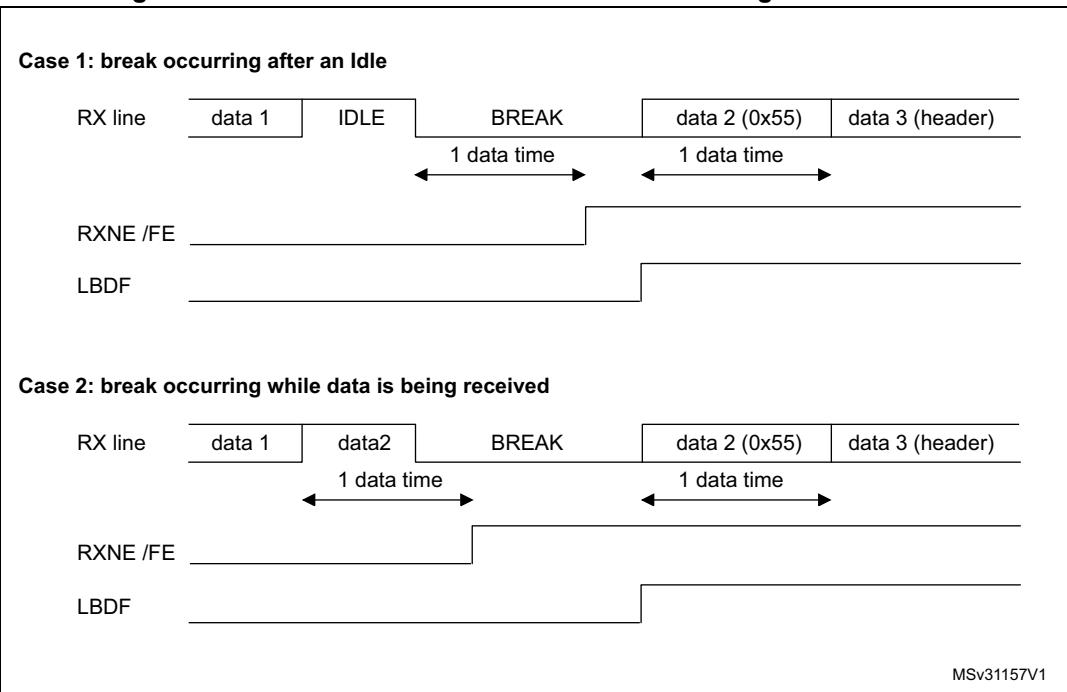
If the LIN mode is enabled (LINEN=1), as soon as a framing error occurs (i.e. stop bit detected at '0', which will be the case for any break frame), the receiver stops until the break detection circuit receives either a '1', if the break word was not complete, or a delimiter character if a break has been detected.

The behavior of the break detector state machine and the break flag is shown on the [Figure 326: Break detection in LIN mode \(11-bit break length - LBDL bit is set\) on page 911](#).

Examples of break frames are given on [Figure 327: Break detection in LIN mode vs. Framing error detection on page 912](#).

Figure 326. Break detection in LIN mode (11-bit break length - LBDL bit is set)**Case 1: break signal not long enough => break discarded, LBDF is not set****Case 2: break signal just long enough => break detected, LBDF is set****Case 3: break signal long enough => break detected, LBDF is set**

MSv31156V1

Figure 327. Break detection in LIN mode vs. Framing error detection

27.5.11 USART synchronous mode

The synchronous mode is selected by writing the CLKEN bit in the USART_CR2 register to 1. In synchronous mode, the following bits must be kept cleared:

- LINEN bit in the USART_CR2 register,
- SCEN, HDSEL and IREN bits in the USART_CR3 register.

In this mode, the USART can be used to control bidirectional synchronous serial communications in master mode. The CK pin is the output of the USART transmitter clock. No clock pulses are sent to the CK pin during start bit and stop bit. Depending on the state of the LBCL bit in the USART_CR2 register, clock pulses are, or are not, generated during the last valid data bit (address mark). The CPOL bit in the USART_CR2 register is used to select the clock polarity, and the CPHA bit in the USART_CR2 register is used to select the phase of the external clock (see [Figure 328](#), [Figure 329](#) and [Figure 330](#)).

During the Idle state, preamble and send break, the external CK clock is not activated.

In synchronous mode the USART transmitter works exactly like in asynchronous mode. But as CK is synchronized with TX (according to CPOL and CPHA), the data on TX is synchronous.

In this mode the USART receiver works in a different manner compared to the asynchronous mode. If RE=1, the data is sampled on CK (rising or falling edge, depending on CPOL and CPHA), without any oversampling. A setup and a hold time must be respected (which depends on the baud rate: 1/16 bit duration).

Note: The CK pin works in conjunction with the TX pin. Thus, the clock is provided only if the transmitter is enabled ($TE=1$) and data is being transmitted (the data register USART_TDR written). This means that it is not possible to receive synchronous data without transmitting data.

The LBCL, CPOL and CPHA bits have to be selected when the USART is disabled ($UE=0$) to ensure that the clock pulses function correctly.

Figure 328. USART example of synchronous transmission

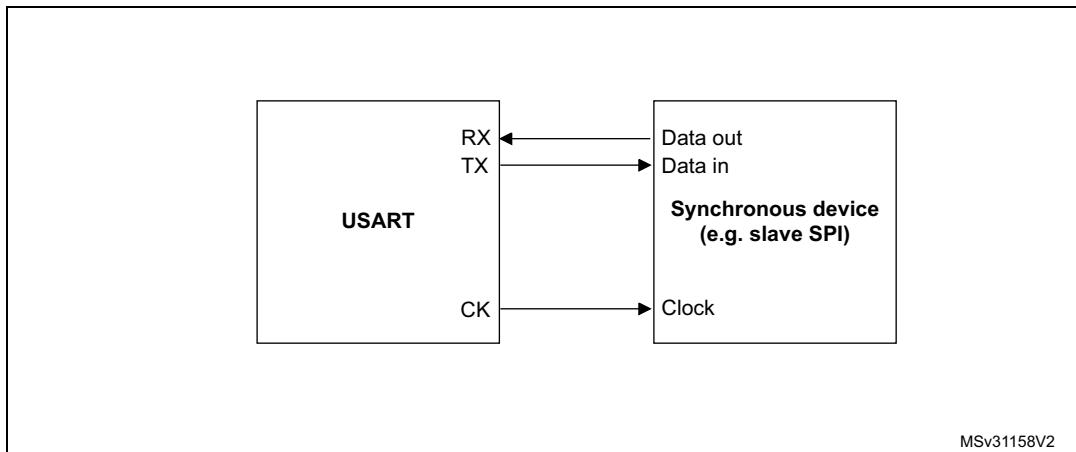


Figure 329. USART data clock timing diagram (M bits = 00)

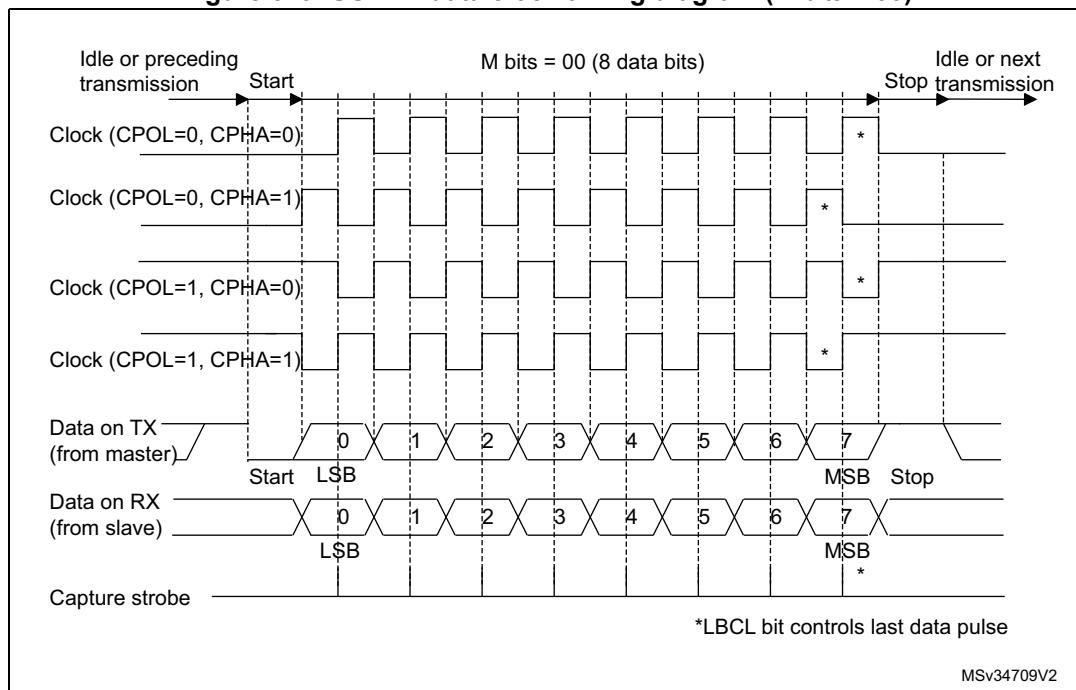
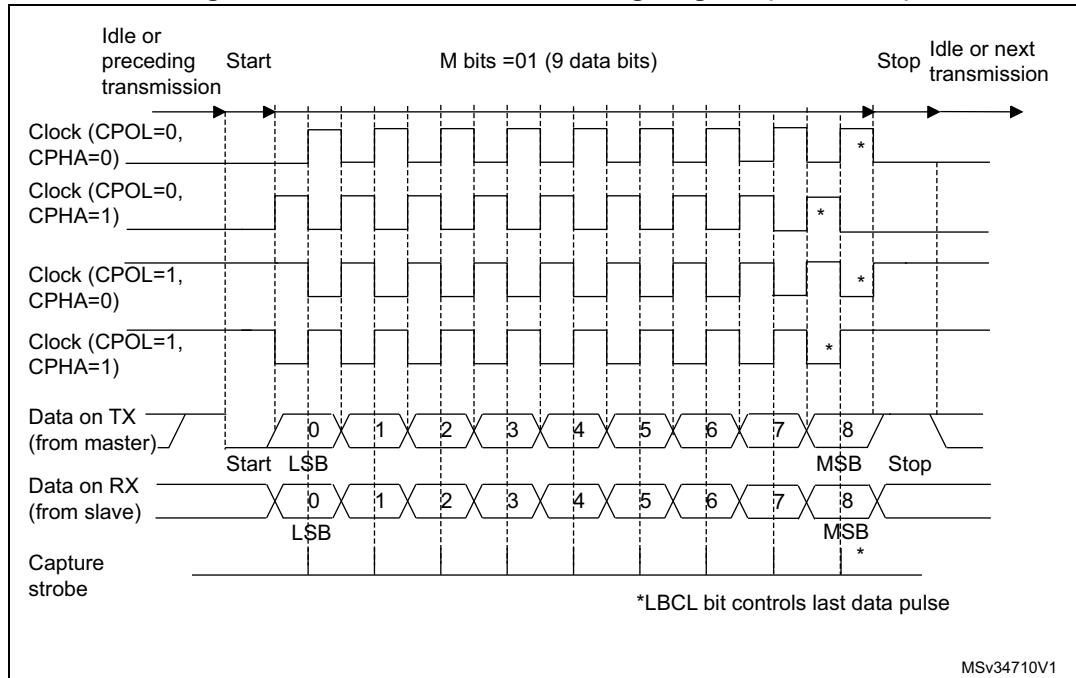
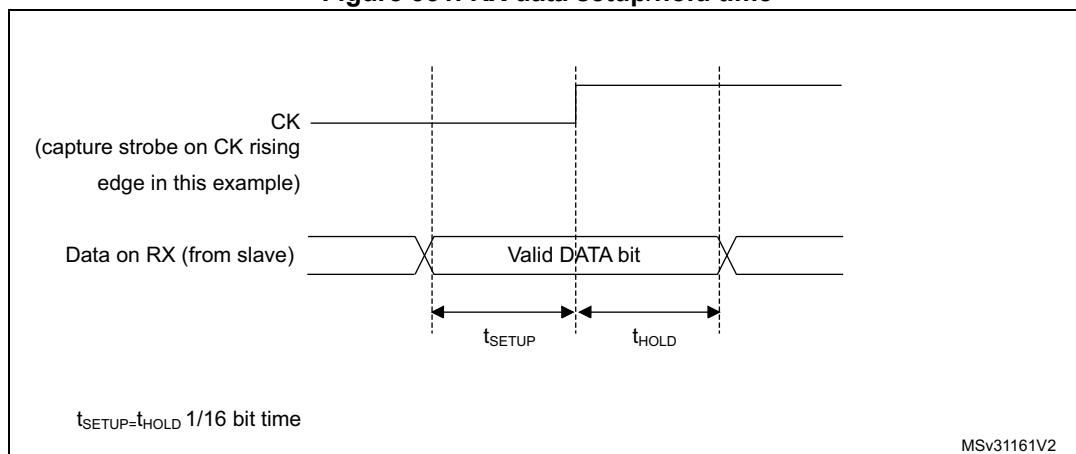


Figure 330. USART data clock timing diagram (M bits = 01)



MSv34710V1

Figure 331. RX data setup/hold time



MSv31161V2

Note: The function of CK is different in Smartcard mode. Refer to [Section 27.5.13: USART Smartcard mode](#) for more details.

27.5.12 USART Single-wire Half-duplex communication

Single-wire Half-duplex mode is selected by setting the HDSEL bit in the USART_CR3 register. In this mode, the following bits must be kept cleared:

- LINEN and CLKEN bits in the USART_CR2 register,
- SCEN and IREN bits in the USART_CR3 register.

The USART can be configured to follow a Single-wire Half-duplex protocol where the TX and RX lines are internally connected. The selection between half- and Full-duplex communication is made with a control bit HDSEL in USART_CR3.

As soon as HDSEL is written to 1:

- The TX and RX lines are internally connected
- The RX pin is no longer used
- The TX pin is always released when no data is transmitted. Thus, it acts as a standard I/O in idle or in reception. It means that the I/O must be configured so that TX is configured as alternate function open-drain with an external pull-up.

Apart from this, the communication protocol is similar to normal USART mode. Any conflicts on the line must be managed by software (by the use of a centralized arbiter, for instance). In particular, the transmission is never blocked by hardware and continues as soon as data is written in the data register while the TE bit is set.

27.5.13 USART Smartcard mode

This section is relevant only when Smartcard mode is supported. Please refer to [Section 27.4: USART implementation on page 888](#).

Smartcard mode is selected by setting the SCEN bit in the USART_CR3 register. In Smartcard mode, the following bits must be kept cleared:

- LINEN bit in the USART_CR2 register,
- HDSEL and IREN bits in the USART_CR3 register.

Moreover, the CLKEN bit may be set in order to provide a clock to the smartcard.

The smartcard interface is designed to support asynchronous protocol for smartcards as defined in the ISO 7816-3 standard. Both T=0 (character mode) and T=1 (block mode) are supported.

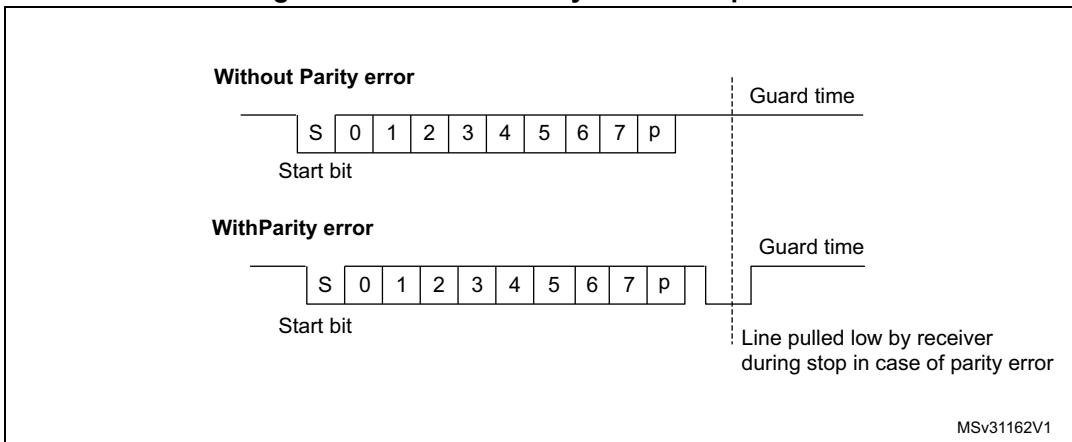
The USART should be configured as:

- 8 bits plus parity: where word length is set to 8 bits and PCE=1 in the USART_CR1 register
- 1.5 stop bits when transmitting and receiving data: where STOP=11 in the USART_CR2 register. It is also possible to choose 0.5 stop bit for receiving.

In T=0 (character) mode, the parity error is indicated at the end of each character during the guard time period.

[Figure 332](#) shows examples of what can be seen on the data line with and without parity error.

Figure 332. ISO 7816-3 asynchronous protocol



When connected to a smartcard, the TX output of the USART drives a bidirectional line that is also driven by the smartcard. The TX pin must be configured as open drain.

Smartcard mode implements a single wire half duplex communication protocol.

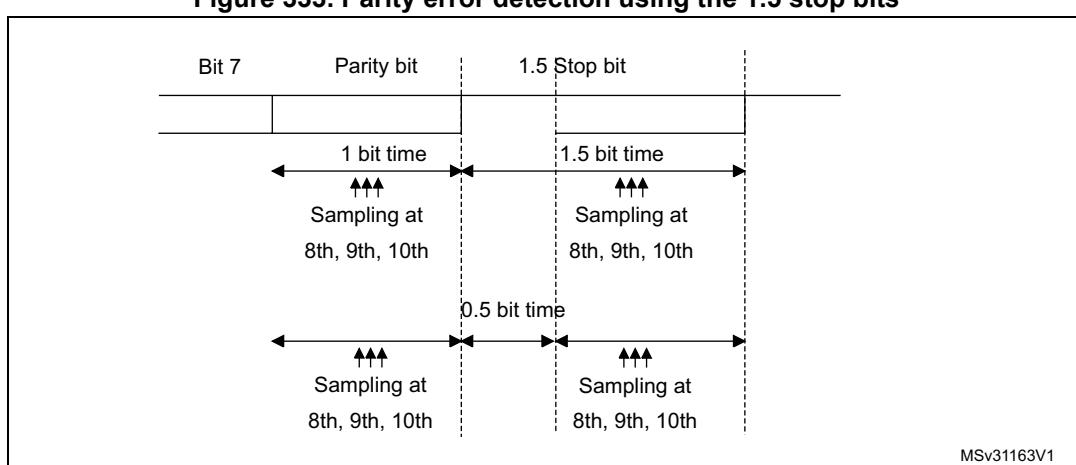
- Transmission of data from the transmit shift register is guaranteed to be delayed by a minimum of 1/2 baud clock. In normal operation a full transmit shift register starts shifting on the next baud clock edge. In Smartcard mode this transmission is further delayed by a guaranteed 1/2 baud clock.
- In transmission, if the smartcard detects a parity error, it signals this condition to the USART by driving the line low (NACK). This NACK signal (pulling transmit line low for 1 baud clock) causes a framing error on the transmitter side (configured with 1.5 stop bits). The USART can handle automatic re-sending of data according to the protocol. The number of retries is programmed in the SCARCNT bit field. If the USART continues receiving the NACK after the programmed number of retries, it stops transmitting and signals the error as a framing error. The TXE bit can be set using the TXFRQ bit in the USART_RQR register.
- Smartcard auto-retry in transmission: a delay of 2.5 baud periods is inserted between the NACK detection by the USART and the start bit of the repeated character. The TC bit is set immediately at the end of reception of the last repeated character (no guard-time). If the software wants to repeat it again, it must insure the minimum 2 baud periods required by the standard.
- If a parity error is detected during reception of a frame programmed with a 1.5 stop bits period, the transmit line is pulled low for a baud clock period after the completion of the receive frame. This is to indicate to the smartcard that the data transmitted to the USART has not been correctly received. A parity error is NACKed by the receiver if the NACK control bit is set, otherwise a NACK is not transmitted (to be used in T=1 mode). If the received character is erroneous, the RXNE/receive DMA request is not activated. According to the protocol specification, the smartcard must resend the same character. If the received character is still erroneous after the maximum number of retries specified in the SCARCNT bit field, the USART stops transmitting the NACK and signals the error as a parity error.
- Smartcard auto-retry in reception: the BUSY flag remains set if the USART NACKs the card but the card doesn't repeat the character.

- In transmission, the USART inserts the Guard Time (as programmed in the Guard Time register) between two successive characters. As the Guard Time is measured after the stop bit of the previous character, the GT[7:0] register must be programmed to the desired CGT (Character Guard Time, as defined by the 7816-3 specification) minus 12 (the duration of one character).
- The assertion of the TC flag can be delayed by programming the Guard Time register. In normal operation, TC is asserted when the transmit shift register is empty and no further transmit requests are outstanding. In Smartcard mode an empty transmit shift register triggers the Guard Time counter to count up to the programmed value in the Guard Time register. TC is forced low during this time. When the Guard Time counter reaches the programmed value TC is asserted high.
- The TCBGT flag can be used to detect the end of data transfer without waiting for guard time completion. This flag is set just after the end of frame transmission and if no NACK has been received from the card.
- The de-assertion of TC flag is unaffected by Smartcard mode.
- If a framing error is detected on the transmitter end (due to a NACK from the receiver), the NACK is not detected as a start bit by the receive block of the transmitter. According to the ISO protocol, the duration of the received NACK can be 1 or 2 baud clock periods.
- On the receiver side, if a parity error is detected and a NACK is transmitted the receiver does not detect the NACK as a start bit.

Note: A break character is not significant in Smartcard mode. A 0x00 data with a framing error is treated as data and not as a break.

No Idle frame is transmitted when toggling the TE bit. The Idle frame (as defined for the other configurations) is not defined by the ISO protocol.

Figure 333. Parity error detection using the 1.5 stop bits



The USART can provide a clock to the smartcard through the CK output. In Smartcard mode, CK is not associated to the communication but is simply derived from the internal peripheral input clock through a 5-bit prescaler. The division ratio is configured in the prescaler register USART_GTPR. CK frequency can be programmed from $f_{CK}/2$ to $f_{CK}/62$, where f_{CK} is the peripheral input clock.

Block mode (T=1)

In T=1 (block) mode, the parity error transmission is deactivated, by clearing the NACK bit in the USART_CR3 register.

When requesting a read from the smartcard, in block mode, the software must enable the receiver Timeout feature by setting the RTOEN bit in the USART_CR2 register and program the RTO bits field in the RTOR register to the BWT (block wait time) - 11 value. If no answer is received from the card before the expiration of this period, the RTOF flag will be set and a timeout interrupt will be generated (if RTOIE bit in the USART_CR1 register is set). If the first character is received before the expiration of the period, it is signaled by the RXNE interrupt.

Note: *The RXNE interrupt must be enabled even when using the USART in DMA mode to read from the smartcard in block mode. In parallel, the DMA must be enabled only after the first received byte.*

After the reception of the first character (RXNE interrupt), the RTO bit fields in the RTOR register must be programmed to the CWT (character wait time) - 11 value, in order to allow the automatic check of the maximum wait time between two consecutive characters. This time is expressed in baudtime units. If the smartcard does not send a new character in less than the CWT period after the end of the previous character, the USART signals this to the software through the RTOF flag and interrupt (when RTOIE bit is set).

Note: *The RTO counter starts counting:*

- From the end of the stop bit in case STOP = 00.
- From the end of the second stop bit in case of STOP = 10.
- 1 bit duration after the beginning of the STOP bit in case STOP = 11.
- From the beginning of the STOP bit in case STOP = 01.

As in the Smartcard protocol definition, the BWT/CWT values are defined from the beginning (start bit) of the last character. The RTO register must be programmed to BWT - 11 or CWT -11, respectively, taking into account the length of the last character itself.

A block length counter is used to count all the characters received by the USART. This counter is reset when the USART is transmitting (TXE=0). The length of the block is communicated by the smartcard in the third byte of the block (prologue field). This value must be programmed to the BLEN field in the USART_RTOR register. When using DMA mode, before the start of the block, this register field must be programmed to the minimum value (0x0). With this value, an interrupt is generated after the 4th received character. The software must read the LEN field (third byte), its value must be read from the receive buffer.

In interrupt driven receive mode, the length of the block may be checked by software or by programming the BLEN value. However, before the start of the block, the maximum value of BLEN (0xFF) may be programmed. The real value will be programmed after the reception of the third character.

If the block is using the LRC longitudinal redundancy check (1 epilogue byte), the BLEN=LEN. If the block is using the CRC mechanism (2 epilogue bytes), BLEN=LEN+1 must be programmed. The total block length (including prologue, epilogue and information fields) equals BLEN+4. The end of the block is signaled to the software through the EOBF flag and interrupt (when EOBIIE bit is set).

In case of an error in the block length, the end of the block is signaled by the RTO interrupt (Character wait Time overflow).

Note: *The error checking code (LRC/CRC) must be computed/verified by software.*

Direct and inverse convention

The Smartcard protocol defines two conventions: direct and inverse.

The direct convention is defined as: LSB first, logical bit value of 1 corresponds to a H state of the line and parity is even. In order to use this convention, the following control bits must be programmed: MSBFIRST=0, DATAINV=0 (default values).

The inverse convention is defined as: MSB first, logical bit value 1 corresponds to an L state on the signal line and parity is even. In order to use this convention, the following control bits must be programmed: MSBFIRST=1, DATAINV=1.

Note: *When logical data values are inverted (0=H, 1=L), the parity bit is also inverted in the same way.*

In order to recognize the card convention, the card sends the initial character, TS, as the first character of the ATR (Answer To Reset) frame. The two possible patterns for the TS are: LHHL LLL LLH and LHHL HHH LLH.

- (H) LHHL LLL LLH sets up the inverse convention: state L encodes value 1 and moment 2 conveys the most significant bit (MSB first). when decoded by inverse convention, the conveyed byte is equal to '3F'.
- (H) LHHL HHH LLH sets up the direct convention: state H encodes value 1 and moment 2 conveys the least significant bit (LSB first). when decoded by direct convention, the conveyed byte is equal to '3B'.

Character parity is correct when there is an even number of bits set to 1 in the nine moments 2 to 10.

As the USART does not know which convention is used by the card, it needs to be able to recognize either pattern and act accordingly. The pattern recognition is not done in hardware, but through a software sequence. Moreover, supposing that the USART is configured in direct convention (default) and the card answers with the inverse convention, TS = LHHL LLL LLH => the USART received character will be '03' and the parity will be odd.

Therefore, two methods are available for TS pattern recognition:

Method 1

The USART is programmed in standard Smartcard mode/direct convention. In this case, the TS pattern reception generates a parity error interrupt and error signal to the card.

- The parity error interrupt informs the software that the card didn't answer correctly in direct convention. Software then reprograms the USART for inverse convention
- In response to the error signal, the card retries the same TS character, and it will be correctly received this time, by the reprogrammed USART

Alternatively, in answer to the parity error interrupt, the software may decide to reprogram the USART and to also generate a new reset command to the card, then wait again for the TS.

Method 2

The USART is programmed in 9-bit/no-parity mode, no bit inversion. In this mode it receives any of the two TS patterns as:

- (H) LHHL LLL LLH = 0x103 -> inverse convention to be chosen
- (H) LHHL HHH LLH = 0x13B -> direct convention to be chosen

The software checks the received character against these two patterns and, if any of them match, then programs the USART accordingly for the next character reception.

If none of the two is recognized, a card reset may be generated in order to restart the negotiation.

27.5.14 USART IrDA SIR ENDEC block

This section is relevant only when IrDA mode is supported. Please refer to [Section 27.4: USART implementation on page 888](#).

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 334](#)).

The SIR Transmit encoder modulates the Non Return to Zero (NRZ) transmit bit stream output from USART. The output pulse stream is transmitted to an external output driver and infrared LED. USART supports only bit rates up to 115.2 Kbps for the SIR ENDEC. In normal mode the transmitted pulse width is specified as 3/16 of a bit period.

The SIR receive decoder demodulates the return-to-zero bit stream from the infrared detector and outputs the received NRZ serial bit stream to the USART. The decoder input is normally high (marking state) in the Idle state. The transmit encoder output has the opposite polarity to the decoder input. A start bit is detected when the decoder input is low.

- IrDA is a half duplex communication protocol. If the Transmitter is busy (when the USART is sending data to the IrDA encoder), any data on the IrDA receive line is ignored by the IrDA decoder and if the Receiver is busy (when the USART is receiving decoded data from the IrDA decoder), data on the TX from the USART to IrDA is not encoded. While receiving data, transmission should be avoided as the data to be transmitted could be corrupted.
- A 0 is transmitted as a high pulse and a 1 is transmitted as a 0. The width of the pulse is specified as 3/16th of the selected bit period in normal mode (see [Figure 335](#)).
- The SIR decoder converts the IrDA compliant receive signal into a bit stream for USART.
- The SIR receive logic interprets a high state as a logic one and low pulses as logic zeros.
- The transmit encoder output has the opposite polarity to the decoder input. The SIR output is in low state when Idle.

- The IrDA specification requires the acceptance of pulses greater than $1.41 \mu s$. The acceptable pulse width is programmable. Glitch detection logic on the receiver end filters out pulses of width less than 2 PSC periods (PSC is the prescaler value programmed in the USART_GTPR). Pulses of width less than 1 PSC period are always rejected, but those of width greater than one and less than two periods may be accepted or rejected, those greater than 2 periods 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 ($1.42 \text{ MHz} < \text{PSC} < 2.12 \text{ MHz}$). A low-power mode programmable divisor divides the system clock to achieve this value.

Receiver

Receiving in low-power mode is similar to receiving in normal mode. For glitch detection the USART should discard pulses of duration shorter than 1 PSC period. A valid low is accepted only if its duration is greater than 2 periods of the IrDA low-power Baud clock (PSC value in the USART_GTPR).

Note: *A pulse of width less than two and greater than one PSC period(s) may or may not be rejected.*

The receiver set up time should be managed by software. The IrDA physical layer specification specifies a minimum of 10 ms delay between transmission and reception (IrDA is a half duplex protocol).

Figure 334. IrDA SIR ENDEC- block diagram

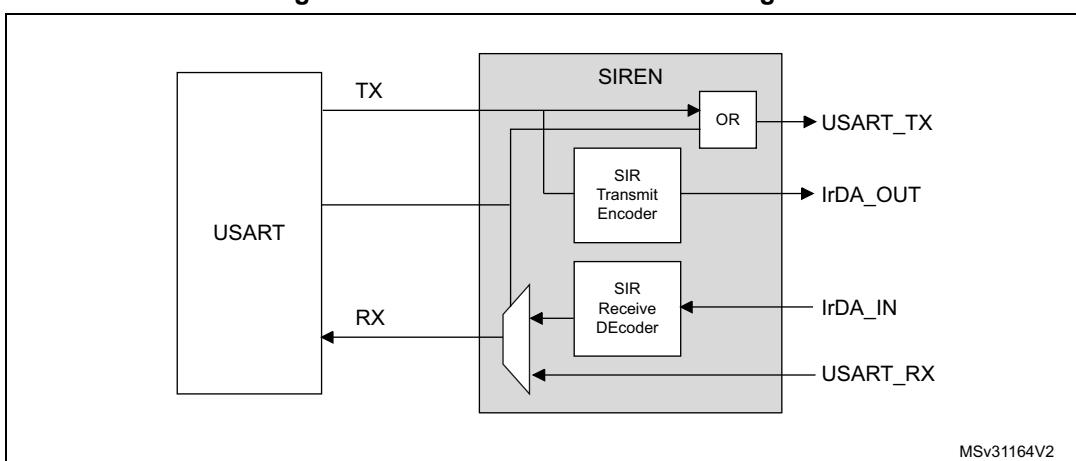
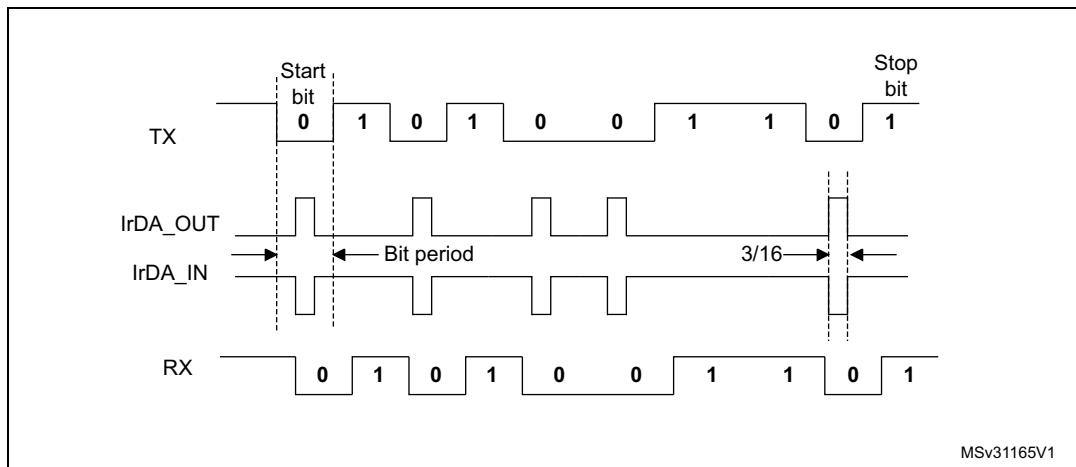


Figure 335. IrDA data modulation (3/16) -Normal Mode



27.5.15 USART continuous communication in DMA mode

The USART is capable of performing continuous communication using the DMA. The DMA requests for Rx buffer and Tx buffer are generated independently.

Note: Please refer to [Section 27.4: USART implementation on page 888](#) to determine if the DMA mode is supported. If DMA is not supported, use the USART as explained in [Section 27.5.2: USART transmitter](#) or [Section 27.5.3: USART receiver](#). To perform continuous communication, the user can clear the TXE/RXNE flags in the USART_ISR register.

Transmission using DMA

DMA mode can be enabled for transmission by setting DMAT bit in the USART_CR3 register. Data is loaded from a SRAM area configured using the DMA peripheral (refer to [Section 8: Direct memory access controller \(DMA\) on page 242](#)) to the USART_TDR register whenever the TXE bit is set. To map a DMA channel for USART transmission, use the following procedure (x denotes the channel number):

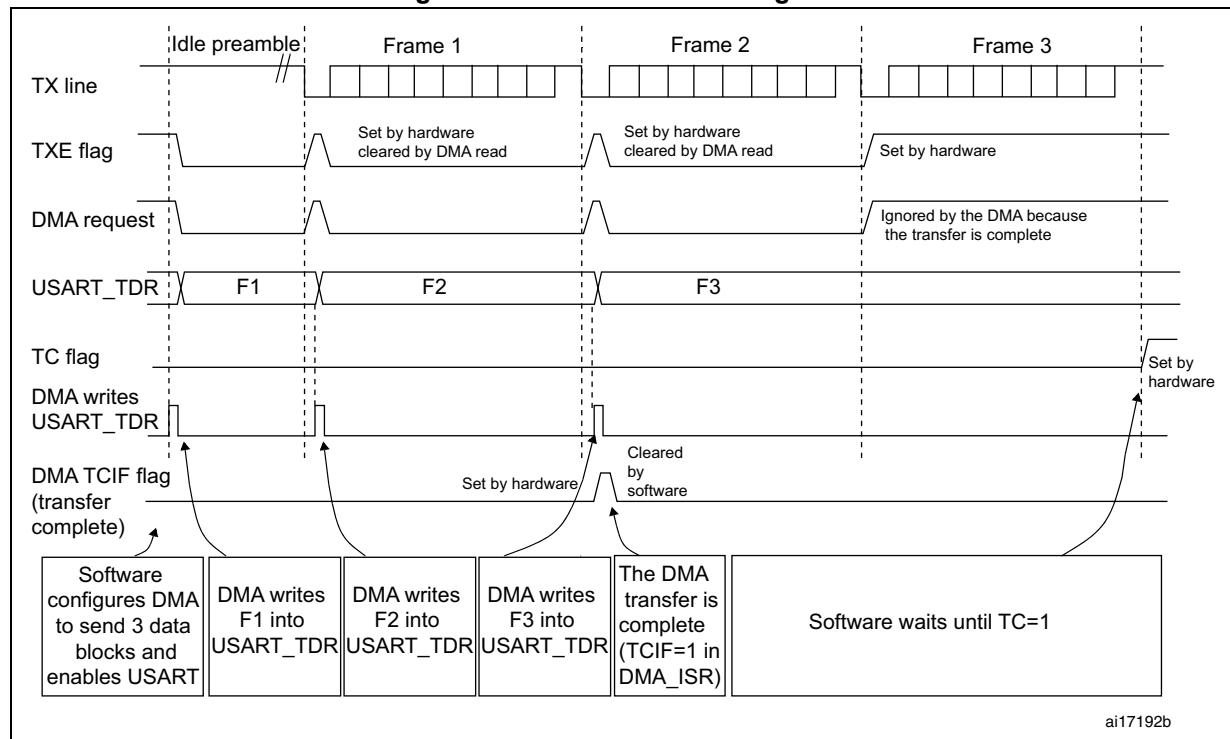
1. Write the USART_TDR register address in the DMA control register to configure it as the destination of the transfer. The data is moved to this address from memory after each TXE event.
2. Write the memory address in the DMA control register to configure it as the source of the transfer. The data is loaded into the USART_TDR register from this memory area after each TXE event.
3. Configure the total number of bytes to be transferred to the DMA control register.
4. Configure the channel priority in the DMA register
5. Configure DMA interrupt generation after half/ full transfer as required by the application.
6. Clear the TC flag in the USART_ISR register by setting the TCCF bit in the USART_ICR register.
7. Activate the channel in the DMA register.

When the number of data transfers programmed in the DMA Controller is reached, the DMA controller generates an interrupt on the DMA channel interrupt vector.

In transmission mode, once the DMA has written all the data to be transmitted (the TCIF flag is set in the DMA_ISR register), the TC flag can be monitored to make sure that the USART

communication is complete. This is required to avoid corrupting the last transmission before disabling the USART or entering Stop mode. Software must wait until TC=1. The TC flag remains cleared during all data transfers and it is set by hardware at the end of transmission of the last frame.

Figure 336. Transmission using DMA



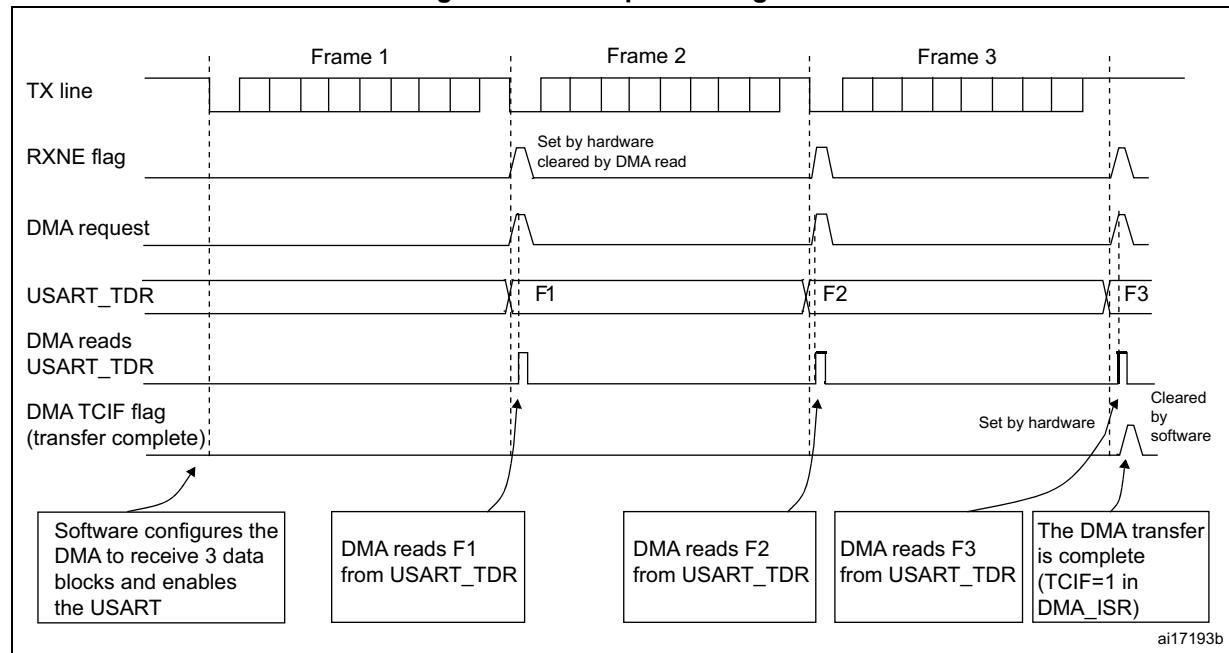
Reception using DMA

DMA mode can be enabled for reception by setting the DMAR bit in USART_CR3 register. Data is loaded from the USART_RDR register to a SRAM area configured using the DMA peripheral (refer to [Section 8: Direct memory access controller \(DMA\)](#)) whenever a data byte is received. To map a DMA channel for USART reception, use the following procedure:

1. Write the USART_RDR register address in the DMA control register to configure it as the source of the transfer. The data is moved from this address to the memory after each RXNE event.
2. Write the memory address in the DMA control register to configure it as the destination of the transfer. The data is loaded from USART_RDR to this memory area after each RXNE event.
3. Configure the total number of bytes to be transferred to the DMA control register.
4. Configure the channel priority in the DMA control register
5. Configure interrupt generation after half/ full transfer as required by the application.
6. Activate the channel in the DMA control register.

When the number of data transfers programmed in the DMA Controller is reached, the DMA controller generates an interrupt on the DMA channel interrupt vector.

Figure 337. Reception using DMA



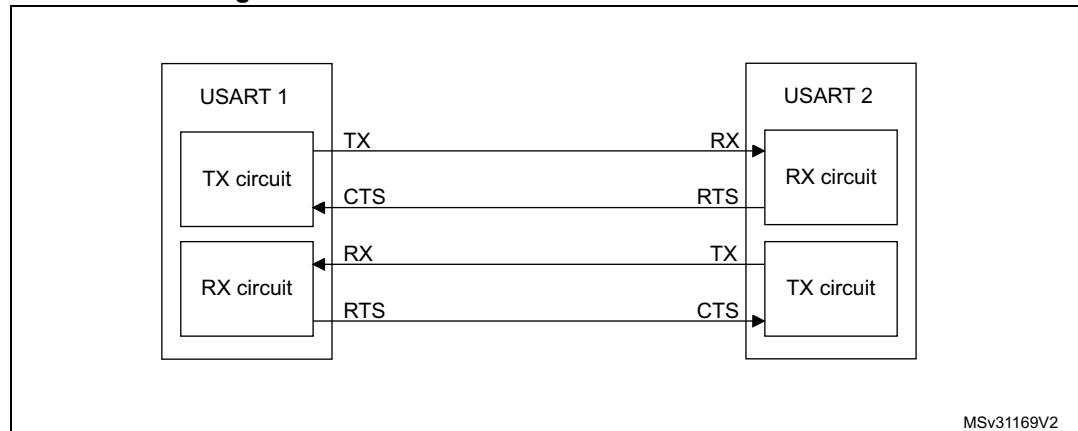
Error flagging and interrupt generation in multibuffer communication

In multibuffer communication if any error occurs during the transaction the error flag is asserted after the current byte. An interrupt is generated if the interrupt enable flag is set. For framing error, overrun error and noise flag which are asserted with RXNE in single byte reception, there is a separate error flag interrupt enable bit (EIE bit in the USART_CR3 register), which, if set, enables an interrupt after the current byte if any of these errors occur.

27.5.16 RS232 hardware flow control and RS485 driver enable using USART

It is possible to control the serial data flow between 2 devices by using the CTS input and the RTS output. The [Figure 338](#) shows how to connect 2 devices in this mode:

Figure 338. Hardware flow control between 2 USARTs

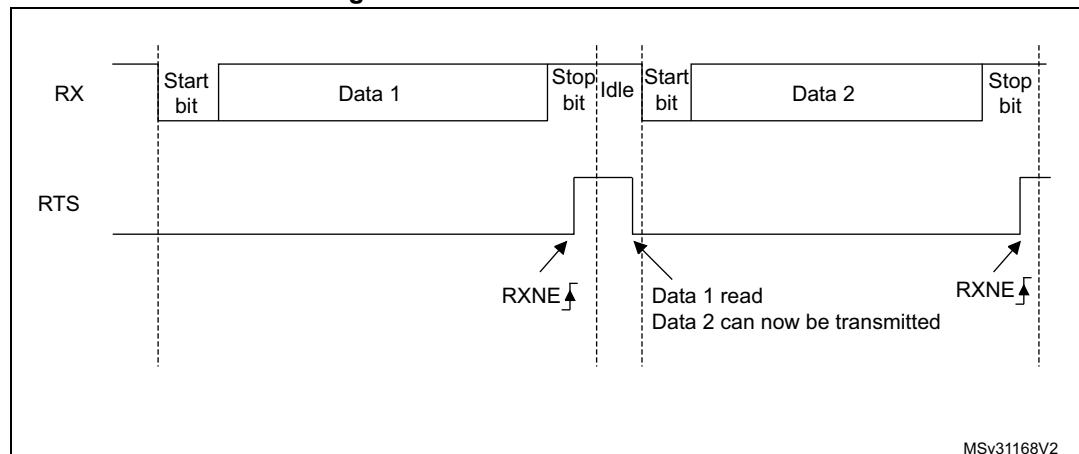


RS232 RTS and CTS flow control can be enabled independently by writing the RTSE and CTSE bits respectively to 1 (in the USART_CR3 register).

RS232 RTS flow control

If the RTS flow control is enabled (RTSE=1), then RTS is asserted (tied low) as long as the USART receiver is ready to receive a new data. When the receive register is full, RTS is de-asserted, indicating that the transmission is expected to stop at the end of the current frame. [Figure 339](#) shows an example of communication with RTS flow control enabled.

Figure 339. RS232 RTS flow control

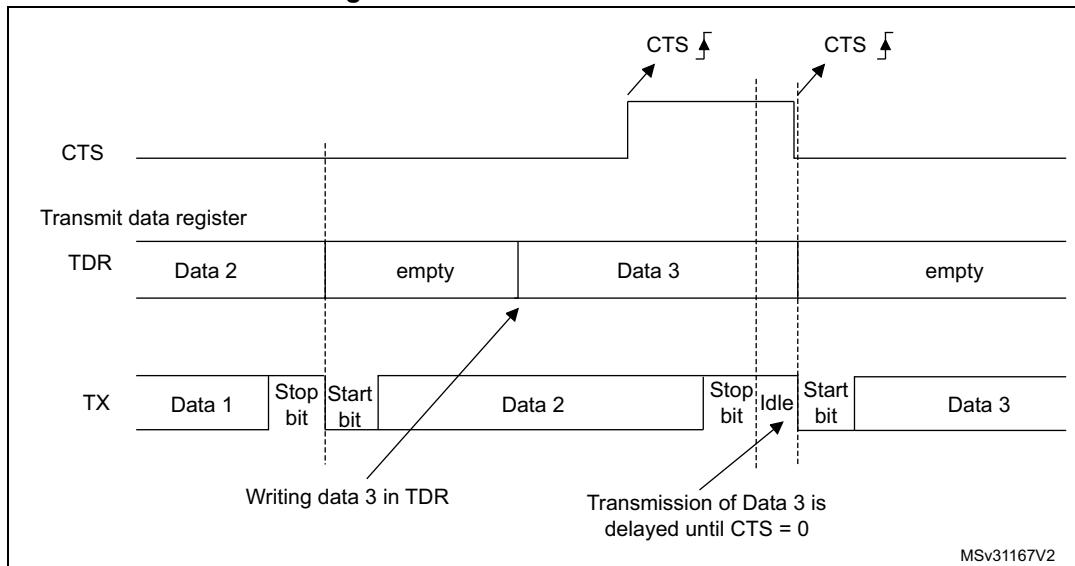


RS232 CTS flow control

If the CTS flow control is enabled (CTSE=1), then the transmitter checks the CTS input before transmitting the next frame. If CTS is asserted (tied low), then the next data is transmitted (assuming that data is to be transmitted, in other words, if TXE=0), else the transmission does not occur. When CTS is de-asserted during a transmission, the current transmission is completed before the transmitter stops.

When CTSE=1, the CTSIF status bit is automatically set by hardware as soon as the CTS input toggles. It indicates when the receiver becomes ready or not ready for communication. An interrupt is generated if the CTSIE bit in the USART_CR3 register is set. [Figure 340](#) shows an example of communication with CTS flow control enabled.

Figure 340. RS232 CTS flow control



Note: For correct behavior, CTS must be asserted at least 3 USART clock source periods before the end of the current character. In addition it should be noted that the CTSCF flag may not be set for pulses shorter than 2 x PCLK periods.

RS485 Driver Enable

The driver enable feature is enabled by setting bit DEM in the USART_CR3 control register. This allows the user to activate the external transceiver control, through the DE (Driver Enable) signal. The assertion time is the time between the activation of the DE signal and the beginning of the START bit. It is programmed using the DEAT [4:0] bit fields in the USART_CR1 control register. The de-assertion time is the time between the end of the last stop bit, in a transmitted message, and the de-activation of the DE signal. It is programmed using the DEDT [4:0] bit fields in the USART_CR1 control register. The polarity of the DE signal can be configured using the DEP bit in the USART_CR3 control register.

In USART, the DEAT and DEDT are expressed in sample time units (1/8 or 1/16 bit duration, depending on the oversampling rate).

27.6 USART low-power modes

Table 167. Effect of low-power modes on the USART

Mode	Description
Sleep	No effect. USART interrupt causes the device to exit Sleep mode.
Stop	No effect.
Standby	The USART is powered down and must be reinitialized when the device has exited from Standby mode.

27.7 USART interrupts

Table 168. USART interrupt requests

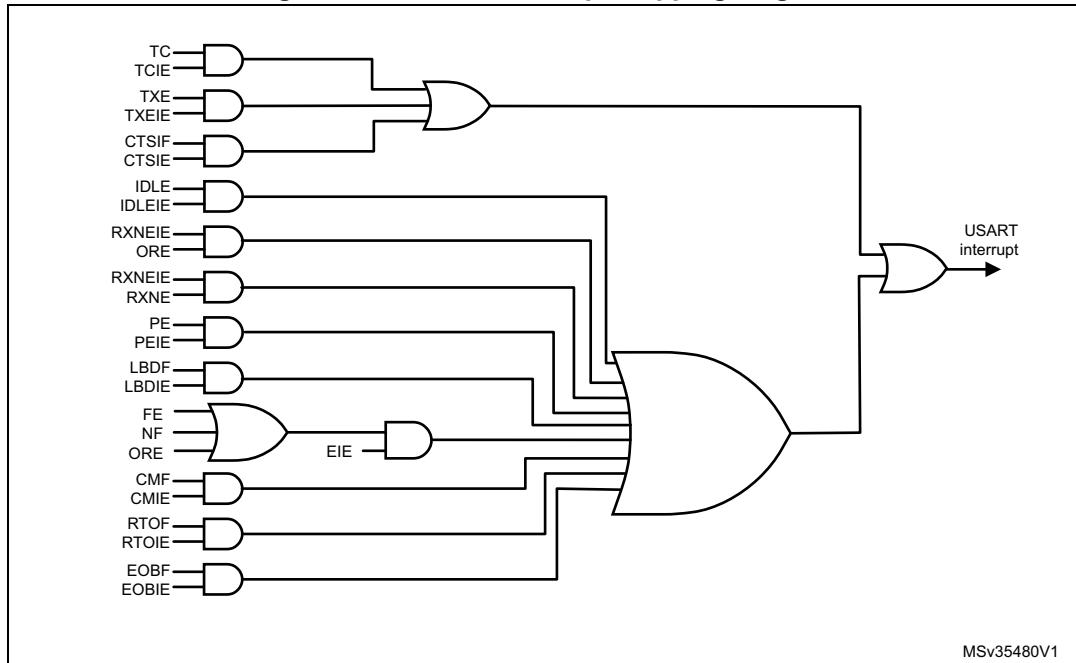
Interrupt event	Event flag	Enable Control bit
Transmit data register empty	TXE	TXEIE
CTS interrupt	CTSIF	CTSIE
Transmission Complete	TC	TCIE
Receive data register not empty (data ready to be read)	RXNE	RXNEIE
Overrun error detected	ORE	
Idle line detected	IDLE	IDLEIE
Parity error	PE	PEIE
LIN break	LBDF	LBDIE
Noise Flag, Overrun error and Framing Error in multibuffer communication.	NF or ORE or FE	EIE
Character match	CMF	CMIE
Receiver timeout	RTOF	RTOIE
End of Block	EOBF	EOBIE
Transmission complete before guard time	TCBGT	TCBGTE

The USART interrupt events are connected to the same interrupt vector (see [Figure 341](#)).

- During transmission: Transmission Complete, Transmission complete before guard time, Clear to Send, Transmit data Register empty or Framing error (in Smartcard mode) interrupt.
- During reception: Idle Line detection, Overrun error, Receive data register not empty, Parity error, LIN break detection, Noise Flag, Framing Error, Character match, etc.

These events generate an interrupt if the corresponding Enable Control Bit is set.

Figure 341. USART interrupt mapping diagram



27.8 USART registers

Refer to [Section 1.2 on page 53](#) for a list of abbreviations used in register descriptions.

27.8.1 Control register 1 (USART_CR1)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16		
Res.	Res.	Res.	M1	EOBIE	RTOIE	DEAT[4:0]						DEDT[4:0]					
			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
OVER8	CMIE	MME	M0	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	Res.	UE		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw		

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

Bit 28 M1: Word length

This bit, with bit 12 (M0), determines the word length. It is set or cleared by software.

M[1:0] = 00: 1 Start bit, 8 data bits, n stop bits

M[1:0] = 01: 1 Start bit, 9 data bits, n stop bits

M[1:0] = 10: 1 Start bit, 7 data bits, n stop bits

This bit can only be written when the USART is disabled (UE=0).

Note: Not all modes are supported In 7-bit data length mode. Refer to [Section 27.4: USART implementation](#) for details.

Bit 27 EOBIE: End of Block interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A USART interrupt is generated when the EOBF flag is set in the USART_ISR register.

Note: If the USART does not support Smartcard mode, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bit 26 RTOIE: Receiver timeout interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated when the RTOF bit is set in the USART_ISR register.

Note: If the USART does not support the Receiver timeout feature, this bit is reserved and must be kept at reset value. [Section 27.4: USART implementation on page 888](#).

Bits 25:21 DEAT[4:0]: Driver Enable assertion time

This 5-bit value defines the time between the activation of the DE (Driver Enable) signal and the beginning of the start bit. It is expressed in sample time units (1/8 or 1/16 bit duration, depending on the oversampling rate).

This bit field can only be written when the USART is disabled (UE=0).

Note: If the Driver Enable feature is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bits 20:16 **DEDT[4:0]**: Driver Enable de-assertion time

This 5-bit value defines the time between the end of the last stop bit, in a transmitted message, and the de-activation of the DE (Driver Enable) signal. It is expressed in sample time units (1/8 or 1/16 bit duration, depending on the oversampling rate).

If the USART_TDR register is written during the DEDT time, the new data is transmitted only when the DEDT and DEAT times have both elapsed.

This bit field can only be written when the USART is disabled (UE=0).

Note: If the Driver Enable feature is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bit 15 **OVER8**: Oversampling mode

- 0: Oversampling by 16
- 1: Oversampling by 8

This bit can only be written when the USART is disabled (UE=0).

Note: In LIN, IrDA and modes, this bit must be kept at reset value.

Bit 14 **CMIE**: Character match interrupt enable

This bit is set and cleared by software.

- 0: Interrupt is inhibited
- 1: A USART interrupt is generated when the CMF bit is set in the USART_ISR register.

Bit 13 **MME**: Mute mode enable

This bit activates the mute mode function of the USART. When set, the USART can switch between the active and mute modes, as defined by the WAKE bit. It is set and cleared by software.

- 0: Receiver in active mode permanently
- 1: Receiver can switch between mute mode and active mode.

Bit 12 **M0**: Word length

This bit, with bit 28 (M1), determines the word length. It is set or cleared by software. See Bit 28 (M1) description.

This bit can only be written when the USART is disabled (UE=0).

Bit 11 **WAKE**: Receiver wakeup method

This bit determines the USART wakeup method from Mute mode. It is set or cleared by software.

- 0: Idle line
- 1: Address mark

This bit field can only be written when the USART is disabled (UE=0).

Bit 10 **PCE**: Parity control enable

This bit selects the hardware parity control (generation and detection). When the parity control is enabled, the computed parity is inserted at the MSB position (9th bit if M=1; 8th bit if M=0) and parity is checked on the received data. This bit is set and cleared by software. Once it is set, PCE is active after the current byte (in reception and in transmission).

- 0: Parity control disabled
- 1: Parity control enabled

This bit field can only be written when the USART is disabled (UE=0).

Bit 9 **PS**: Parity selection

This bit selects the odd or even parity when the parity generation/detection is enabled (PCE bit set). It is set and cleared by software. The parity will be selected after the current byte.

- 0: Even parity
- 1: Odd parity

This bit field can only be written when the USART is disabled (UE=0).

Bit 8 **PEIE**: PE interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A USART interrupt is generated whenever PE=1 in the USART_ISR register

Bit 7 **TXEIE**: interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A USART interrupt is generated whenever TXE=1 in the USART_ISR register

Bit 6 **TCIE**: Transmission complete interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A USART interrupt is generated whenever TC=1 in the USART_ISR register

Bit 5 **RXNEIE**: RXNE interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A USART interrupt is generated whenever ORE=1 or RXNE=1 in the USART_ISR register

Bit 4 **IDLEIE**: IDLE interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A USART interrupt is generated whenever IDLE=1 in the USART_ISR register

Bit 3 **TE**: Transmitter enable

This bit enables the transmitter. It is set and cleared by software.

0: Transmitter is disabled

1: Transmitter is enabled

Note: During transmission, a “0” pulse on the TE bit (“0” followed by “1”) sends a preamble (idle line) after the current word, except in Smartcard mode. In order to generate an idle character, the TE must not be immediately written to 1. In order to ensure the required duration, the software can poll the TEACK bit in the USART_ISR register.

In Smartcard mode, when TE is set there is a 1 bit-time delay before the transmission starts.

Bit 2 **RE**: Receiver enable

This bit enables the receiver. It is set and cleared by software.

0: Receiver is disabled

1: Receiver is enabled and begins searching for a start bit

Bit 1 Reserved, must be kept at reset value.

Bit 0 **UE**: USART enable

When this bit is cleared, the USART prescalers and outputs are stopped immediately, and current operations are discarded. The configuration of the USART is kept, but all the status flags, in the USART_ISR are set to their default values. This bit is set and cleared by software.

0: USART prescaler and outputs disabled, low-power mode

1: USART enabled

Note: In order to go into low-power mode without generating errors on the line, the TE bit must be reset before and the software must wait for the TC bit in the USART_ISR to be set before resetting the UE bit.

The DMA requests are also reset when UE = 0 so the DMA channel must be disabled before resetting the UE bit.

27.8.2 Control register 2 (USART_CR2)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADD[7:4]				ADD[3:0]				RTOEN	ABRMOD[1:0]		ABREN	MSBFIRST	DATAINV	TXINV	RXINV
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWAP	LINEN	STOP[1:0]		CLKEN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	ADDM7	Res.	Res.	Res.	Res.
rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw				

Bits 31:28 **ADD[7:4]: Address of the USART node**

This bit-field gives the address of the USART node or a character code to be recognized.

This is used in multiprocessor communication during Mute mode, for wakeup with 7-bit address mark detection. The MSB of the character sent by the transmitter should be equal to 1. It may also be used for character detection during normal reception, Mute mode inactive (for example, end of block detection in ModBus protocol). In this case, the whole received character (8-bit) is compared to the ADD[7:0] value and CMF flag is set on match.

This bit field can only be written when reception is disabled (RE = 0) or the USART is disabled (UE=0)

Bits 27:24 **ADD[3:0]: Address of the USART node**

This bit-field gives the address of the USART node or a character code to be recognized.

This is used in multiprocessor communication during Mute mode, for wakeup with address mark detection.

This bit field can only be written when reception is disabled (RE = 0) or the USART is disabled (UE=0)

Bit 23 **RTOEN: Receiver timeout enable**

This bit is set and cleared by software.

0: Receiver timeout feature disabled.

1: Receiver timeout feature enabled.

When this feature is enabled, the RTOF flag in the USART_ISR register is set if the RX line is idle (no reception) for the duration programmed in the RTOR (receiver timeout register).

Note: If the USART does not support the Receiver timeout feature, this bit is reserved and must be kept at reset value. Please refer to Section 27.4: USART implementation on page 888.

Bits 22:21 **ABRMODE[1:0]: Auto baud rate mode**

These bits are set and cleared by software.

00: Measurement of the start bit is used to detect the baud rate.

01: Falling edge to falling edge measurement. (the received frame must start with a single bit = 1 -> Frame = Start10xxxxxx)

10: 0x7F frame detection.

11: 0x55 frame detection

This bit field can only be written when ABREN = 0 or the USART is disabled (UE=0).

Note: If DATAINV=1 and/or MSBFIRST=1 the patterns must be the same on the line, for example 0xAA for MSBFIRST)

If the USART does not support the auto baud rate feature, this bit is reserved and must be kept at reset value. Please refer to Section 27.4: USART implementation on page 888.

Bit 20 **ABREN**: Auto baud rate enable

This bit is set and cleared by software.

0: Auto baud rate detection is disabled.

1: Auto baud rate detection is enabled.

Note: If the USART does not support the auto baud rate feature, this bit is reserved and must be kept at reset value. Please refer to Section 27.4: USART implementation on page 888.

Bit 19 **MSBFIRST**: Most significant bit first

This bit is set and cleared by software.

0: data is transmitted/received with data bit 0 first, following the start bit.

1: data is transmitted/received with the MSB (bit 7/8/9) first, following the start bit.

This bit field can only be written when the USART is disabled (UE=0).

Bit 18 **DATAINV**: Binary data inversion

This bit is set and cleared by software.

0: Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)

1: Logical data from the data register are send/received in negative/inverse logic. (1=L, 0=H). The parity bit is also inverted.

This bit field can only be written when the USART is disabled (UE=0).

Bit 17 **TXINV**: TX pin active level inversion

This bit is set and cleared by software.

0: TX pin signal works using the standard logic levels ($V_{DD} = 1/\text{idle}$, Gnd=0/mark)

1: TX pin signal values are inverted. ($V_{DD} = 0/\text{mark}$, Gnd=1/idle).

This allows the use of an external inverter on the TX line.

This bit field can only be written when the USART is disabled (UE=0).

Bit 16 **RXINV**: RX pin active level inversion

This bit is set and cleared by software.

0: RX pin signal works using the standard logic levels ($V_{DD} = 1/\text{idle}$, Gnd=0/mark)

1: RX pin signal values are inverted. ($V_{DD} = 0/\text{mark}$, Gnd=1/idle).

This allows the use of an external inverter on the RX line.

This bit field can only be written when the USART is disabled (UE=0).

Bit 15 **SWAP**: Swap TX/RX pins

This bit is set and cleared by software.

0: TX/RX pins are used as defined in standard pinout

1: The TX and RX pins functions are swapped. This allows to work in the case of a cross-wired connection to another USART.

This bit field can only be written when the USART is disabled (UE=0).

Bit 14 **LINEN**: LIN mode enable

This bit is set and cleared by software.

0: LIN mode disabled

1: LIN mode enabled

The LIN mode enables the capability to send LIN synchronous breaks (13 low bits) using the SBKRQ bit in the USART_RQR register, and to detect LIN Sync breaks.

This bit field can only be written when the USART is disabled (UE=0).

Note: If the USART does not support LIN mode, this bit is reserved and must be kept at reset value. Please refer to Section 27.4: USART implementation on page 888.

Bits 13:12 **STOP[1:0]**: STOP bits

These bits are used for programming the stop bits.

- 00: 1 stop bit
- 01: 0.5 stop bit
- 10: 2 stop bits
- 11: 1.5 stop bits

This bit field can only be written when the USART is disabled (UE=0).

Bit 11 **CLKEN**: Clock enable

This bit allows the user to enable the CK pin.

- 0: CK pin disabled
- 1: CK pin enabled

This bit can only be written when the USART is disabled (UE=0).

Note: If neither synchronous mode nor Smartcard mode is supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

In order to provide correctly the CK clock to the Smartcard when CK is always available When CLKEN = 1, regardless of the UE bit value, the steps below must be respected:

- UE = 0
- SCEN = 1
- GTPR configuration (If PSC needs to be configured, it is recommended to configure PSC and GT in a single access to USART_GTPR register).
- CLKEN= 1
- UE = 1

Bit 10 **CPOL**: Clock polarity

This bit allows the user to select the polarity of the clock output on the CK pin in synchronous mode. It works in conjunction with the CPHA bit to produce the desired clock/data relationship

- 0: Steady low value on CK pin outside transmission window
- 1: Steady high value on CK pin outside transmission window

This bit can only be written when the USART is disabled (UE=0).

Note: If synchronous mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bit 9 **CPHA**: Clock phase

This bit is used to select the phase of the clock output on the CK pin in synchronous mode. It works in conjunction with the CPOL bit to produce the desired clock/data relationship (see [Figure 329](#) and [Figure 330](#))

- 0: The first clock transition is the first data capture edge
- 1: The second clock transition is the first data capture edge

This bit can only be written when the USART is disabled (UE=0).

Note: If synchronous mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bit 8 **LBCL**: Last bit clock pulse

This bit is used to select whether the clock pulse associated with the last data bit transmitted (MSB) has to be output on the CK pin in synchronous mode.

- 0: The clock pulse of the last data bit is not output to the CK pin
- 1: The clock pulse of the last data bit is output to the CK pin

Caution: The last bit is the 7th or 8th or 9th data bit transmitted depending on the 7 or 8 or 9 bit format selected by the M bits in the USART_CR1 register.

This bit can only be written when the USART is disabled (UE=0).

Note: If synchronous mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bit 7 Reserved, must be kept at reset value.

Bit 6 **LBDIE**: LIN break detection interrupt enable

Break interrupt mask (break detection using break delimiter).

0: Interrupt is inhibited

1: An interrupt is generated whenever LBDF=1 in the USART_ISR register

Note: If LIN mode is not supported, this bit is reserved and must be kept at reset value. Please refer to Section 27.4: USART implementation on page 888.

Bit 5 **LBDL**: LIN break detection length

This bit is for selection between 11 bit or 10 bit break detection.

0: 10-bit break detection

1: 11-bit break detection

This bit can only be written when the USART is disabled (UE=0).

Note: If LIN mode is not supported, this bit is reserved and must be kept at reset value. Please refer to Section 27.4: USART implementation on page 888.

Bit 4 **ADDM7**: 7-bit Address Detection/4-bit Address Detection

This bit is for selection between 4-bit address detection or 7-bit address detection.

0: 4-bit address detection

1: 7-bit address detection (in 8-bit data mode)

This bit can only be written when the USART is disabled (UE=0)

Note: In 7-bit and 9-bit data modes, the address detection is done on 6-bit and 8-bit address (ADD[5:0] and ADD[7:0]) respectively.

Bits 3:0 Reserved, must be kept at reset value.

Note: The 3 bits (CPOL, CPHA, LBCL) should not be written while the transmitter is enabled.

27.8.3 Control register 3 (USART_CR3)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	TCBGT IE	Res.	Res.	Res.	Res.	SCARC NT2	SCARC NT1	SCARC NT0	Res.
							rw					rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DEP	DEM	DDRE	OVRDI S	ONEBI T	CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bit 24 **TCBGTIE**: Transmission complete before guard time interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated whenever TCBGT=1 in the USART_ISR register.

Note: If Smartcard mode is not supported, this bit is reserved and must be kept at reset value (see Section 27.4: USART implementation).

Bit 23 Reserved, must be kept at reset value.

Bits 22:20 Reserved, must be kept at reset value.

Bits 19:17 **SCARCNT[2:0]**: Smartcard auto-retry count

This bit-field specifies the number of retries in transmit and receive, in Smartcard mode. In transmission mode, it specifies the number of automatic retransmission retries, before generating a transmission error (FE bit set). In reception mode, it specifies the number of erroneous reception trials, before generating a reception error (RXNE and PE bits set). This bit field must be programmed only when the USART is disabled (UE=0). When the USART is enabled (UE=1), this bit field may only be written to 0x0, in order to stop retransmission.

0x0: retransmission disabled - No automatic retransmission in transmit mode.
0x1 to 0x7: number of automatic retransmission attempts (before signaling error)

Note: If Smartcard mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bit 16 Reserved, must be kept at reset value.

Bit 15 **DEP**: Driver enable polarity selection

0: DE signal is active high.
1: DE signal is active low.

This bit can only be written when the USART is disabled (UE=0).

Note: If the Driver Enable feature is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bit 14 **DEM**: Driver enable mode

This bit allows the user to activate the external transceiver control, through the DE signal.
0: DE function is disabled.
1: DE function is enabled. The DE signal is output on the RTS pin.
This bit can only be written when the USART is disabled (UE=0).

Note: If the Driver Enable feature is not supported, this bit is reserved and must be kept at reset value. [Section 27.4: USART implementation on page 888](#).

Bit 13 **DDRE**: DMA Disable on Reception Error

0: DMA is not disabled in case of reception error. The corresponding error flag is set but RXNE is kept 0 preventing from overrun. As a consequence, the DMA request is not asserted, so the erroneous data is not transferred (no DMA request), but next correct received data will be transferred (used for Smartcard mode).
1: DMA is disabled following a reception error. The corresponding error flag is set, as well as RXNE. The DMA request is masked until the error flag is cleared. This means that the software must first disable the DMA request (DMAR = 0) or clear RXNE before clearing the error flag.
This bit can only be written when the USART is disabled (UE=0).

Note: The reception errors are: parity error, framing error or noise error.

Bit 12 **OVRDIS**: Overrun Disable

This bit is used to disable the receive overrun detection.
0: Overrun Error Flag, ORE, is set when received data is not read before receiving new data.
1: Overrun functionality is disabled. If new data is received while the RXNE flag is still set the ORE flag is not set and the new received data overwrites the previous content of the USART_RDR register.
This bit can only be written when the USART is disabled (UE=0).

Note: This control bit allows checking the communication flow without reading the data.

Bit 11 **ONEBIT**: One sample bit method enable

This bit allows the user to select the sample method. When the one sample bit method is selected the noise detection flag (NF) is disabled.

- 0: Three sample bit method
- 1: One sample bit method

This bit can only be written when the USART is disabled (UE=0).

Note: ONEBIT feature applies only to data bits, It does not apply to Start bit.

Bit 10 **CTSIE**: CTS interrupt enable

- 0: Interrupt is inhibited
- 1: An interrupt is generated whenever CTSIF=1 in the USART_ISR register

Note: If the hardware flow control feature is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bit 9 **CTSE**: CTS enable

- 0: CTS hardware flow control disabled
- 1: CTS mode enabled, data is only transmitted when the CTS input is asserted (tied to 0). If the CTS input is de-asserted while data is being transmitted, then the transmission is completed before stopping. If data is written into the data register while CTS is de-asserted, the transmission is postponed until CTS is asserted.

This bit can only be written when the USART is disabled (UE=0)

Note: If the hardware flow control feature is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bit 8 **RTSE**: RTS enable

- 0: RTS hardware flow control disabled
- 1: RTS output enabled, data is only requested when there is space in the receive buffer. The transmission of data is expected to cease after the current character has been transmitted. The RTS output is asserted (pulled to 0) when data can be received.

This bit can only be written when the USART is disabled (UE=0)

Note: If the hardware flow control feature is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bit 7 **DMAT**: DMA enable transmitter

- This bit is set/reset by software
- 1: DMA mode is enabled for transmission
- 0: DMA mode is disabled for transmission

Bit 6 **DMAR**: DMA enable receiver

- This bit is set/reset by software
- 1: DMA mode is enabled for reception
- 0: DMA mode is disabled for reception

Bit 5 **SCEN**: Smartcard mode enable

This bit is used for enabling Smartcard mode.

- 0: Smartcard Mode disabled
- 1: Smartcard Mode enabled

This bit field can only be written when the USART is disabled (UE=0).

Note: If the USART does not support Smartcard mode, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bit 4 **NACK**: Smartcard NACK enable

0: NACK transmission in case of parity error is disabled

1: NACK transmission during parity error is enabled

This bit field can only be written when the USART is disabled (UE=0).

Note: If the USART does not support Smartcard mode, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bit 3 **HDSEL**: Half-duplex selection

Selection of Single-wire Half-duplex mode

0: Half duplex mode is not selected

1: Half duplex mode is selected

This bit can only be written when the USART is disabled (UE=0).

Bit 2 **IRLP**: IrDA low-power

This bit is used for selecting between normal and low-power IrDA modes

0: Normal mode

1: Low-power mode

This bit can only be written when the USART is disabled (UE=0).

Note: If IrDA mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bit 1 **IREN**: IrDA mode enable

This bit is set and cleared by software.

0: IrDA disabled

1: IrDA enabled

This bit can only be written when the USART is disabled (UE=0).

Note: If IrDA mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bit 0 **EIE**: Error interrupt enable

Error Interrupt Enable Bit is required to enable interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the USART_ISR register).

0: Interrupt is inhibited

1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USART_ISR register.

27.8.4 Baud rate register (USART_BRR)

This register can only be written when the USART is disabled (UE=0). It may be automatically updated by hardware in auto baud rate detection mode.

Address offset: 0x0C

Reset value: 0x0000 0000

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

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

Bits 15:4 **BRR[15:4]**

$\text{BRR}[15:4] = \text{USARTDIV}[15:4]$

Bits 3:0 **BRR[3:0]**

When $\text{OVER8} = 0$, $\text{BRR}[3:0] = \text{USARTDIV}[3:0]$.

When $\text{OVER8} = 1$:

$\text{BRR}[2:0] = \text{USARTDIV}[3:0]$ shifted 1 bit to the right.

$\text{BRR}[3]$ must be kept cleared.

27.8.5 Guard time and prescaler register (USART_GTPR)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
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, must be kept at reset value.

Bits 15:8 **GT[7:0]**: Guard time value

This bit-field is used to program the Guard time value in terms of number of baud clock periods.

This is used in Smartcard mode. The Transmission Complete flag is set after this guard time value.

This bit field can only be written when the USART is disabled (UE=0).

Note: If Smartcard mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bits 7:0 **PSC[7:0]**: Prescaler value

In IrDA Low-power and normal IrDA mode:

PSC[7:0] = IrDA Normal and Low-Power Baud Rate

Used for programming the prescaler for dividing the USART source clock to achieve the low-power frequency:

The source clock is divided by the value given in the register (8 significant bits):

00000000: Reserved - do not program this value

00000001: divides the source clock by 1

00000010: divides the source clock by 2

...

In Smartcard mode:

PSC[4:0]: Prescaler value

Used for programming the prescaler for dividing the USART source clock to provide the Smartcard clock.

The value given in the register (5 significant bits) is multiplied by 2 to give the division factor of the source clock frequency:

00000: Reserved - do not program this value

00001: divides the source clock by 2

00010: divides the source clock by 4

00011: divides the source clock by 6

...

This bit field can only be written when the USART is disabled (UE=0).

Note: Bits [7:5] must be kept at reset value if Smartcard mode is used.

This bit field is reserved and must be kept at reset value when the Smartcard and IrDA modes are not supported. Please refer to [Section 27.4: USART implementation on page 888](#).

27.8.6 Receiver timeout register (USART_RTOR)

Address offset: 0x14

Reset value: 0x0000 0000

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

Bits 31:24 BLEN[7:0]: Block Length

This bit-field gives the Block length in Smartcard T=1 Reception. Its value equals the number of information characters + the length of the Epilogue Field (1-LEC/2-CRC) - 1.

Examples:

BLEN = 0 -> 0 information characters + LEC

BLEN = 1 -> 0 information characters + CRC

BLEN = 255 -> 254 information characters + CRC (total 256 characters))

In Smartcard mode, the Block length counter is reset when TXE=0.

This bit-field can be used also in other modes. In this case, the Block length counter is reset when RE=0 (receiver disabled) and/or when the EOBCF bit is written to 1.

Note: This value can be programmed after the start of the block reception (using the data from the LEN character in the Prologue Field). It must be programmed only once per received block.

Bits 23:0 RTO[23:0]: Receiver timeout value

This bit-field gives the Receiver timeout value in terms of number of bit duration.

In standard mode, the RTOF flag is set if, after the last received character, no new start bit is detected for more than the RTO value.

In Smartcard mode, this value is used to implement the CWT and BWT. See Smartcard section for more details.

In this case, the timeout measurement is done starting from the Start Bit of the last received character.

Note: This value must only be programmed once per received character.

Note: *RTOR can be written on the fly. If the new value is lower than or equal to the counter, the RTOF flag is set.*

This register is reserved and forced by hardware to “0x00000000” when the Receiver timeout feature is not supported. Please refer to [Section 27.4: USART implementation on page 888](#).

27.8.7 Request register (USART_RQR)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TXFRQ	RXFRQ	MMRQ	SBKRQ	ABRRQ										
											w	w	w	w	w

Bits 31:5 Reserved, must be kept at reset value.

Bit 4 TXFRQ: Transmit data flush request

Writing 1 to this bit sets the TXE flag.

This allows to discard the transmit data. This bit must be used only in Smartcard mode, when data has not been sent due to errors (NACK) and the FE flag is active in the USART_ISR register.

If the USART does not support Smartcard mode, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bit 3 RXFRQ: Receive data flush request

Writing 1 to this bit clears the RXNE flag.

This allows to discard the received data without reading it, and avoid an overrun condition.

Bit 2 MMRQ: Mute mode request

Writing 1 to this bit puts the USART in mute mode and sets the RWU flag.

Bit 1 SBKRQ: Send break request

Writing 1 to this bit sets the SBKF flag and request to send a BREAK on the line, as soon as the transmit machine is available.

Note: In the case the application needs to send the break character following all previously inserted data, including the ones not yet transmitted, the software should wait for the TXE flag assertion before setting the SBKRQ bit.

Bit 0 ABRRQ: Auto baud rate request

Writing 1 to this bit resets the ABRF flag in the USART_ISR and request an automatic baud rate measurement on the next received data frame.

Note: If the USART does not support the auto baud rate feature, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

27.8.8 Interrupt and status register (USART_ISR)

Address offset: 0x1C

Reset value: 0x0200 00C0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	TCBGT	Res.	Res.	Res.	TEACK	Res.	RWU	SBKF	CMF	BUSY
						r				r		r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABRF	ABRE	Res.	EOBF	RTOF	CTS	CTSIF	LBDF	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
r	r		r	r	r	r	r	r	r	r	r	r	r	r	r

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

Bit 25 **TCBGT**: Transmission complete before guard time completion.

This bit is used in Smartcard mode. It is set by hardware if the transmission of a frame containing data has completed successfully (no NACK received from the card) and before the guard time has elapsed (contrary to the TC flag which is set when the guard time has elapsed).

An interrupt is generated if TCBGTIE=1 in USART_CR3 register. It is cleared by software, by writing 1 to TCBGTCF in USART_ICR or by writing to the USART_TDR register.

0: Transmission not complete or transmission completed with error (i.e. NACK received from the card)

1: Transmission complete (before Guard time has elapsed and no NACK received from the smartcard).

Note: If the USART does not support the Smartcard mode, this bit is reserved and must be kept at reset value. If the USART supports the Smartcard mode and the Smartcard mode is enabled, the TCBGT reset value is 1.

Bits 24:22 Reserved, must be kept at reset value.

Bit 21 **TEACK**: Transmit enable acknowledge flag

This bit is set/reset by hardware, when the Transmit Enable value is taken into account by the USART.

It can be used when an idle frame request is generated by writing TE=0, followed by TE=1 in the USART_CR1 register, in order to respect the TE=0 minimum period.

Bit 20 Reserved, must be kept at reset value.

Bit 19 **RWU**: Receiver wakeup from Mute mode

This bit indicates if the USART is in mute mode. It is cleared/set by hardware when a wakeup/mute sequence is recognized. The mute mode control sequence (address or IDLE) is selected by the WAKE bit in the USART_CR1 register.

When wakeup on IDLE mode is selected, this bit can only be set by software, writing 1 to the MMRQ bit in the USART_RQR register.

0: Receiver in active mode

1: Receiver in mute mode

Bit 18 **SBKF**: Send break flag

This bit indicates that a send break character was requested. It is set by software, by writing 1 to the SBKRQ bit in the USART_RQR register. It is automatically reset by hardware during the stop bit of break transmission.

0: No break character is transmitted

1: Break character will be transmitted

Bit 17 **CMF**: Character match flag

This bit is set by hardware, when the character defined by ADD[7:0] is received. It is cleared by software, writing 1 to the CMCF in the USART_ICR register.

An interrupt is generated if CMIE=1 in the USART_CR1 register.

0: No Character match detected

1: Character Match detected

Bit 16 **BUSY**: Busy flag

This bit is set and reset by hardware. It is active when a communication is ongoing on the RX line (successful start bit detected). It is reset at the end of the reception (successful or not).

0: USART is idle (no reception)

1: Reception on going

Bit 15 **ABRF**: Auto baud rate flag

This bit is set by hardware when the automatic baud rate has been set (RXNE will also be set, generating an interrupt if RXNEIE = 1) or when the auto baud rate operation was completed without success (ABRE=1) (ABRE, RXNE and FE are also set in this case). It is cleared by software, in order to request a new auto baud rate detection, by writing 1 to the ABRRQ in the USART_RQR register.

Note: If the USART does not support the auto baud rate feature, this bit is reserved and kept at reset value.

Bit 14 **ABRE**: Auto baud rate error

This bit is set by hardware if the baud rate measurement failed (baud rate out of range or character comparison failed)

It is cleared by software, by writing 1 to the ABRRQ bit in the USART_CR3 register.

Note: If the USART does not support the auto baud rate feature, this bit is reserved and kept at reset value.

Bit 13 Reserved, must be kept at reset value.

Bit 12 **EOBF**: End of block flag

This bit is set by hardware when a complete block has been received (for example T=1 Smartcard mode). The detection is done when the number of received bytes (from the start of the block, including the prologue) is equal or greater than BLEN + 4.

An interrupt is generated if the EOBIIE=1 in the USART_CR2 register.

It is cleared by software, writing 1 to the EOBCF in the USART_ICR register.

0: End of Block not reached

1: End of Block (number of characters) reached

Note: If Smartcard mode is not supported, this bit is reserved and kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bit 11 **RTOF**: Receiver timeout

This bit is set by hardware when the timeout value, programmed in the RTOR register has lapsed, without any communication. It is cleared by software, writing 1 to the RTOCF bit in the USART_ICR register.

An interrupt is generated if RTOIE=1 in the USART_CR1 register.

In Smartcard mode, the timeout corresponds to the CWT or BWT timings.

0: Timeout value not reached

1: Timeout value reached without any data reception

Note: If a time equal to the value programmed in RTOR register separates 2 characters, RTOF is not set. If this time exceeds this value + 2 sample times (2/16 or 2/8, depending on the oversampling method), RTOF flag is set.

The counter counts even if RE = 0 but RTOF is set only when RE = 1. If the timeout has already elapsed when RE is set, then RTOF will be set.

If the USART does not support the Receiver timeout feature, this bit is reserved and kept at reset value.

Bit 10 **CTS**: CTS flag

This bit is set/reset by hardware. It is an inverted copy of the status of the CTS input pin.

0: CTS line set

1: CTS line reset

Note: If the hardware flow control feature is not supported, this bit is reserved and kept at reset value.

Bit 9 **CTSIF**: CTS interrupt flag

This bit is set by hardware when the CTS input toggles, if the CTSE bit is set. It is cleared by software, by writing 1 to the CTSCF bit in the USART_ICR register.

An interrupt is generated if CTSIE=1 in the USART_CR3 register.

0: No change occurred on the CTS status line

1: A change occurred on the CTS status line

Note: If the hardware flow control feature is not supported, this bit is reserved and kept at reset value.

Bit 8 **LBDF**: LIN break detection flag

This bit is set by hardware when the LIN break is detected. It is cleared by software, by writing 1 to the LBDCF in the USART_ICR.

An interrupt is generated if LBDIE = 1 in the USART_CR2 register.

0: LIN Break not detected

1: LIN break detected

*Note: If the USART does not support LIN mode, this bit is reserved and kept at reset value.
Please refer to [Section 27.4: USART implementation on page 888](#).*

Bit 7 **TXE**: Transmit data register empty

This bit is set by hardware when the content of the USART_TDR register has been transferred into the shift register. It is cleared by a write to the USART_TDR register.

The TXE flag can also be cleared by writing 1 to the TXFRQ in the USART_RQR register, in order to discard the data (only in Smartcard T=0 mode, in case of transmission failure).

An interrupt is generated if the TXIE bit =1 in the USART_CR1 register.

0: data is not transferred to the shift register

1: data is transferred to the shift register)

Note: This bit is used during single buffer transmission.

Bit 6 **TC**: Transmission complete

This bit is set by hardware if the transmission of a frame containing data is complete and if TXE is set. An interrupt is generated if TCIE=1 in the USART_CR1 register. It is cleared by software, writing 1 to the TCCF in the USART_ICR register or by a write to the USART_TDR register.

An interrupt is generated if TCIE=1 in the USART_CR1 register.

0: Transmission is not complete

1: Transmission is complete

Note: If TE bit is reset and no transmission is on going, the TC bit will be set immediately.

Bit 5 **RXNE**: Read data register not empty

This bit is set by hardware when the content of the RDR shift register has been transferred to the USART_RDR register. It is cleared by a read to the USART_RDR register. The RXNE flag can also be cleared by writing 1 to the RXFRQ in the USART_RQR register.

An interrupt is generated if RXNEIE=1 in the USART_CR1 register.

0: data is not received

1: Received data is ready to be read.

Bit 4 **IDLE**: Idle line detected

This bit is set by hardware when an Idle Line is detected. An interrupt is generated if IDLEIE=1 in the USART_CR1 register. It is cleared by software, writing 1 to the IDLECF in the USART_ICR register.

0: No Idle line is detected

1: Idle line is detected

Note: The IDLE bit will not be set again until the RXNE bit has been set (i.e. a new idle line occurs).

If mute mode is enabled (MME=1), IDLE is set if the USART is not mute (RWU=0), whatever the mute mode selected by the WAKE bit. If RWU=1, IDLE is not set.

Bit 3 **ORE**: Overrun error

This bit is set by hardware when the data currently being received in the shift register is ready to be transferred into the RDR register while RXNE=1. It is cleared by a software, writing 1 to the ORECF, in the USART_ICR register.

An interrupt is generated if RXNEIE=1 or EIE = 1 in the USART_CR1 register.

0: No overrun error

1: Overrun error is detected

Note: When this bit is set, the RDR register content is not lost but the shift register is overwritten. An interrupt is generated if the ORE flag is set during multibuffer communication if the EIE bit is set.

This bit is permanently forced to 0 (no overrun detection) when the OVRDIS bit is set in the USART_CR3 register.

Bit 2 **NF**: START bit Noise detection flag

This bit is set by hardware when noise is detected on a received frame. It is cleared by software, writing 1 to the NFCF bit in the USART_ICR register.

0: No noise is detected

1: Noise is detected

Note: This bit does not generate an interrupt as it appears at the same time as the RXNE bit which itself generates an interrupt. An interrupt is generated when the NF flag is set during multibuffer communication if the EIE bit is set.

Note: When the line is noise-free, the NF flag can be disabled by programming the ONEBIT bit to 1 to increase the USART tolerance to deviations (Refer to [Section 27.5.5: Tolerance of the USART receiver to clock deviation on page 904](#)).

Bit 1 **FE**: Framing error

This bit is set by hardware when a de-synchronization, excessive noise or a break character is detected. It is cleared by software, writing 1 to the FECF bit in the USART_ICR register.

In Smartcard mode, in transmission, this bit is set when the maximum number of transmit attempts is reached without success (the card NACKs the data frame).

An interrupt is generated if EIE = 1 in the USART_CR1 register.

0: No Framing error is detected

1: Framing error or break character is detected

Bit 0 **PE**: Parity error

This bit is set by hardware when a parity error occurs in receiver mode. It is cleared by software, writing 1 to the PECF in the USART_ICR register.

An interrupt is generated if PEIE = 1 in the USART_CR1 register.

0: No parity error

1: Parity error

27.8.9 Interrupt flag clear register (USART_ICR)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CMCF	Res.
														rc_w1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	EOBCF	RTOCF	Res.	CTSCF	LBDCF	TCBGT CF	TCCF	Res.	IDLECF	ORECF	NCF	FECF	PECF
			rc_w1	rc_w1		rc_w1	rc_w1	rc_w1	rc_w1		rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:20 Reserved, must be kept at reset value.

Bits 19:18 Reserved, must be kept at reset value.

Bit 17 **CMCF**: Character match clear flag

Writing 1 to this bit clears the CMF flag in the USART_ISR register.

Bits 16:13 Reserved, must be kept at reset value.

Bit 12 **EOBCF**: End of block clear flag

Writing 1 to this bit clears the EOBF flag in the USART_ISR register.

Note: If the USART does not support Smartcard mode, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bit 11 **RTOCF**: Receiver timeout clear flag

Writing 1 to this bit clears the RTOF flag in the USART_ISR register.

Note: If the USART does not support the Receiver timeout feature, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bit 10 Reserved, must be kept at reset value.

Bit 9 **CTSCF**: CTS clear flag

Writing 1 to this bit clears the CTSIF flag in the USART_ISR register.

Note: If the hardware flow control feature is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bit 8 **LBDCF**: LIN break detection clear flag

Writing 1 to this bit clears the LBDF flag in the USART_ISR register.

Note: If LIN mode is not supported, this bit is reserved and must be kept at reset value. Please refer to [Section 27.4: USART implementation on page 888](#).

Bit 7 **TCBGT**: Transmission completed before guard time clear flag

Writing 1 to this bit clears the TCBGT flag in the USART_ISR register.

Note: If the USART does not support SmartCard mode, this bit is reserved and forced by hardware to 0. Please refer to [Section 27.4: USART implementation on page 888](#).

Bit 6 **TCCF**: Transmission complete clear flag

Writing 1 to this bit clears the TC flag in the USART_ISR register.

Bit 5 Reserved, must be kept at reset value.

Bit 4 **IDLECF**: Idle line detected clear flag

Writing 1 to this bit clears the IDLE flag in the USART_ISR register.

Bit 3 **ORECF**: Overrun error clear flag

Writing 1 to this bit clears the ORE flag in the USART_ISR register.

Bit 2 **NCF**: Noise detected clear flag

Writing 1 to this bit clears the NF flag in the USART_ISR register.

Bit 1 **FECF**: Framing error clear flag

Writing 1 to this bit clears the FE flag in the USART_ISR register.

Bit 0 **PECF**: Parity error clear flag

Writing 1 to this bit clears the PE flag in the USART_ISR register.

27.8.10 Receive data register (USART_RDR)

Address offset: 0x24

Reset value: 0x0000 0000

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

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

Bits 8:0 **RDR[8:0]**: Receive data value

Contains the received data character.

The RDR register provides the parallel interface between the input shift register and the internal bus (see [Figure 317](#)).

When receiving with the parity enabled, the value read in the MSB bit is the received parity bit.

27.8.11 Transmit data register (USART_TDR)

Address offset: 0x28

Reset value: 0x0000 0000

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

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

Bits 8:0 **TDR[8:0]**: Transmit data value

Contains the data character to be transmitted.

The TDR register provides the parallel interface between the internal bus and the output shift register (see *Figure 317*).

When transmitting with the parity enabled (PCE bit set to 1 in the USART_CR1 register), the value written in the MSB (bit 7 or bit 8 depending on the data length) has no effect because it is replaced by the parity.

Note: This register must be written only when TXE=1.

27.8.12 USART register map

The table below gives the USART register map and reset values.

Table 169. 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_CR1	Res.	Res.	Res.	M1	EOBIE	RTOIE	DEAT4	DEAT3	DEAT2	DEAT1	DEAT0	DEAT4	MSBFIRST	DATAINV	TXINV	DEDT0	OVER8	CMIIE	MME	0	WAKE	PCE	PS	LBCL	TXEIE	SCEN	LBDL	RXNEIE	5	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		
0x04	USART_CR2	ADD[7:4]				ADD[3:0]				RTEN	ABRMOD1	ABRMOD0	ABRMOD0	ABREN	MSBFIRST	DATAINV	TXINV	DEDT3	DATAINV	CMIE	MME	0	0	0	0	0	0	0	0	0	0	0	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x08	USART_CR3	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SCARCNT2:0	DDRE	STOP[1:0]	RXINV	SWAP	LINEN	0	0	0	0	0	0	0	0	0	0	0	0	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x0C	USART_BRR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x10	USART_GTPR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x14	USART_RTOR	BLEN[7:0]				GT[7:0]															PSC[7:0]												
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x18	USART_RQR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Table 169. USART register map and reset values (continued)

Refer to [Section 1.5 on page 55](#) for the register boundary addresses.

28 Serial peripheral interface / inter-IC sound (SPI/I2S)

28.1 Introduction

The SPI/I²S interface can be used to communicate with external devices using the SPI protocol or the I²S audio protocol. SPI or I²S mode is selectable by software. SPI Motorola mode is selected by default after a device reset.

The serial peripheral interface (SPI) protocol supports half-duplex, full-duplex and simplex synchronous, serial communication with external devices. The interface can be configured as master and in this case it provides the communication clock (SCK) to the external slave device. The interface is also capable of operating in multimaster configuration.

The Inter-IC sound (I²S) protocol is also a synchronous serial communication interface. It can operate in slave or master mode with half-duplex communication. Full-duplex operations are possible by combining two I²S blocks. It can address four different audio standards including the Philips I²S standard, the MSB- and LSB-justified standards and the PCM standard.

28.2 SPI main features

- Master or slave operation
- Full-duplex synchronous transfers on three lines
- Half-duplex synchronous transfer on two lines (with bidirectional data line)
- Simplex synchronous transfers on two lines (with unidirectional data line)
- 4-bit to 16-bit data size selection
- Multimaster mode capability
- 8 master mode baud rate prescalers up to $f_{PCLK}/2$
- Slave mode frequency up to $f_{PCLK}/2$.
- NSS management by hardware or software for both master and slave: dynamic change of master/slave operations
- Programmable clock polarity and phase
- Programmable data order with MSB-first or LSB-first shifting
- Dedicated transmission and reception flags with interrupt capability
- SPI bus busy status flag
- SPI Motorola support
- Hardware CRC feature for reliable communication:
 - CRC value can be transmitted as last byte in Tx mode
 - Automatic CRC error checking for last received byte
- Master mode fault, overrun flags with interrupt capability
- CRC Error flag
- Two 32-bit embedded Rx and Tx FIFOs with DMA capability
- SPI TI mode support

28.3 I2S main features

- Half-duplex communication (only transmitter or receiver)
- Master or slave operations
- 8-bit programmable linear prescaler to reach accurate audio sample frequencies (from 8 kHz to 192 kHz)
- Data format may be 16-bit, 24-bit or 32-bit
- Packet frame is fixed to 16-bit (16-bit data frame) or 32-bit (16-bit, 24-bit, 32-bit data frame) by audio channel
- Programmable clock polarity (steady state)
- Underrun flag in slave transmission mode, overrun flag in reception mode (master and slave) and Frame Error Flag in reception and transmitter mode (slave only)
- 16-bit register for transmission and reception with one data register for both channel sides
- Supported I²S protocols:
 - I²S Philips standard
 - MSB-justified standard (left-justified)
 - LSB-justified standard (right-justified)
 - PCM standard (with short and long frame synchronization on 16-bit channel frame or 16-bit data frame extended to 32-bit channel frame)
- Data direction is always MSB first
- DMA capability for transmission and reception (16-bit wide)
- Master clock can be output to drive an external audio component. Ratio is fixed at $256 \times F_S$ (where F_S is the audio sampling frequency)

28.4 SPI/I2S implementation

This manual describes the SPI/I2S implementation in STM32F72xxx and STM32F73xxx devices.

Table 170. STM32F72xxx and STM32F73xxx SPI implementation

SPI Features ⁽¹⁾	SPI1	SPI2	SPI3	SPI4	SPI5
Hardware CRC calculation	X	X	X	X	X
Rx/Tx FIFO	X	X	X	X	X
NSS pulse mode	X	X	X	X	X
I ² S mode	X	X	X	-	-
TI mode	X	X	X	X	X

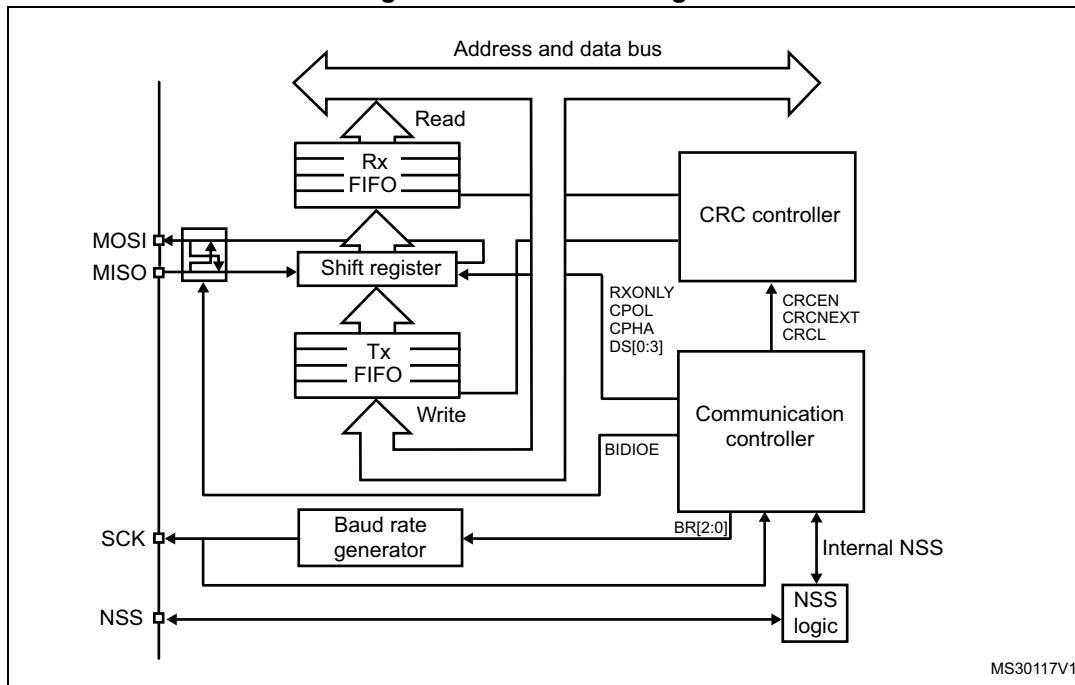
1. X = supported.

28.5 SPI functional description

28.5.1 General description

The SPI allows synchronous, serial communication between the MCU and external devices. Application software can manage the communication by polling the status flag or using dedicated SPI interrupt. The main elements of SPI and their interactions are shown in the following block diagram [Figure 342](#).

Figure 342. SPI block diagram



MS30117V1

Four I/O pins are dedicated to SPI communication with external devices.

- **MISO:** Master In / Slave Out data. In the general case, this pin is used to transmit data in slave mode and receive data in master mode.
- **MOSI:** Master Out / Slave In data. In the general case, this pin is used to transmit data in master mode and receive data in slave mode.
- **SCK:** Serial Clock output pin for SPI masters and input pin for SPI slaves.
- **NSS:** Slave select pin. Depending on the SPI and NSS settings, this pin can be used to either:
 - select an individual slave device for communication
 - synchronize the data frame or
 - detect a conflict between multiple masters

See [Section 28.5.5: Slave select \(NSS\) pin management](#) for details.

The SPI bus allows the communication between one master device and one or more slave devices. The bus consists of at least two wires - one for the clock signal and the other for synchronous data transfer. Other signals can be added depending on the data exchange between SPI nodes and their slave select signal management.

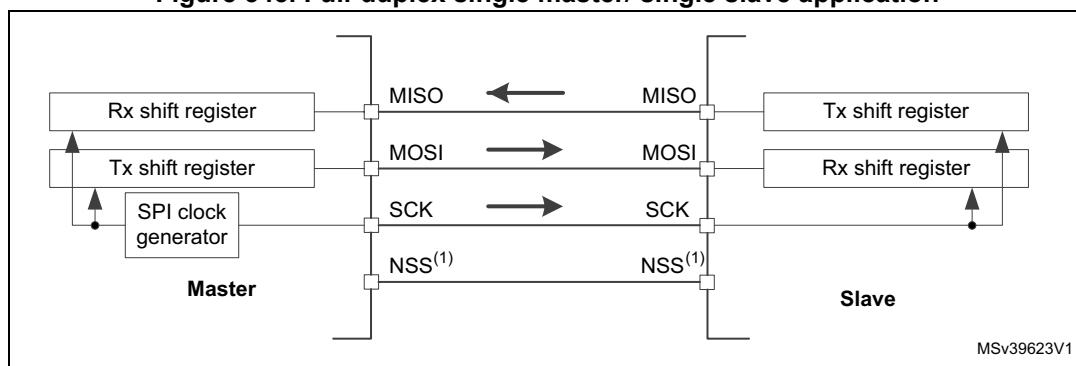
28.5.2 Communications between one master and one slave

The SPI allows the MCU to communicate using different configurations, depending on the device targeted and the application requirements. These configurations use 2 or 3 wires (with software NSS management) or 3 or 4 wires (with hardware NSS management). Communication is always initiated by the master.

Full-duplex communication

By default, the SPI is configured for full-duplex communication. In this configuration, the shift registers of the master and slave are linked using two unidirectional lines between the MOSI and the MISO pins. During SPI communication, data is shifted synchronously on the SCK clock edges provided by the master. The master transmits the data to be sent to the slave via the MOSI line and receives data from the slave via the MISO line. When the data frame transfer is complete (all the bits are shifted) the information between the master and slave is exchanged.

Figure 343. Full-duplex single master/ single slave application

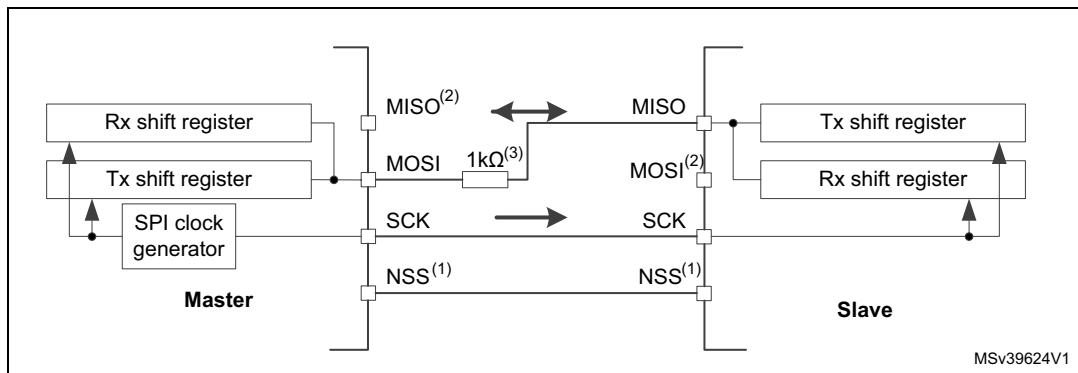


1. The NSS pins can be used to provide a hardware control flow between master and slave. Optionally, the pins can be left unused by the peripheral. Then the flow has to be handled internally for both master and slave. For more details see [Section 28.5.5: Slave select \(NSS\) pin management](#).

Half-duplex communication

The SPI can communicate in half-duplex mode by setting the BIDIMODE bit in the SPIx_CR1 register. In this configuration, one single cross connection line is used to link the shift registers of the master and slave together. During this communication, the data is synchronously shifted between the shift registers on the SCK clock edge in the transfer direction selected reciprocally by both master and slave with the BDIOE bit in their SPIx_CR1 registers. In this configuration, the master's MISO pin and the slave's MOSI pin are free for other application uses and act as GPIOs.

Figure 344. Half-duplex single master/ single slave application



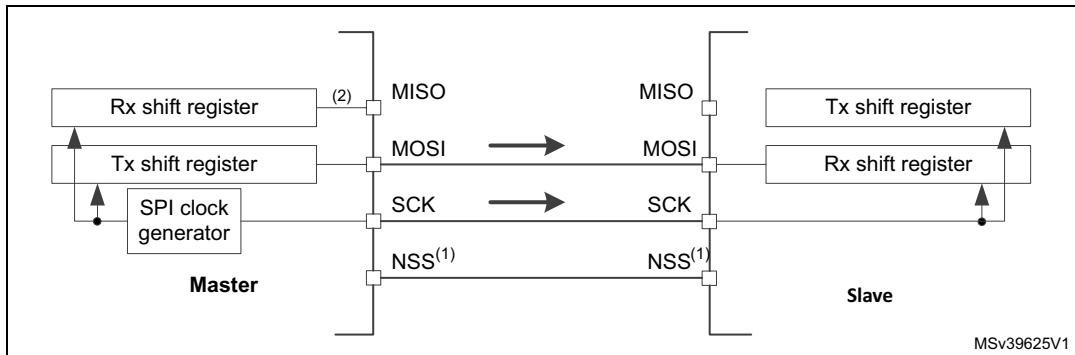
1. The NSS pins can be used to provide a hardware control flow between master and slave. Optionally, the pins can be left unused by the peripheral. Then the flow has to be handled internally for both master and slave. For more details see [Section 28.5.5: Slave select \(NSS\) pin management](#).
2. In this configuration, the master's MISO pin and the slave's MOSI pin can be used as GPIOs.
3. A critical situation can happen when communication direction is changed not synchronously between two nodes working at bidirectional mode and new transmitter accesses the common data line while former transmitter still keeps an opposite value on the line (the value depends on SPI configuration and communication data). Both nodes then fight while providing opposite output levels on the common line temporary till next node changes its direction settings correspondingly, too. It is suggested to insert a serial resistance between MISO and MOSI pins at this mode to protect the outputs and limit the current blowing between them at this situation.

Simplex communications

The SPI can communicate in simplex mode by setting the SPI in transmit-only or in receive-only using the RXONLY bit in the SPIx_CR2 register. In this configuration, only one line is used for the transfer between the shift registers of the master and slave. The remaining MISO and MOSI pins pair is not used for communication and can be used as standard GPIOs.

- **Transmit-only mode (RXONLY=0):** The configuration settings are the same as for full-duplex. The application has to ignore the information captured on the unused input pin. This pin can be used as a standard GPIO.
- **Receive-only mode (RXONLY=1):** The application can disable the SPI output function by setting the RXONLY bit. In slave configuration, the MISO output is disabled and the pin can be used as a GPIO. The slave continues to receive data from the MOSI pin while its slave select signal is active (see [28.5.5: Slave select \(NSS\) pin management](#)). Received data events appear depending on the data buffer configuration. In the master configuration, the MOSI output is disabled and the pin can be used as a GPIO. The clock signal is generated continuously as long as the SPI is enabled. The only way to stop the clock is to clear the RXONLY bit or the SPE bit and wait until the incoming pattern from the MISO pin is finished and fills the data buffer structure, depending on its configuration.

Figure 345. Simplex single master/single slave application (master in transmit-only/slave in receive-only mode)



1. The NSS pins can be used to provide a hardware control flow between master and slave. Optionally, the pins can be left unused by the peripheral. Then the flow has to be handled internally for both master and slave. For more details see [Section 28.5.5: Slave select \(NSS\) pin management](#).
2. An accidental input information is captured at the input of transmitter Rx shift register. All the events associated with the transmitter receive flow must be ignored in standard transmit only mode (e.g. OVF flag).
3. In this configuration, both the MISO pins can be used as GPIOs.

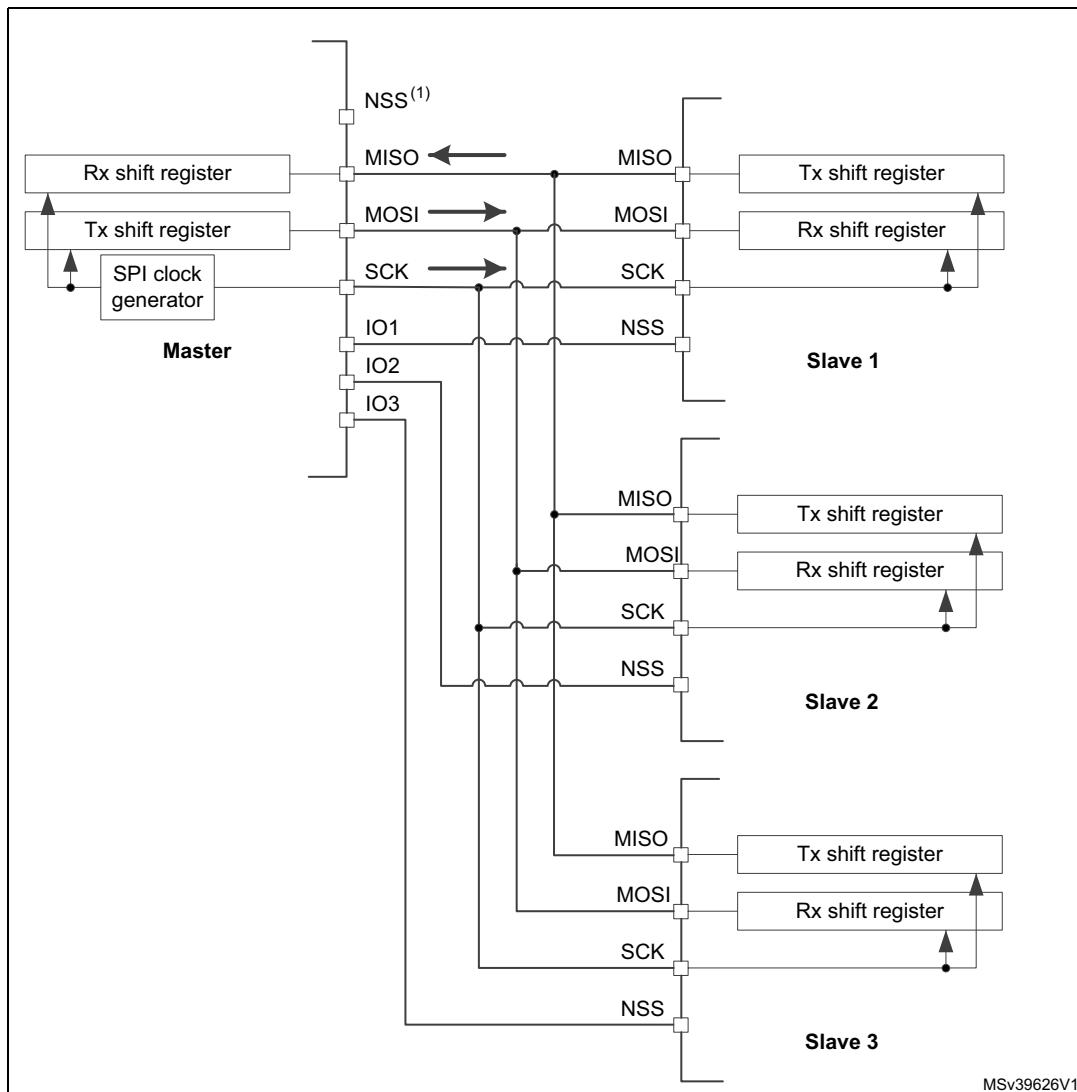
Note:

Any simplex communication can be alternatively replaced by a variant of the half-duplex communication with a constant setting of the transaction direction (bidirectional mode is enabled while BDIO bit is not changed).

28.5.3 Standard multi-slave communication

In a configuration with two or more independent slaves, the master uses GPIO pins to manage the chip select lines for each slave (see [Figure 346](#)). The master must select one of the slaves individually by pulling low the GPIO connected to the slave NSS input. When this is done, a standard master and dedicated slave communication is established.

Figure 346. Master and three independent slaves



1. NSS pin is not used on master side at this configuration. It has to be managed internally ($SSM=1$, $SSI=1$) to prevent any MODF error.
2. As MISO pins of the slaves are connected together, all slaves must have the GPIO configuration of their MISO pin set as alternate function open-drain (see [Section 7.3.7: I/O alternate function input/output](#)).

28.5.4 Multi-master communication

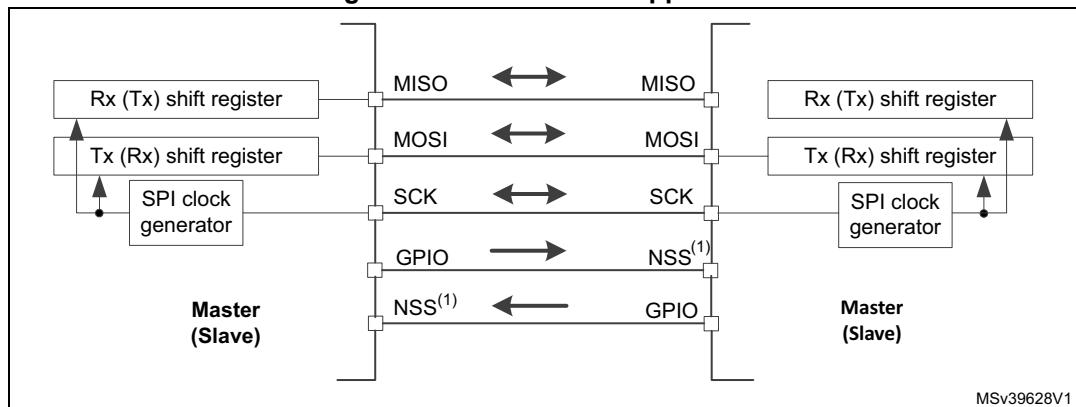
Unless SPI bus is not designed for a multi-master capability primarily, the user can use build in feature which detects a potential conflict between two nodes trying to master the bus at the same time. For this detection, NSS pin is used configured at hardware input mode.

The connection of more than two SPI nodes working at this mode is impossible as only one node can apply its output on a common data line at time.

When nodes are non active, both stay at slave mode by default. Once one node wants to overtake control on the bus, it switches itself into master mode and applies active level on the slave select input of the other node via dedicated GPIO pin. After the session is completed, the active slave select signal is released and the node mastering the bus temporary returns back to passive slave mode waiting for next session start.

If potentially both nodes raised their mastering request at the same time a bus conflict event appears (see mode fault MODF event). Then the user can apply some simple arbitration process (e.g. to postpone next attempt by predefined different time-outs applied at both nodes).

Figure 347. Multi-master application



1. The NSS pin is configured at hardware input mode at both nodes. Its active level enables the MISO line output control as the passive node is configured as a slave.

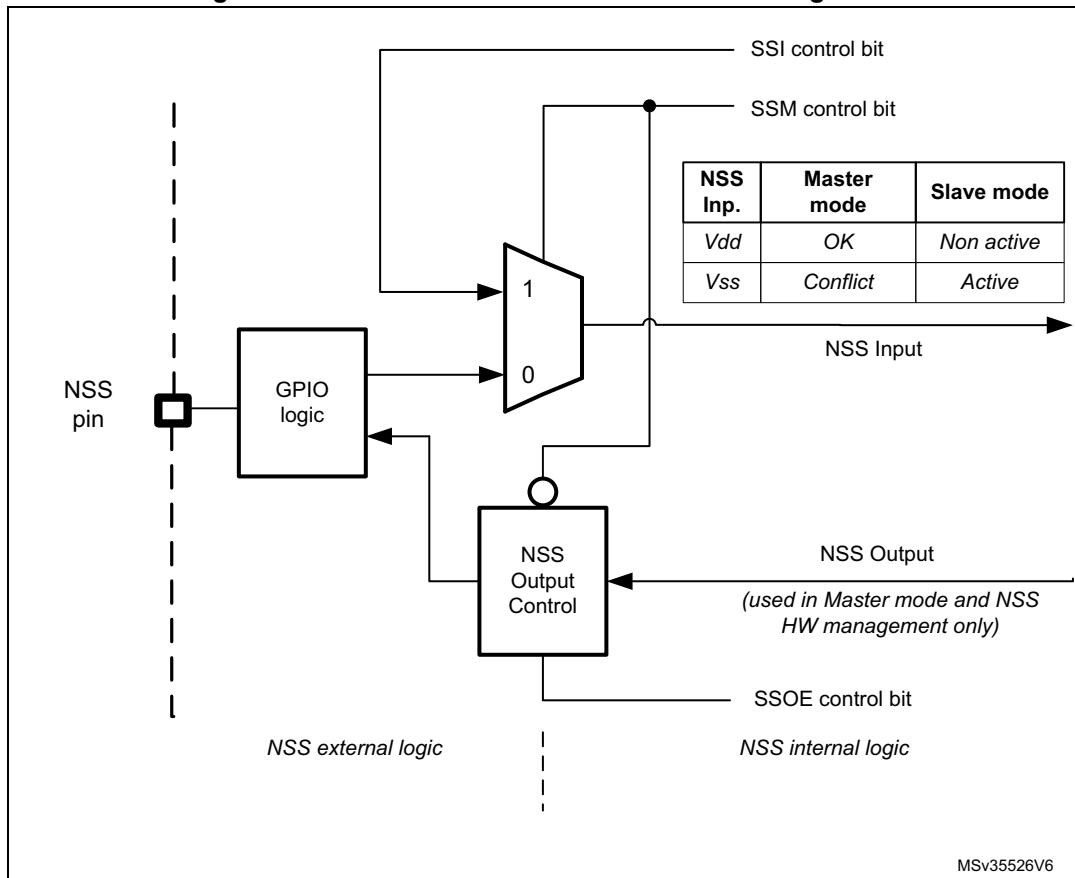
28.5.5 Slave select (NSS) pin management

In slave mode, the NSS works as a standard “chip select” input and lets the slave communicate with the master. In master mode, NSS can be used either as output or input. As an input it can prevent multimaster bus collision, and as an output it can drive a slave select signal of a single slave.

Hardware or software slave select management can be set using the SSM bit in the SPIx_CR1 register:

- **Software NSS management (SSM = 1):** in this configuration, slave select information is driven internally by the SSI bit value in register SPIx_CR1. The external NSS pin is free for other application uses.
- **Hardware NSS management (SSM = 0):** in this case, there are two possible configurations. The configuration used depends on the NSS output configuration (SSOE bit in register SPIx_CR1).
 - **NSS output enable (SSM=0,SSOE = 1):** this configuration is only used when the MCU is set as master. The NSS pin is managed by the hardware. The NSS signal is driven low as soon as the SPI is enabled in master mode (SPE=1), and is kept low until the SPI is disabled (SPE =0). A pulse can be generated between continuous communications if NSS pulse mode is activated (NSSP=1). The SPI cannot work in multimaster configuration with this NSS setting.
 - **NSS output disable (SSM=0, SSOE = 0):** if the microcontroller is acting as the master on the bus, this configuration allows multimaster capability. If the NSS pin is pulled low in this mode, the SPI enters master mode fault state and the device is automatically reconfigured in slave mode. In slave mode, the NSS pin works as a standard “chip select” input and the slave is selected while NSS line is at low level.

Figure 348. Hardware/software slave select management



28.5.6 Communication formats

During SPI communication, receive and transmit operations are performed simultaneously. The serial clock (SCK) synchronizes the shifting and sampling of the information on the data lines. The communication format depends on the clock phase, the clock polarity and the data frame format. To be able to communicate together, the master and slaves devices must follow the same communication format.

Clock phase and polarity controls

Four possible timing relationships may be chosen by software, using the CPOL and CPHA bits in the SPIx_CR1 register. The CPOL (clock polarity) bit controls the idle state value of the clock when no data is being transferred. This bit affects both master and slave modes. If CPOL is reset, the SCK pin has a low-level idle state. If CPOL is set, the SCK pin has a high-level idle state.

If the CPHA bit is set, the second edge on the SCK pin captures the first data bit transacted (falling edge if the CPOL bit is reset, rising edge if the CPOL bit is set). Data are latched on each occurrence of this clock transition type. If the CPHA bit is reset, the first edge on the SCK pin captures the first data bit transacted (falling edge if the CPOL bit is set, rising edge if the CPOL bit is reset). Data are latched on each occurrence of this clock transition type.

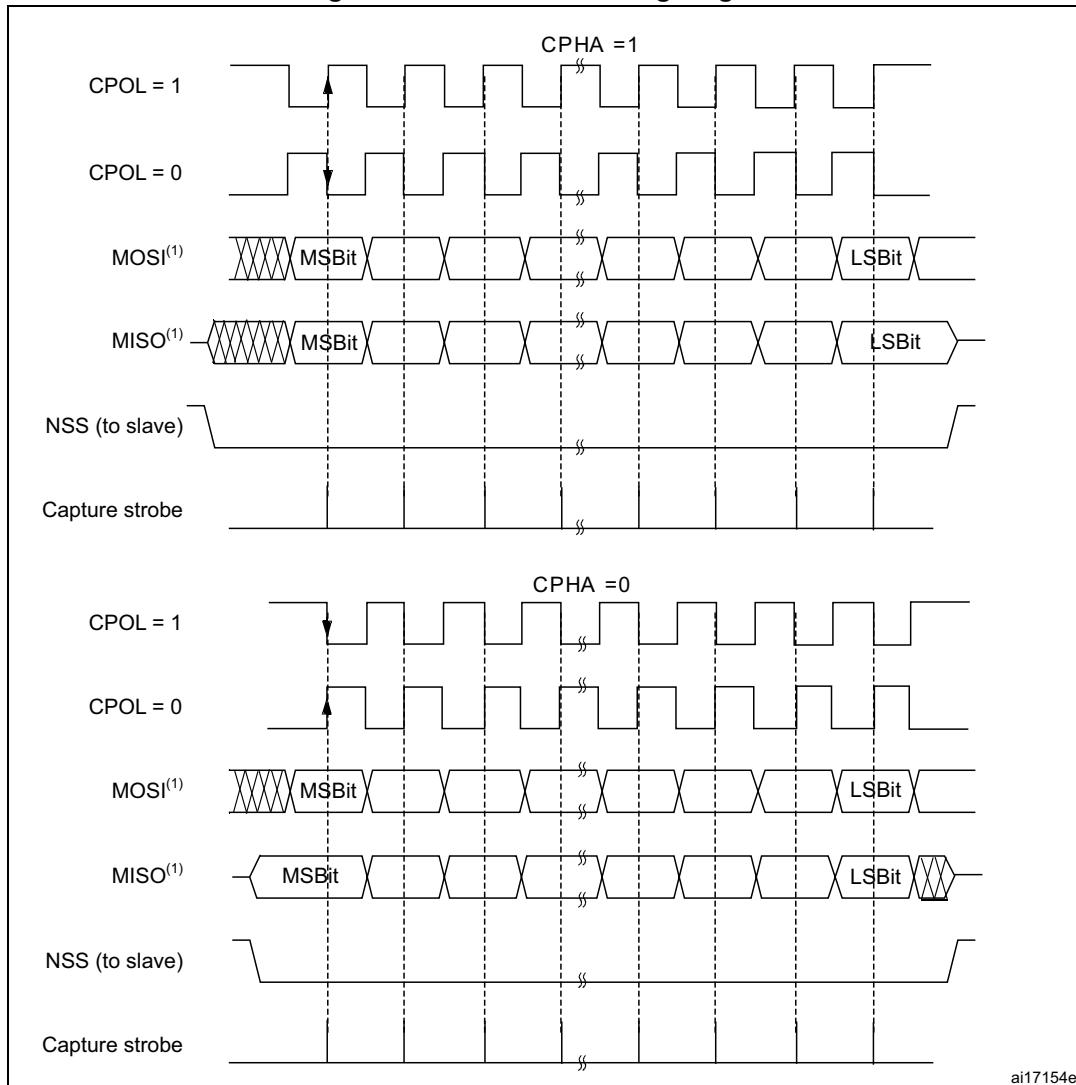
The combination of CPOL (clock polarity) and CPHA (clock phase) bits selects the data capture clock edge.

[Figure 349](#), shows an SPI full-duplex transfer with the four combinations of the CPHA and CPOL bits.

Note: Prior to changing the CPOL/CPHA bits the SPI must be disabled by resetting the SPE bit.

The idle state of SCK must correspond to the polarity selected in the SPIx_CR1 register (by pulling up SCK if CPOL=1 or pulling down SCK if CPOL=0).

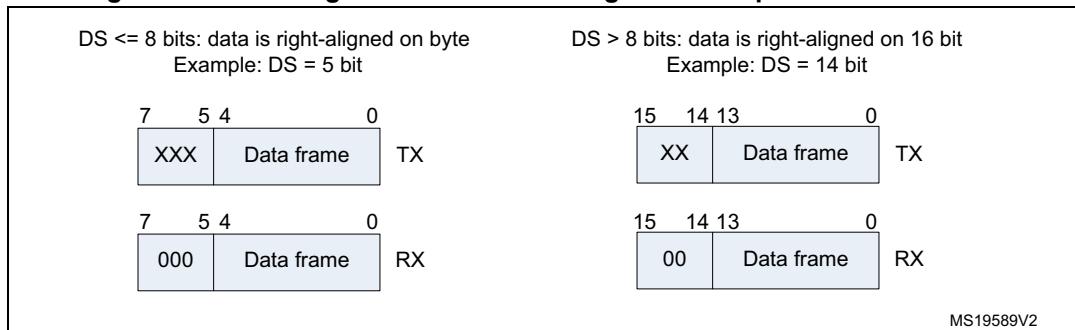
Figure 349. Data clock timing diagram



1. The order of data bits depends on LSBFIRST bit setting.

Data frame format

The SPI shift register can be set up to shift out MSB-first or LSB-first, depending on the value of the LSBFIRST bit. The data frame size is chosen by using the DS bits. It can be set from 4-bit up to 16-bit length and the setting applies for both transmission and reception. Whatever the selected data frame size, read access to the FIFO must be aligned with the FRXTH level. When the SPIx_DR register is accessed, data frames are always right-aligned into either a byte (if the data fits into a byte) or a half-word (see [Figure 350](#)). During communication, only bits within the data frame are clocked and transferred.

Figure 350. Data alignment when data length is not equal to 8-bit or 16-bit

Note: The minimum data length is 4 bits. If a data length of less than 4 bits is selected, it is forced to an 8-bit data frame size.

28.5.7 Configuration of SPI

The configuration procedure is almost the same for master and slave. For specific mode setups, follow the dedicated sections. When a standard communication is to be initialized, perform these steps:

1. Write proper GPIO registers: Configure GPIO for MOSI, MISO and SCK pins.
2. Write to the SPI_CR1 register:
 - a) Configure the serial clock baud rate using the BR[2:0] bits (Note: 4).
 - b) Configure the CPOL and CPHA bits combination to define one of the four relationships between the data transfer and the serial clock (CPHA must be cleared in NSSP mode). (Note: 2 - except the case when CRC is enabled at TI mode).
 - c) Select simplex or half-duplex mode by configuring RXONLY or BIDIMODE and BIDIOE (RXONLY and BIDIMODE can't be set at the same time).
 - d) Configure the LSBFIRST bit to define the frame format (Note: 2).
 - e) Configure the CRCL and CRCEN bits if CRC is needed (while SCK clock signal is at idle state).
 - f) Configure SSM and SSI (Notes: 2 & 3).
 - g) Configure the MSTR bit (in multimaster NSS configuration, avoid conflict state on NSS if master is configured to prevent MODF error).
3. Write to SPI_CR2 register:
 - a) Configure the DS[3:0] bits to select the data length for the transfer.
 - b) Configure SSOE (Notes: 1 & 2 & 3).
 - c) Set the FRF bit if the TI protocol is required (keep NSSP bit cleared in TI mode).
 - d) Set the NSSP bit if the NSS pulse mode between two data units is required (keep CHPA and TI bits cleared in NSSP mode).
 - e) Configure the RXTH bit. The RXFIFO threshold must be aligned to the read access size for the SPIx_DR register.
 - f) Initialize LDMA_TX and LDMA_RX bits if DMA is used in packed mode.
4. Write to SPI_CRCPR register: Configure the CRC polynomial if needed.
5. Write proper DMA registers: Configure DMA streams dedicated for SPI Tx and Rx in DMA registers if the DMA streams are used.

- Note:
- (1) Step is not required in slave mode.
 - (2) Step is not required in TI mode.
 - (3) Step is not required in NSSP mode.
 - (4) The step is not required in slave mode except slave working at TI mode

28.5.8 Procedure for enabling SPI

It is recommended to enable the SPI slave before the master sends the clock. If not, undesired data transmission might occur. The data register of the slave must already contain data to be sent before starting communication with the master (either on the first edge of the communication clock, or before the end of the ongoing communication if the clock signal is continuous). The SCK signal must be settled at an idle state level corresponding to the selected polarity before the SPI slave is enabled.

The master at full-duplex (or in any transmit-only mode) starts to communicate when the SPI is enabled and TXFIFO is not empty, or with the next write to TXFIFO.

In any master receive only mode (RXONLY=1 or BIDIMODE=1 & BIDIOE=0), master starts to communicate and the clock starts running immediately after SPI is enabled.

For handling DMA, follow the dedicated section.

28.5.9 Data transmission and reception procedures

RXFIFO and TXFIFO

All SPI data transactions pass through the 32-bit embedded FIFOs. This enables the SPI to work in a continuous flow, and prevents overruns when the data frame size is short. Each direction has its own FIFO called TXFIFO and RXFIFO. These FIFOs are used in all SPI modes except for receiver-only mode (slave or master) with CRC calculation enabled (see [Section 28.5.14: CRC calculation](#)).

The handling of FIFOs depends on the data exchange mode (duplex, simplex), data frame format (number of bits in the frame), access size performed on the FIFO data registers (8-bit or 16-bit), and whether or not data packing is used when accessing the FIFOs (see [Section 28.5.13: TI mode](#)).

A read access to the SPIx_DR register returns the oldest value stored in RXFIFO that has not been read yet. A write access to the SPIx_DR stores the written data in the TXFIFO at the end of a send queue. The read access must be always aligned with the RXFIFO threshold configured by the FRXTH bit in SPIx_CR2 register. FTLVL[1:0] and FRLVL[1:0] bits indicate the current occupancy level of both FIFOs.

A read access to the SPIx_DR register must be managed by the RXNE event. This event is triggered when data is stored in RXFIFO and the threshold (defined by FRXTH bit) is reached. When RXNE is cleared, RXFIFO is considered to be empty. In a similar way, write access of a data frame to be transmitted is managed by the TXE event. This event is triggered when the TXFIFO level is less than or equal to half of its capacity. Otherwise TXE is cleared and the TXFIFO is considered as full. In this way, RXFIFO can store up to four data frames, whereas TXFIFO can only store up to three when the data frame format is not greater than 8 bits. This difference prevents possible corruption of 3x 8-bit data frames already stored in the TXFIFO when software tries to write more data in 16-bit mode into TXFIFO. Both TXE and RXNE events can be polled or handled by interrupts. See [Figure 352](#) through [Figure 355](#).

Another way to manage the data exchange is to use DMA (see [Communication using DMA \(direct memory addressing\)](#)).

If the next data is received when the RXFIFO is full, an overrun event occurs (see description of OVR flag at [Section 28.5.10: SPI status flags](#)). An overrun event can be polled or handled by an interrupt.

The BSY bit being set indicates ongoing transaction of a current data frame. When the clock signal runs continuously, the BSY flag stays set between data frames at master but becomes low for a minimum duration of one SPI clock at slave between each data frame transfer.

Sequence handling

A few data frames can be passed at single sequence to complete a message. When transmission is enabled, a sequence begins and continues while any data is present in the TXFIFO of the master. The clock signal is provided continuously by the master until TXFIFO becomes empty, then it stops waiting for additional data.

In receive-only modes, half-duplex (BIDIMODE=1, BIDIOE=0) or simplex (BIDIMODE=0, RXONLY=1) the master starts the sequence immediately when both SPI is enabled and receive-only mode is activated. The clock signal is provided by the master and it does not stop until either SPI or receive-only mode is disabled by the master. The master receives data frames continuously up to this moment.

While the master can provide all the transactions in continuous mode (SCK signal is continuous) it has to respect slave capability to handle data flow and its content at anytime. When necessary, the master must slow down the communication and provide either a slower clock or separate frames or data sessions with sufficient delays. Be aware there is no underflow error signal for master or slave in SPI mode, and data from the slave is always transacted and processed by the master even if the slave could not prepare it correctly in time. It is preferable for the slave to use DMA, especially when data frames are shorter and bus rate is high.

Each sequence must be encased by the NSS pulse in parallel with the multislide system to select just one of the slaves for communication. In a single slave system it is not necessary to control the slave with NSS, but it is often better to provide the pulse here too, to synchronize the slave with the beginning of each data sequence. NSS can be managed by both software and hardware (see [Section 28.5.5: Slave select \(NSS\) pin management](#)).

When the BSY bit is set it signifies an ongoing data frame transaction. When the dedicated frame transaction is finished, the RXNE flag is raised. The last bit is just sampled and the complete data frame is stored in the RXFIFO.

Procedure for disabling the SPI

When SPI is disabled, it is mandatory to follow the disable procedures described in this paragraph. It is important to do this before the system enters a low-power mode when the peripheral clock is stopped. Ongoing transactions can be corrupted in this case. In some modes the disable procedure is the only way to stop continuous communication running.

Master in full-duplex or transmit only mode can finish any transaction when it stops providing data for transmission. In this case, the clock stops after the last data transaction. Special care must be taken in packing mode when an odd number of data frames are transacted to prevent some dummy byte exchange (refer to [Data packing](#) section). Before the SPI is disabled in these modes, the user must follow standard disable procedure. When

the SPI is disabled at the master transmitter while a frame transaction is ongoing or next data frame is stored in TXFIFO, the SPI behavior is not guaranteed.

When the master is in any receive only mode, the only way to stop the continuous clock is to disable the peripheral by SPE=0. This must occur in specific time window within last data frame transaction just between the sampling time of its first bit and before its last bit transfer starts (in order to receive a complete number of expected data frames and to prevent any additional “dummy” data reading after the last valid data frame). Specific procedure must be followed when disabling SPI in this mode.

Data received but not read remains stored in RXFIFO when the SPI is disabled, and must be processed the next time the SPI is enabled, before starting a new sequence. To prevent having unread data, ensure that RXFIFO is empty when disabling the SPI, by using the correct disabling procedure, or by initializing all the SPI registers with a software reset via the control of a specific register dedicated to peripheral reset (see the SPIiRST bits in the RCC_APBiRSTR registers).

Standard disable procedure is based on pulling BSY status together with FTLVL[1:0] to check if a transmission session is fully completed. This check can be done in specific cases, too, when it is necessary to identify the end of ongoing transactions, for example:

- When NSS signal is managed by software and master has to provide proper end of NSS pulse for slave, or
- When transactions’ streams from DMA or FIFO are completed while the last data frame or CRC frame transaction is still ongoing in the peripheral bus.

The correct disable procedure is (except when receive only mode is used):

1. Wait until FTLVL[1:0] = 00 (no more data to transmit).
2. Wait until BSY=0 (the last data frame is processed).
3. Disable the SPI (SPE=0).
4. Read data until FRLVL[1:0] = 00 (read all the received data).

The correct disable procedure for certain receive only modes is:

1. Interrupt the receive flow by disabling SPI (SPE=0) in the specific time window while the last data frame is ongoing.
2. Wait until BSY=0 (the last data frame is processed).
3. Read data until FRLVL[1:0] = 00 (read all the received data).

Note:

If packing mode is used and an odd number of data frames with a format less than or equal to 8 bits (fitting into one byte) has to be received, FRXTH must be set when FRLVL[1:0] = 01, in order to generate the RXNE event to read the last odd data frame and to keep good FIFO pointer alignment.

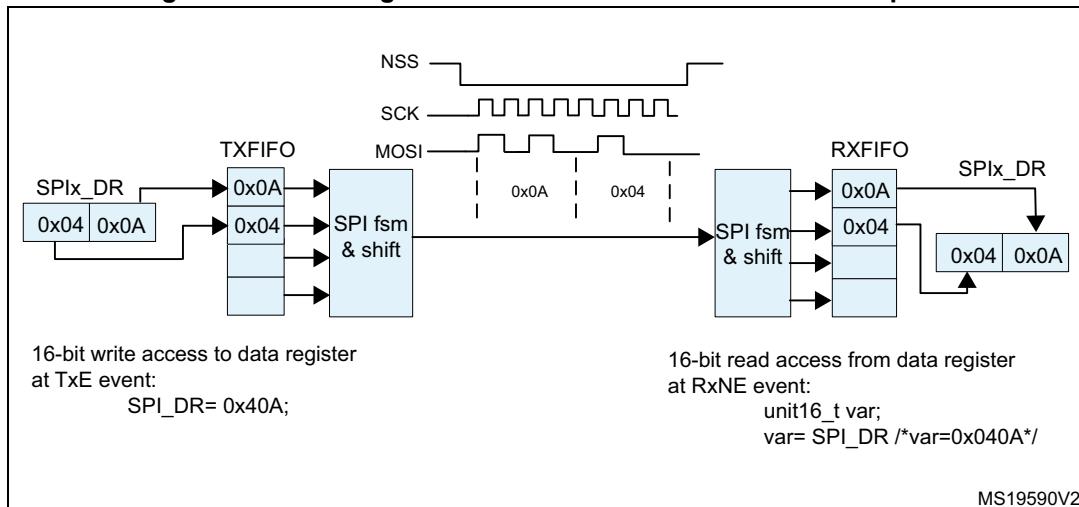
Data packing

When the data frame size fits into one byte (less than or equal to 8 bits), data packing is used automatically when any read or write 16-bit access is performed on the SPIx_DR register. The double data frame pattern is handled in parallel in this case. At first, the SPI operates using the pattern stored in the LSB of the accessed word, then with the other half stored in the MSB. [Figure 351](#) provides an example of data packing mode sequence handling. Two data frames are sent after the single 16-bit access the SPIx_DR register of the transmitter. This sequence can generate just one RXNE event in the receiver if the RXFIFO threshold is set to 16 bits (FRXTH=0). The receiver then has to access both data frames by a single 16-bit read of SPIx_DR as a response to this single RXNE event. The

RxFIFO threshold setting and the following read access must be always kept aligned at the receiver side, as data can be lost if it is not in line.

A specific problem appears if an odd number of such “fit into one byte” data frames must be handled. On the transmitter side, writing the last data frame of any odd sequence with an 8-bit access to SPIx_DR is enough. The receiver has to change the Rx_FIFO threshold level for the last data frame received in the odd sequence of frames in order to generate the RXNE event.

Figure 351. Packing data in FIFO for transmission and reception



1. In this example: Data size DS[3:0] is 4-bit configured, CPOL=0, CPHA=1 and LSBFIRST =0. The Data storage is always right aligned while the valid bits are performed on the bus only, the content of LSB byte goes first on the bus, the unused bits are not taken into account on the transmitter side and padded by zeros at the receiver side.

Communication using DMA (direct memory addressing)

To operate at its maximum speed and to facilitate the data register read/write process required to avoid overrun, the SPI features a DMA capability, which implements a simple request/acknowledge protocol.

A DMA access is requested when the TXE or RXNE enable bit in the SPIx_CR2 register is set. Separate requests must be issued to the Tx and Rx buffers.

- In transmission, a DMA request is issued each time TXE is set to 1. The DMA then writes to the SPIx_DR register.
- In reception, a DMA request is issued each time RXNE is set to 1. The DMA then reads the SPIx_DR register.

See [Figure 352](#) through [Figure 355](#).

When the SPI is used only to transmit data, it is possible to enable only the SPI Tx DMA channel. In this case, the OVR flag is set because the data received is not read. When the SPI is used only to receive data, it is possible to enable only the SPI Rx DMA channel.

In transmission mode, when the DMA has written all the data to be transmitted (the TCIF flag is set in the DMA_ISR register), the BSY flag can be monitored to ensure that the SPI communication is complete. This is required to avoid corrupting the last transmission before disabling the SPI or entering the Stop mode. The software must first wait until FTLVL[1:0]=00 and then until BSY=0.

When starting communication using DMA, to prevent DMA channel management raising error events, these steps must be followed in order:

1. Enable DMA Rx buffer in the RXDMAEN bit in the SPI_CR2 register, if DMA Rx is used.
2. Enable DMA streams for Tx and Rx in DMA registers, if the streams are used.
3. Enable DMA Tx buffer in the TXDMAEN bit in the SPI_CR2 register, if DMA Tx is used.
4. Enable the SPI by setting the SPE bit.

To close communication it is mandatory to follow these steps in order:

1. Disable DMA streams for Tx and Rx in the DMA registers, if the streams are used.
2. Disable the SPI by following the SPI disable procedure.
3. Disable DMA Tx and Rx buffers by clearing the TXDMAEN and RXDMAEN bits in the SPI_CR2 register, if DMA Tx and/or DMA Rx are used.

Packing with DMA

If the transfers are managed by DMA (TXDMAEN and RXDMAEN set in the SPIx_CR2 register) packing mode is enabled/disabled automatically depending on the PSIZE value configured for SPI TX and the SPI RX DMA channel. If the DMA channel PSIZE value is equal to 16-bit and SPI data size is less than or equal to 8-bit, then packing mode is enabled. The DMA then automatically manages the write operations to the SPIx_DR register.

If data packing mode is used and the number of data to transfer is not a multiple of two, the LDMA_TX/LDMA_RX bits must be set. The SPI then considers only one data for the transmission or reception to serve the last DMA transfer (for more details refer to [Data packing on page 964](#).)

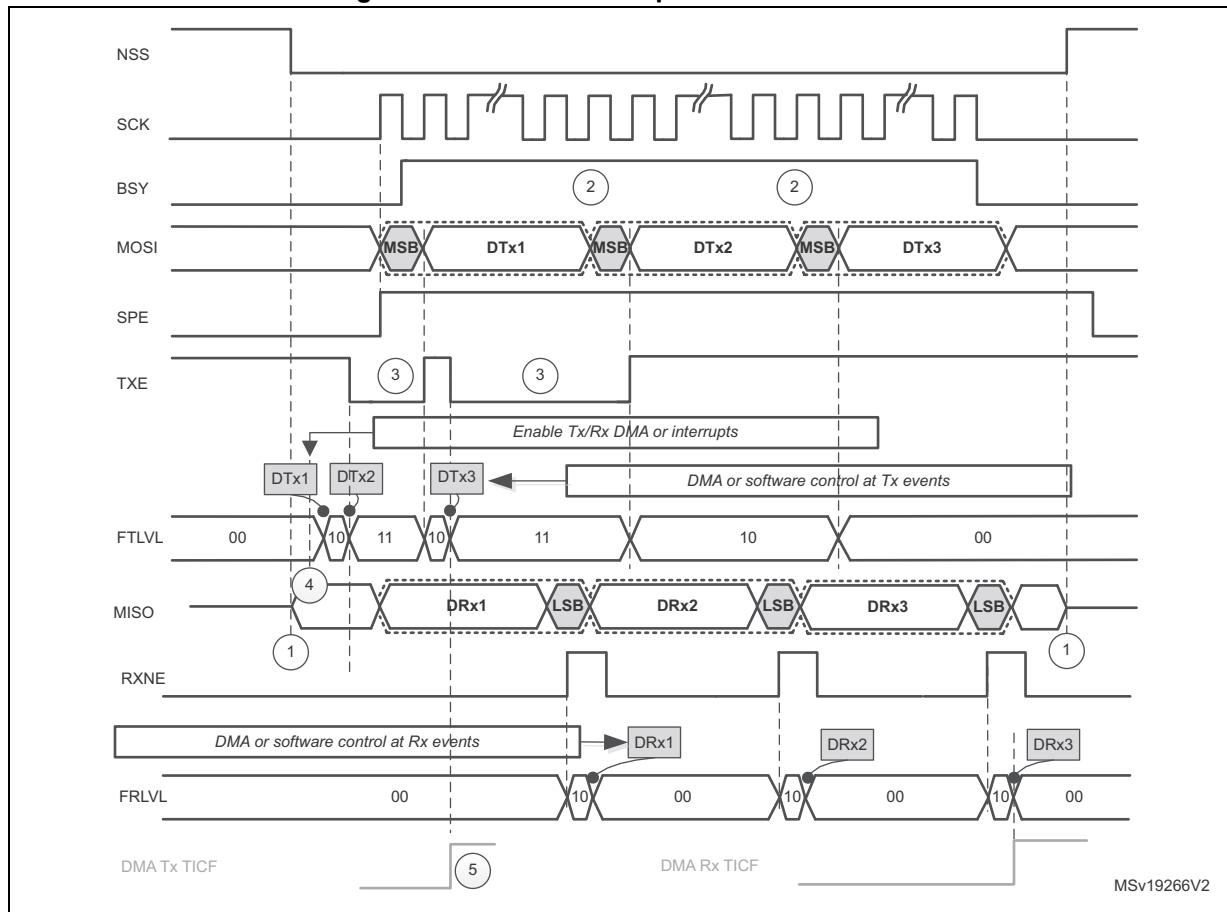
Communication diagrams

Some typical timing schemes are explained in this section. These schemes are valid no matter if the SPI events are handled by polling, interrupts or DMA. For simplicity, the LSBFIRST=0, CPOL=0 and CPHA=1 setting is used as a common assumption here. No complete configuration of DMA streams is provided.

The following numbered notes are common for [Figure 352 on page 968](#) through [Figure 355 on page 971](#).

1. The slave starts to control MISO line as NSS is active and SPI is enabled, and is disconnected from the line when one of them is released. Sufficient time must be provided for the slave to prepare data dedicated to the master in advance before its transaction starts.
At the master, the SPI peripheral takes control at MOSI and SCK signals (occasionally at NSS signal as well) only if SPI is enabled. If SPI is disabled the SPI peripheral is disconnected from GPIO logic, so the levels at these lines depends on GPIO setting exclusively.
2. At the master, BSY stays active between frames if the communication (clock signal) is continuous. At the slave, BSY signal always goes down for at least one clock cycle between data frames.
3. The TXE signal is cleared only if TxFIFO is full.
4. The DMA arbitration process starts just after the TXDMAEN bit is set. The TXE interrupt is generated just after the TXEIE is set. As the TXE signal is at an active level, data transfers to TxFIFO start, until TxFIFO becomes full or the DMA transfer completes.
5. If all the data to be sent can fit into TxFIFO, the DMA Tx TCIF flag can be raised even before communication on the SPI bus starts. This flag always rises before the SPI transaction is completed.
6. The CRC value for a package is calculated continuously frame by frame in the SPIx_TXCRCR and SPIx_RXCRCR registers. The CRC information is processed after the entire data package has completed, either automatically by DMA (Tx channel must be set to the number of data frames to be processed) or by SW (the user must handle CRCNEXT bit during the last data frame processing).
While the CRC value calculated in SPIx_TXCRCR is simply sent out by transmitter, received CRC information is loaded into Rx FIFO and then compared with the SPIx_RXCRCR register content (CRC error flag can be raised here if any difference). This is why the user must take care to flush this information from the FIFO, either by software reading out all the stored content of Rx FIFO, or by DMA when the proper number of data frames is preset for Rx channel (number of data frames + number of CRC frames) (see the settings at the example assumption).
7. In data packed mode, TxE and RxNE events are paired and each read/write access to the FIFO is 16 bits wide until the number of data frames are even. If the Tx FIFO is $\frac{3}{4}$ full FTLVL status stays at FIFO full level. That is why the last odd data frame cannot be stored before the Tx FIFO becomes $\frac{1}{2}$ full. This frame is stored into Tx FIFO with an 8-bit access either by software or automatically by DMA when LDMA_TX control is set.
8. To receive the last odd data frame in packed mode, the Rx threshold must be changed to 8-bit when the last data frame is processed, either by software setting FRXTH=1 or automatically by a DMA internal signal when LDMA_RX is set.

Figure 352. Master full-duplex communication



Assumptions for master full-duplex communication example:

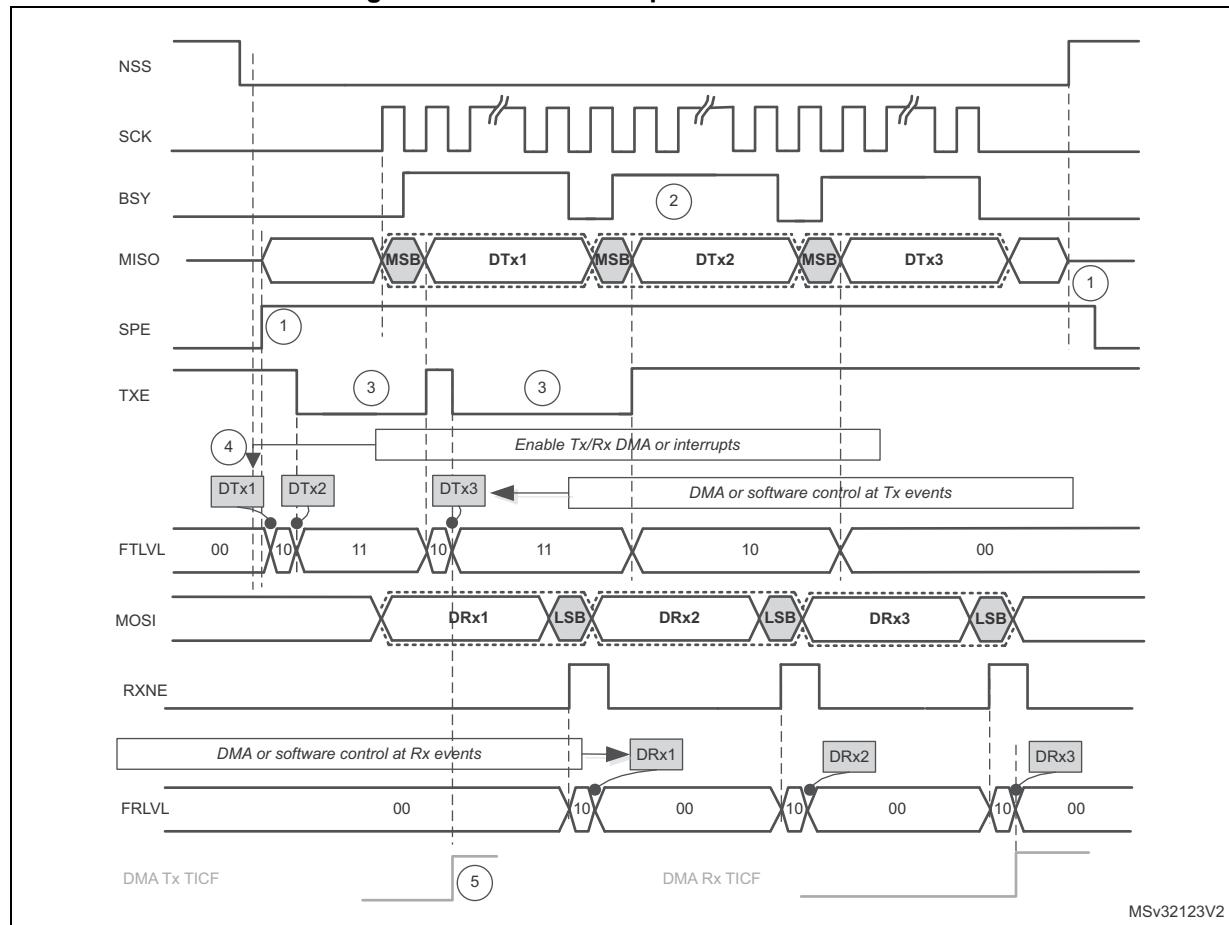
- Data size > 8 bit

If DMA is used:

- Number of Tx frames transacted by DMA is set to 3
- Number of Rx frames transacted by DMA is set to 3

See also : [Communication diagrams on page 967](#) for details about common assumptions and notes.

Figure 353. Slave full-duplex communication



Assumptions for slave full-duplex communication example:

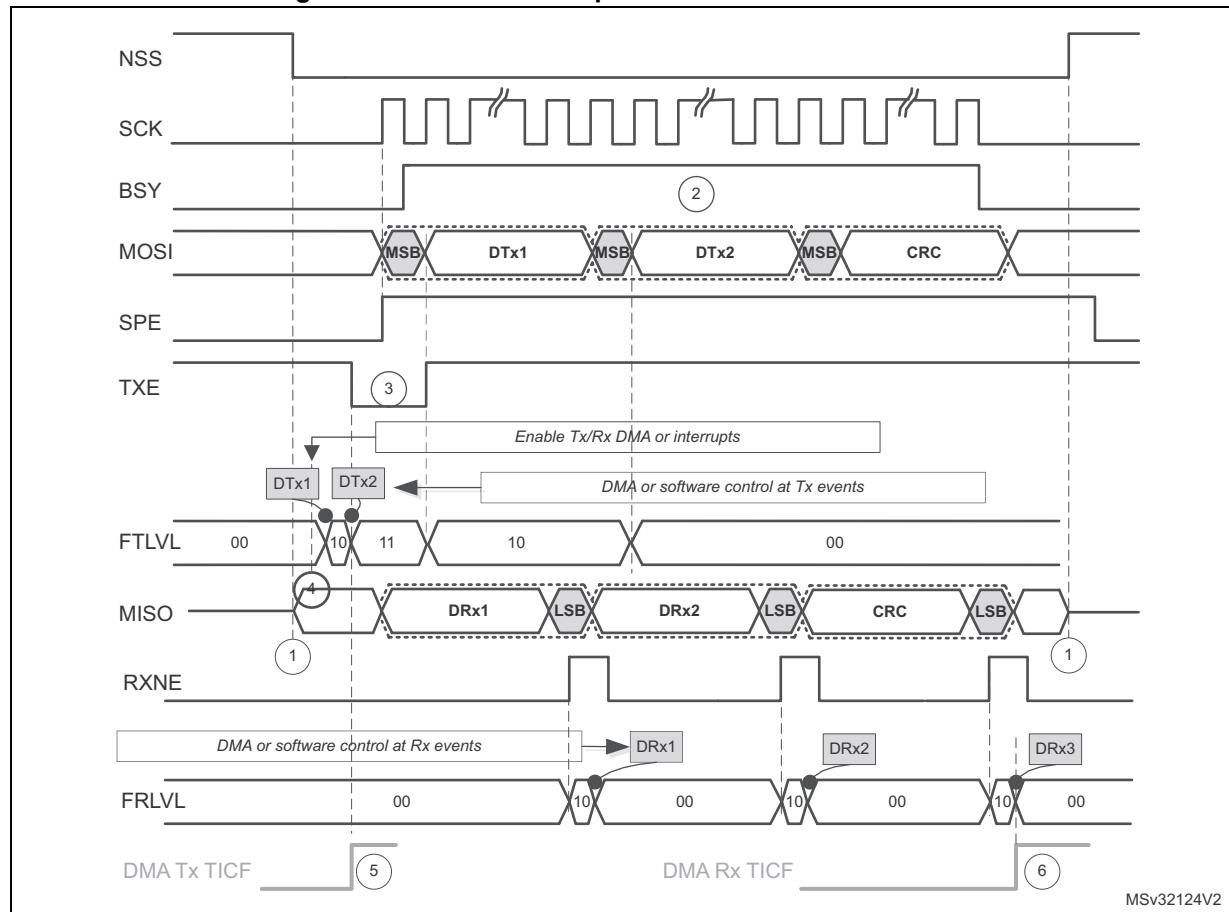
- Data size > 8 bit

If DMA is used:

- Number of Tx frames transacted by DMA is set to 3
- Number of Rx frames transacted by DMA is set to 3

See also : [Communication diagrams on page 967](#) for details about common assumptions and notes.

Figure 354. Master full-duplex communication with CRC



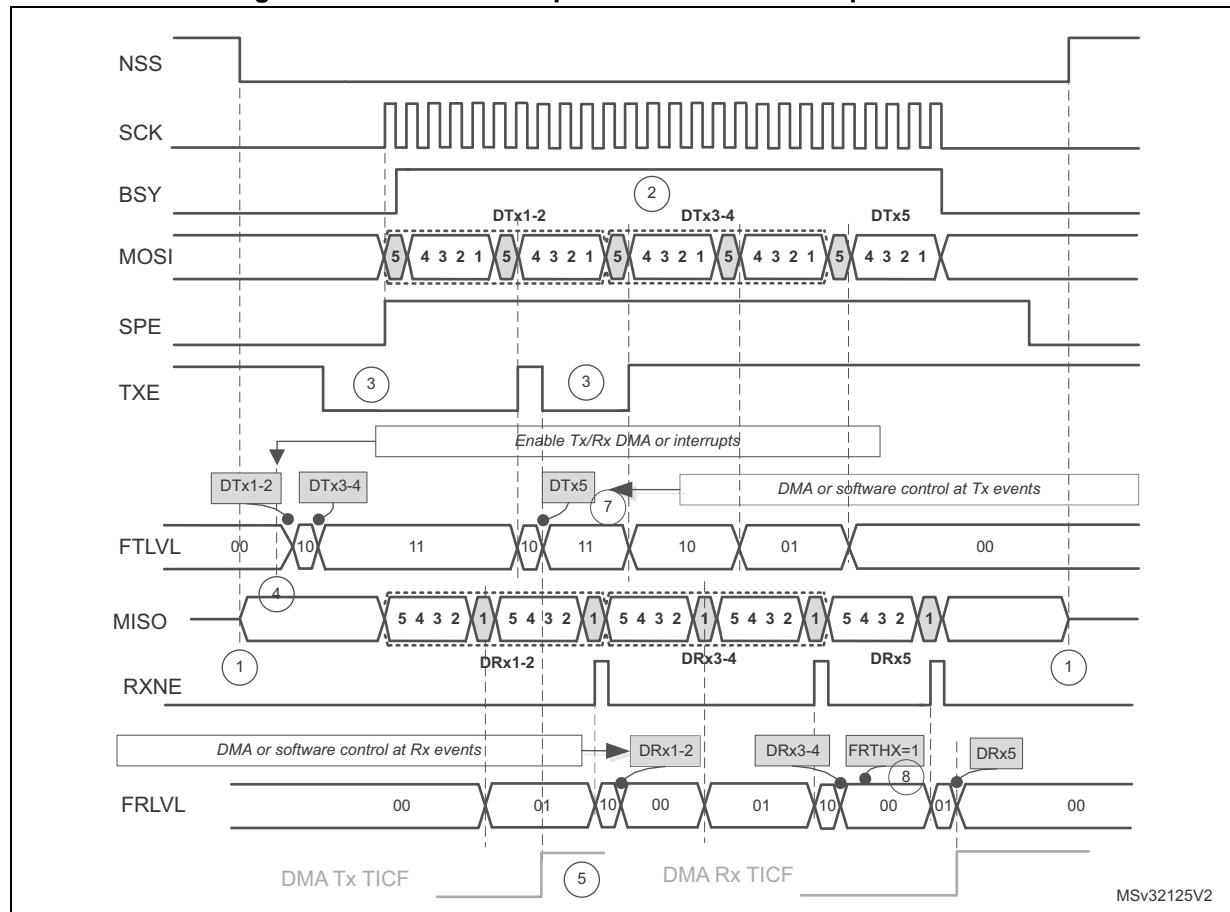
Assumptions for master full-duplex communication with CRC example:

- Data size = 16 bit
- CRC enabled

If DMA is used:

- Number of Tx frames transacted by DMA is set to 2
- Number of Rx frames transacted by DMA is set to 3

See also : [Communication diagrams on page 967](#) for details about common assumptions and notes.

Figure 355. Master full-duplex communication in packed mode

Assumptions for master full-duplex communication in packed mode example:

- Data size = 5 bit
- Read/write FIFO is performed mostly by 16-bit access
- FRXTH=0

If DMA is used:

- Number of Tx frames to be transacted by DMA is set to 3
- Number of Rx frames to be transacted by DMA is set to 3
- PSIZE for both Tx and Rx DMA channel is set to 16-bit
- LDMA_TX=1 and LDMA_RX=1

See also : [Communication diagrams on page 967](#) for details about common assumptions and notes.

28.5.10 SPI status flags

Three status flags are provided for the application to completely monitor the state of the SPI bus.

Tx buffer empty flag (TXE)

The TXE flag is set when transmission TXFIFO has enough space to store data to send. TXE flag is linked to the TXFIFO level. The flag goes high and stays high until the TXFIFO level is lower or equal to 1/2 of the FIFO depth. An interrupt can be generated if the TXEIE bit in the SPIx_CR2 register is set. The bit is cleared automatically when the TXFIFO level becomes greater than 1/2.

Rx buffer not empty (RXNE)

The RXNE flag is set depending on the FRXTH bit value in the SPIx_CR2 register:

- If FRXTH is set, RXNE goes high and stays high until the RXFIFO level is greater or equal to 1/4 (8-bit).
- If FRXTH is cleared, RXNE goes high and stays high until the RXFIFO level is greater than or equal to 1/2 (16-bit).

An interrupt can be generated if the RXNEIE bit in the SPIx_CR2 register is set.

The RXNE is cleared by hardware automatically when the above conditions are no longer true.

Busy flag (BSY)

The BSY flag is set and cleared by hardware (writing to this flag has no effect).

When BSY is set, it indicates that a data transfer is in progress on the SPI (the SPI bus is busy).

The BSY flag can be used in certain modes to detect the end of a transfer so that the software can disable the SPI or its peripheral clock before entering a low-power mode which does not provide a clock for the peripheral. This avoids corrupting the last transfer.

The BSY flag is also useful for preventing write collisions in a multimaster system.

The BSY flag is cleared under any one of the following conditions:

- When the SPI is correctly disabled
- When a fault is detected in Master mode (MODF bit set to 1)
- In Master mode, when it finishes a data transmission and no new data is ready to be sent
- In Slave mode, when the BSY flag is set to '0' for at least one SPI clock cycle between each data transfer.

Note:

When the next transmission can be handled immediately by the master (e.g. if the master is in Receive-only mode or its Transmit FIFO is not empty), communication is continuous and the BSY flag remains set to '1' between transfers on the master side. Although this is not the case with a slave, it is recommended to use always the TXE and RXNE flags (instead of the BSY flags) to handle data transmission or reception operations.

28.5.11 SPI error flags

An SPI interrupt is generated if one of the following error flags is set and interrupt is enabled by setting the ERRIE bit.

Overrun flag (OVR)

An overrun condition occurs when data is received by a master or slave and the RXFIFO has not enough space to store this received data. This can happen if the software or the DMA did not have enough time to read the previously received data (stored in the RXFIFO) or when space for data storage is limited e.g. the RXFIFO is not available when CRC is enabled in receive only mode so in this case the reception buffer is limited into a single data frame buffer (see [Section 28.5.14: CRC calculation](#)).

When an overrun condition occurs, the newly received value does not overwrite the previous one in the RXFIFO. The newly received value is discarded and all data transmitted subsequently is lost. Clearing the OVR bit is done by a read access to the SPI_DR register followed by a read access to the SPI_SR register.

Mode fault (MODF)

Mode fault occurs when the master device has its internal NSS signal (NSS pin in NSS hardware mode, or SSI bit in NSS software mode) pulled low. This automatically sets the MODF bit. Master mode fault affects the SPI interface in the following ways:

- The MODF bit is set and an SPI interrupt is generated if the ERRIE bit is set.
- The SPE bit is cleared. This blocks all output from the device and disables the SPI interface.
- The MSTR bit is cleared, thus forcing the device into slave mode.

Use the following software sequence to clear the MODF bit:

1. Make a read or write access to the SPIx_SR register while the MODF bit is set.
2. Then write to the SPIx_CR1 register.

To avoid any multiple slave conflicts in a system comprising several MCUs, the NSS pin must be pulled high during the MODF bit clearing sequence. The SPE and MSTR bits can be restored to their original state after this clearing sequence. As a security, hardware does not allow the SPE and MSTR bits to be set while the MODF bit is set. In a slave device the MODF bit cannot be set except as the result of a previous multimaster conflict.

CRC error (CRCERR)

This flag is used to verify the validity of the value received when the CRCEN bit in the SPIx_CR1 register is set. The CRCERR flag in the SPIx_SR register is set if the value received in the shift register does not match the receiver SPIx_RXCRCR value. The flag is cleared by the software.

TI mode frame format error (FRE)

A TI mode frame format error is detected when an NSS pulse occurs during an ongoing communication when the SPI is operating in slave mode and configured to conform to the TI mode protocol. When this error occurs, the FRE flag is set in the SPIx_SR register. The SPI is not disabled when an error occurs, the NSS pulse is ignored, and the SPI waits for the next NSS pulse before starting a new transfer. The data may be corrupted since the error detection may result in the loss of two data bytes.

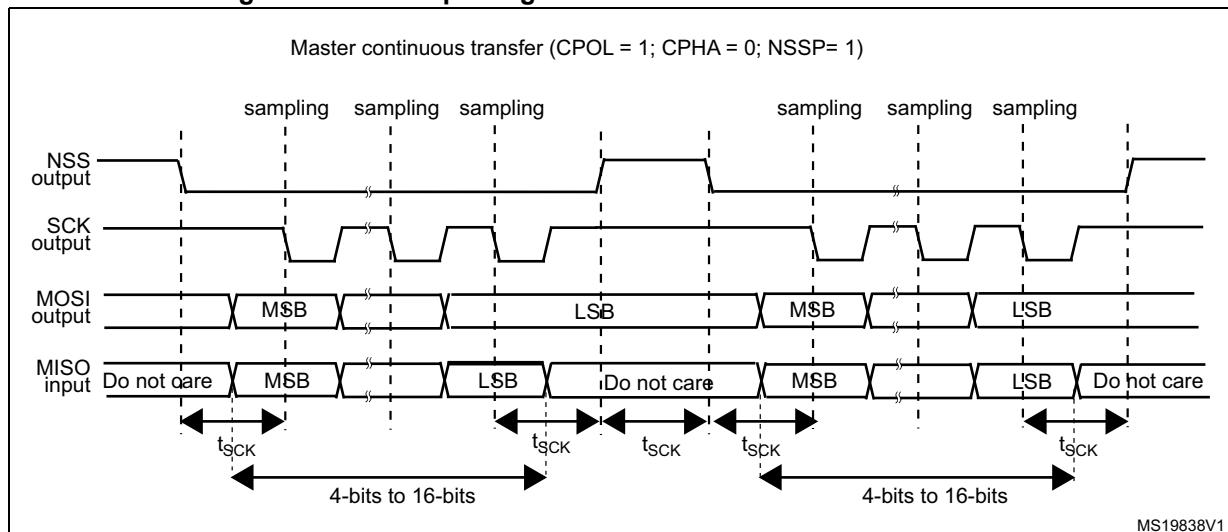
The FRE flag is cleared when SPIx_SR register is read. If the ERRIE bit is set, an interrupt is generated on the NSS error detection. In this case, the SPI should be disabled because data consistency is no longer guaranteed and communications should be reinitiated by the master when the slave SPI is enabled again.

28.5.12 NSS pulse mode

This mode is activated by the NSSP bit in the SPIx_CR2 register and it takes effect only if the SPI interface is configured as Motorola SPI master (FRF=0) with capture on the first edge (SPIx_CR1 CPHA = 0, CPOL setting is ignored). When activated, an NSS pulse is generated between two consecutive data frame transfers when NSS stays at high level for the duration of one clock period at least. This mode allows the slave to latch data. NSSP pulse mode is designed for applications with a single master-slave pair.

Figure 356 illustrates NSS pin management when NSSP pulse mode is enabled.

Figure 356. NSSP pulse generation in Motorola SPI master mode



Note: Similar behavior is encountered when CPOL = 0. In this case the sampling edge is the *rising* edge of SCK, and NSS assertion and deassertion refer to this sampling edge.

28.5.13 TI mode

TI protocol in master mode

The SPI interface is compatible with the TI protocol. The FRF bit of the SPIx_CR2 register can be used to configure the SPI to be compliant with this protocol.

The clock polarity and phase are forced to conform to the TI protocol requirements whatever the values set in the SPIx_CR1 register. NSS management is also specific to the TI protocol which makes the configuration of NSS management through the SPIx_CR1 and SPIx_CR2 registers (SSM, SSI, SSOE) impossible in this case.

In slave mode, the SPI baud rate prescaler is used to control the moment when the MISO pin state changes to HiZ when the current transaction finishes (see *Figure 357*). Any baud rate can be used, making it possible to determine this moment with optimal flexibility. However, the baud rate is generally set to the external master clock baud rate. The delay for the MISO signal to become HiZ ($t_{release}$) depends on internal resynchronization and on the

baud rate value set in through the BR[2:0] bits in the SPIx_CR1 register. It is given by the formula:

$$\frac{t_{\text{baud_rate}}}{2} + 4 \times t_{\text{pclk}} < t_{\text{release}} < \frac{t_{\text{baud_rate}}}{2} + 6 \times t_{\text{pclk}}$$

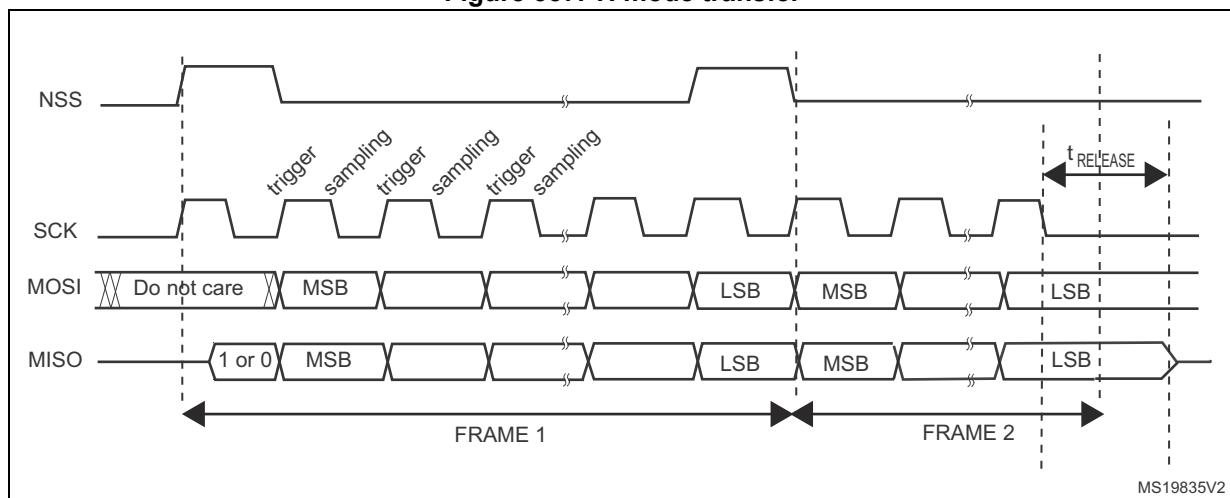
If the slave detects a misplaced NSS pulse during a data frame transaction the TIFRE flag is set.

If the data size is equal to 4-bits or 5-bits, the master in full-duplex mode or transmit-only mode uses a protocol with one more dummy data bit added after LSB. TI NSS pulse is generated above this dummy bit clock cycle instead of the LSB in each period.

This feature is not available for Motorola SPI communications (FRF bit set to 0).

Figure 357: TI mode transfer shows the SPI communication waveforms when TI mode is selected.

Figure 357. TI mode transfer



28.5.14 CRC calculation

Two separate CRC calculators are implemented in order to check the reliability of transmitted and received data. The SPI offers CRC8 or CRC16 calculation independently of the frame data length, which can be fixed to 8-bit or 16-bit. For all the other data frame lengths, no CRC is available.

CRC principle

CRC calculation is enabled by setting the CRCEN bit in the SPIx_CR1 register before the SPI is enabled (SPE = 1). The CRC value is calculated using an odd programmable polynomial on each bit. The calculation is processed on the sampling clock edge defined by the CPHA and CPOL bits in the SPIx_CR1 register. The calculated CRC value is checked automatically at the end of the data block as well as for transfer managed by CPU or by the DMA. When a mismatch is detected between the CRC calculated internally on the received data and the CRC sent by the transmitter, a CRCERR flag is set to indicate a data corruption error. The right procedure for handling the CRC calculation depends on the SPI configuration and the chosen transfer management.

Note: *The polynomial value should only be odd. No even values are supported.*

CRC transfer managed by CPU

Communication starts and continues normally until the last data frame has to be sent or received in the SPIx_DR register. Then CRCNEXT bit has to be set in the SPIx_CR1 register to indicate that the CRC frame transaction will follow after the transaction of the currently processed data frame. The CRCNEXT bit must be set before the end of the last data frame transaction. CRC calculation is frozen during CRC transaction.

The received CRC is stored in the RXFIFO like a data byte or word. That is why in CRC mode only, the reception buffer has to be considered as a single 16-bit buffer used to receive only one data frame at a time.

A CRC-format transaction usually takes one more data frame to communicate at the end of data sequence. However, when setting an 8-bit data frame checked by 16-bit CRC, two more frames are necessary to send the complete CRC.

When the last CRC data is received, an automatic check is performed comparing the received value and the value in the SPIx_RXCRC register. Software has to check the CRCERR flag in the SPIx_SR register to determine if the data transfers were corrupted or not. Software clears the CRCERR flag by writing '0' to it.

After the CRC reception, the CRC value is stored in the RXFIFO and must be read in the SPIx_DR register in order to clear the RXNE flag.

CRC transfer managed by DMA

When SPI communication is enabled with CRC communication and DMA mode, the transmission and reception of the CRC at the end of communication is automatic (with the exception of reading CRC data in receive only mode). The CRCNEXT bit does not have to be handled by the software. The counter for the SPI transmission DMA channel has to be set to the number of data frames to transmit excluding the CRC frame. On the receiver side, the received CRC value is handled automatically by DMA at the end of the transaction but user must take care to flush out received CRC information from RXFIFO as it is always loaded into it. In full-duplex mode, the counter of the reception DMA channel can be set to the number of data frames to receive including the CRC, which means, for example, in the specific case of an 8-bit data frame checked by 16-bit CRC:

$$\text{DMA_RX} = \text{Numb_of_data} + 2$$

In receive only mode, the DMA reception channel counter should contain only the amount of data transferred, excluding the CRC calculation. Then based on the complete transfer from DMA, all the CRC values must be read back by software from FIFO as it works as a single buffer in this mode.

At the end of the data and CRC transfers, the CRCERR flag in the SPIx_SR register is set if corruption occurred during the transfer.

If packing mode is used, the LDMA_RX bit needs managing if the number of data is odd.

Resetting the SPIx_TXCRC and SPIx_RXCRC values

The SPIx_TXCRC and SPIx_RXCRC values are cleared automatically when new data is sampled after a CRC phase. This allows the use of DMA circular mode (not available in receive-only mode) in order to transfer data without any interruption, (several data blocks covered by intermediate CRC checking phases).

If the SPI is disabled during a communication the following sequence must be followed:

1. Disable the SPI
2. Clear the CRCEN bit
3. Enable the CRCEN bit
4. Enable the SPI

Note:

When the SPI interface is configured as a slave, the NSS internal signal needs to be kept low during transaction of the CRC phase once the CRCNEXT signal is released. That is why the CRC calculation can't be used at NSS Pulse mode when NSS hardware mode should be applied at slave normally (see more details at the product errata sheet).

At TI mode, despite the fact that clock phase and clock polarity setting is fixed and independent on SPIx_CR1 register, the corresponding setting CPOL=0 CPHA=1 has to be kept at the SPIx_CR1 register anyway if CRC is applied. In addition, the CRC calculation has to be reset between sessions by SPI disable sequence with re-enable the CRCEN bit described above at both master and slave side, else CRC calculation can be corrupted at this specific mode.

28.6 SPI interrupts

During SPI communication an interrupts can be generated by the following events:

- Transmit TXFIFO ready to be loaded
- Data received in Receive RXFIFO
- Master mode fault
- Overrun error
- TI frame format error
- CRC protocol error

Interrupts can be enabled and disabled separately.

Table 171. SPI interrupt requests

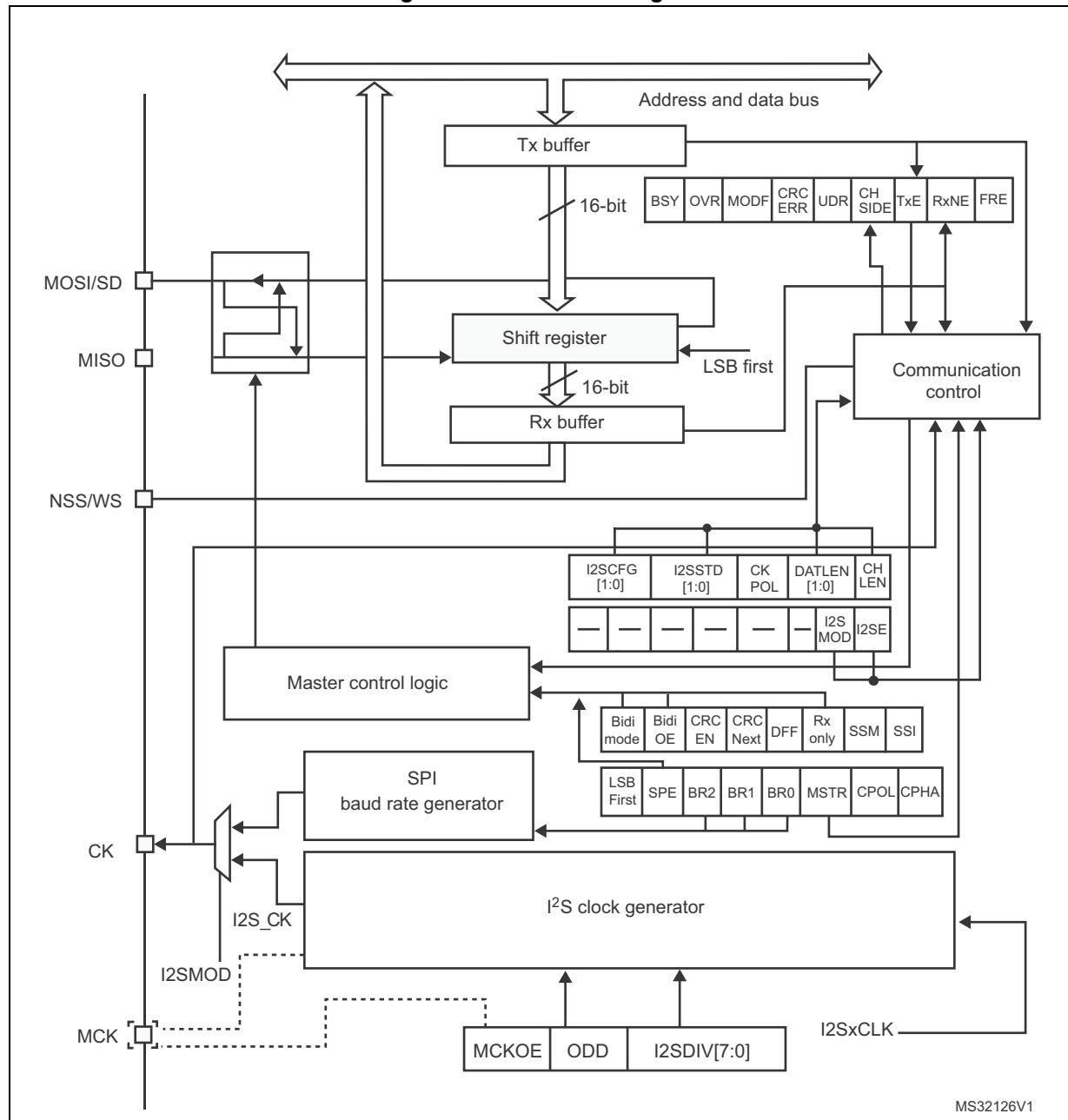
Interrupt event	Event flag	Enable Control bit
Transmit TXFIFO ready to be loaded	TXE	TXEIE
Data received in RXFIFO	RXNE	RXNEIE
Master Mode fault event	MODF	ERRIE
Overrun error	OVR	
TI frame format error	FRE	
CRC protocol error	CRCERR	

28.7 I2S functional description

28.7.1 I2S general description

The block diagram of the I2S is shown in [Figure 358](#).

Figure 358. I2S block diagram



1. MCK is mapped on the MISO pin.

The SPI can function as an audio I2S interface when the I2S capability is enabled (by setting the I2SMOD bit in the SPIx_I2SCFGR register). This interface mainly uses the same pins, flags and interrupts as the SPI.

The I2S shares three common pins with the SPI:

- SD: Serial Data (mapped on the MOSI pin) to transmit or receive the two time-multiplexed data channels (in half-duplex mode only).
- WS: Word Select (mapped on the NSS pin) is the data control signal output in master mode and input in slave mode.
- CK: Serial Clock (mapped on the SCK pin) is the serial clock output in master mode and serial clock input in slave mode.

An additional pin can be used when a master clock output is needed for some external audio devices:

- MCK: Master Clock (mapped separately) is used, when the I2S is configured in master mode (and when the MCKOE bit in the SPIx_I2SPR register is set), to output this additional clock generated at a preconfigured frequency rate equal to $256 \times f_S$, where f_S is the audio sampling frequency.

The I2S uses its own clock generator to produce the communication clock when it is set in master mode. This clock generator is also the source of the master clock output. Two additional registers are available in I²S mode. One is linked to the clock generator configuration SPIx_I2SPR and the other one is a generic I2S configuration register SPIx_I2SCFGR (audio standard, slave/master mode, data format, packet frame, clock polarity, etc.).

The SPIx_CR1 register and all CRC registers are not used in the I²S mode. Likewise, the SSOE bit in the SPIx_CR2 register and the MODF and CRCERR bits in the SPIx_SR are not used.

The I2S uses the same SPI register for data transfer (SPIx_DR) in 16-bit wide mode.

28.7.2 I2S full duplex

Figure 359 shows how to perform full-duplex communications using two SPI/I2S instances. In this case, the WS and CK IOs of both SPI/I2S must be connected together.

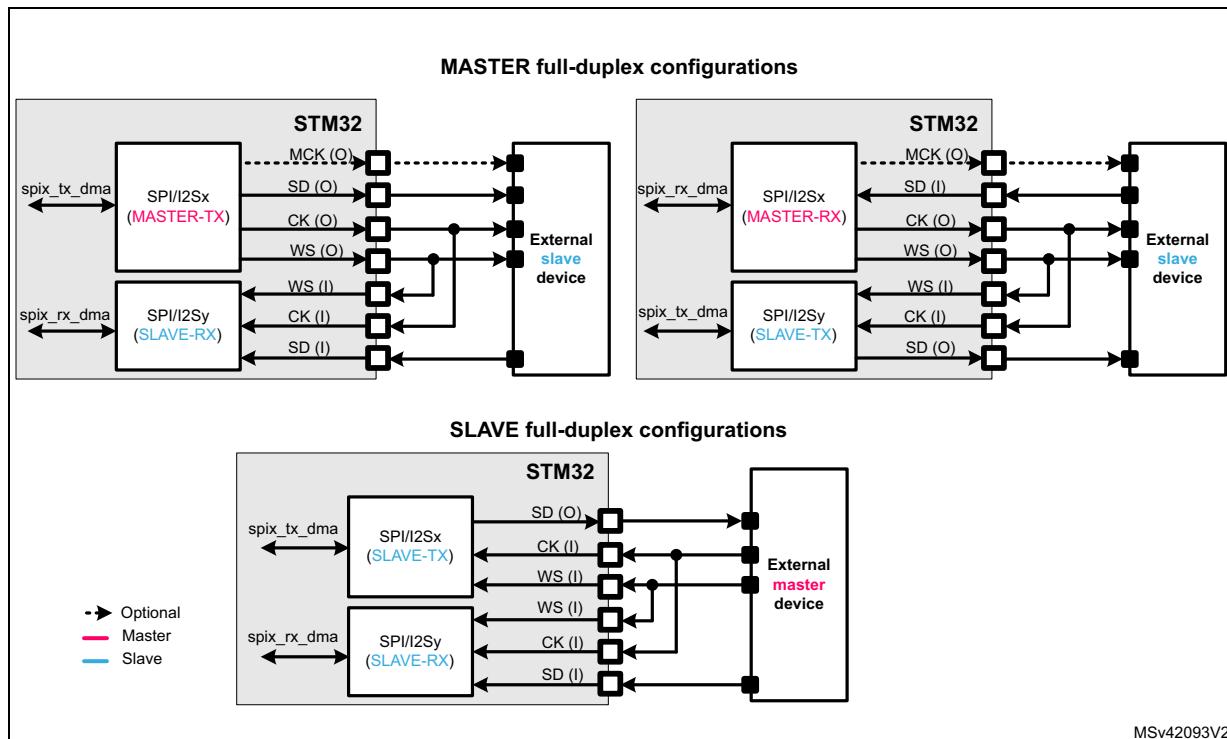
For the master full-duplex mode, one of the SPI/I2S block must be programmed in master (I2SCFG = '10' or '11'), and the other SPI/I2S block must be programmed in slave (I2SCFG = '00' or '01'). The MCK can be generated or not, depending on the application needs.

For the slave full-duplex mode, both SPI/I2S blocks must be programmed in slave. One of them in the slave receiver (I2SCFG = '01'), and the other in the slave transmitter (I2SCFG = '00'). The master external device then provides the bit clock (CK) and the frame synchronization (WS).

Note that the full-duplex mode can be used for all the supported standards: I²S Philips, MSB-justified, LSB-justified and PCM.

For the full-duplex mode, both SPI/I2S instances must use the same standard, with the same parameters: I2SMOD, I2SSTD, CKPOL, PCMSYNC, DATLEN and CHLEN must contain the same value on both instances.

Figure 359. Full-duplex communication



28.7.3 Supported audio protocols

The three-line bus has to handle only audio data generally time-multiplexed on two channels: the right channel and the left channel. However there is only one 16-bit register for transmission or reception. So, it is up to the software to write into the data register the appropriate value corresponding to each channel side, or to read the data from the data register and to identify the corresponding channel by checking the CHSIDE bit in the SPIx_SR register. Channel left is always sent first followed by the channel right (CHSIDE has no meaning for the PCM protocol).

Four data and packet frames are available. Data may be sent with a format of:

- 16-bit data packed in a 16-bit frame
- 16-bit data packed in a 32-bit frame
- 24-bit data packed in a 32-bit frame
- 32-bit data packed in a 32-bit frame

When using 16-bit data extended on 32-bit packet, the first 16 bits (MSB) are the significant bits, the 16-bit LSB is forced to 0 without any need for software action or DMA request (only one read/write operation).

The 24-bit and 32-bit data frames need two CPU read or write operations to/from the SPIx_DR register or two DMA operations if the DMA is preferred for the application. For 24-bit data frame specifically, the 8 non-significant bits are extended to 32 bits with 0-bits (by hardware).

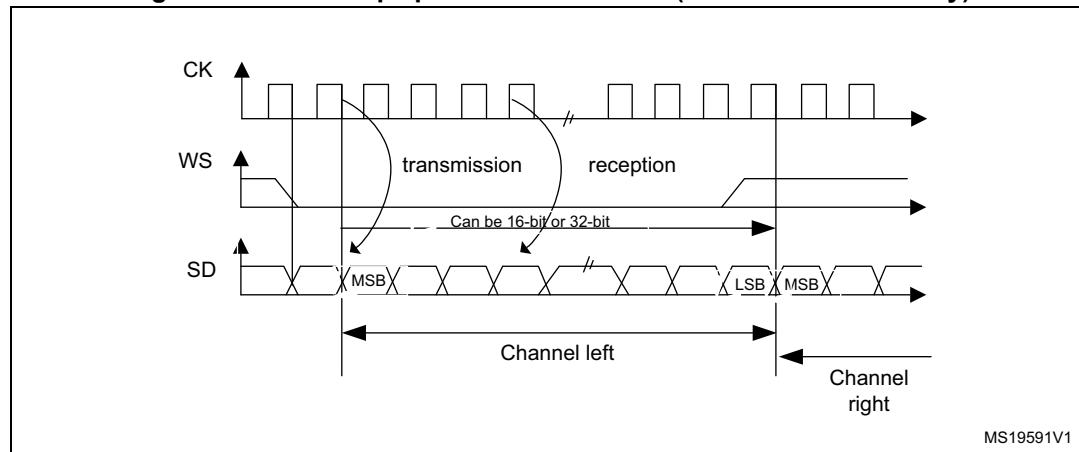
For all data formats and communication standards, the most significant bit is always sent first (MSB first).

The I²S interface supports four audio standards, configurable using the I2SSTD[1:0] and PCMSYNC bits in the SPIx_I2SCFGR register.

I²S Philips standard

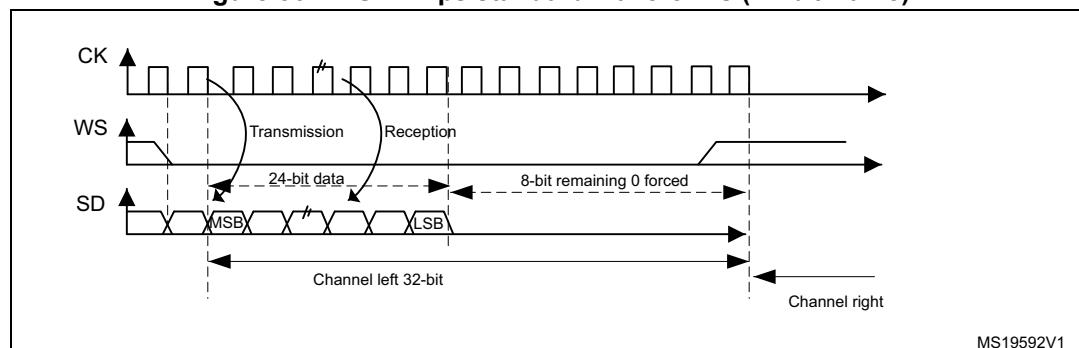
For this standard, the WS signal is used to indicate which channel is being transmitted. It is activated one CK clock cycle before the first bit (MSB) is available.

Figure 360. I²S Philips protocol waveforms (16/32-bit full accuracy)



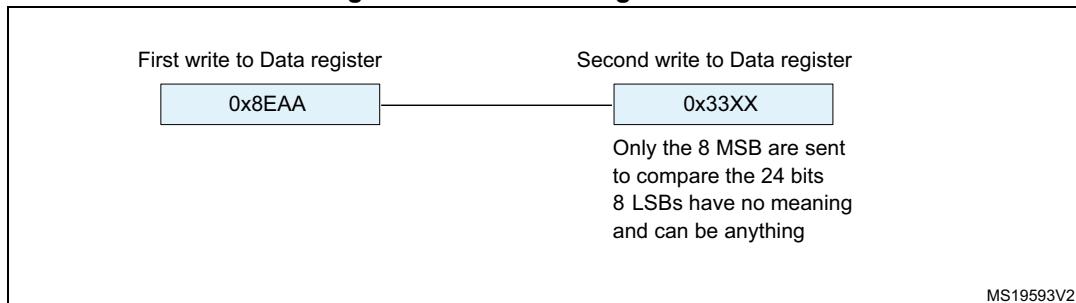
Data are latched on the falling edge of CK (for the transmitter) and are read on the rising edge (for the receiver). The WS signal is also latched on the falling edge of CK.

Figure 361. I²S Philips standard waveforms (24-bit frame)

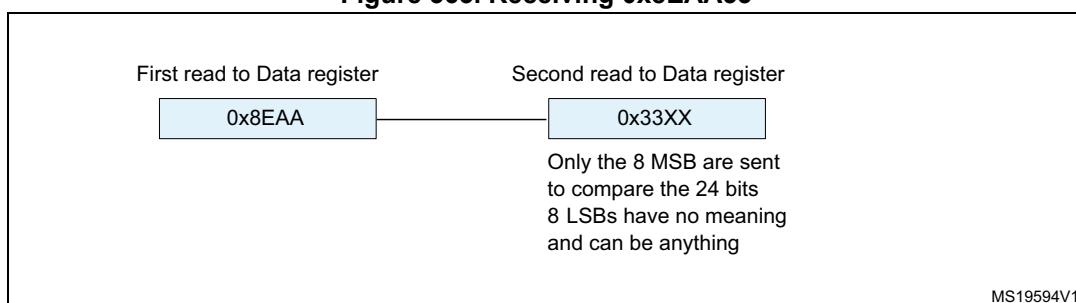
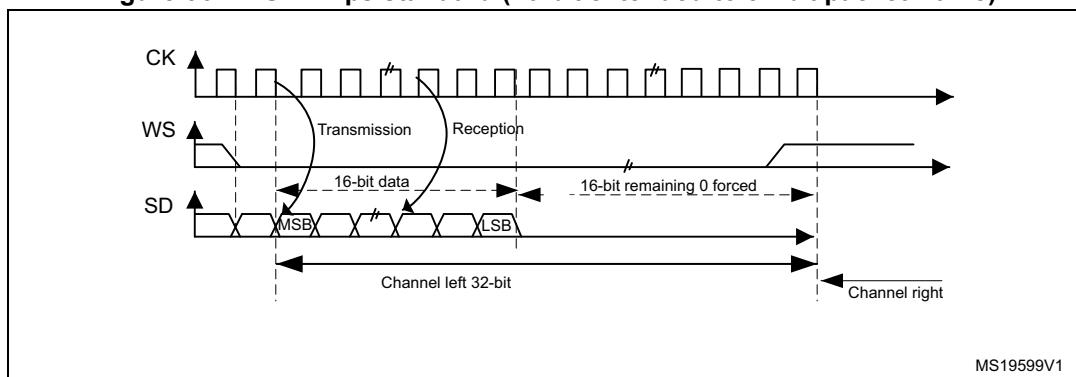


This mode needs two write or read operations to/from the SPIx_DR register.

- In transmission mode:
If 0x8EAA33 has to be sent (24-bit):

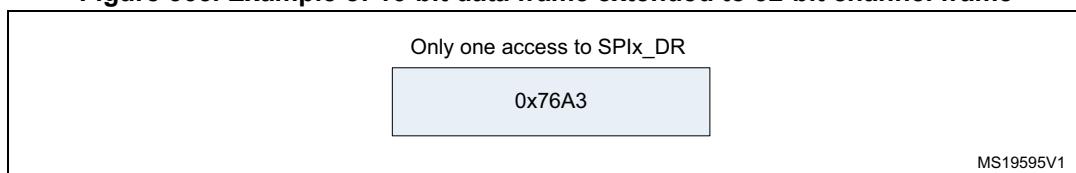
Figure 362. Transmitting 0x8EAA33

- In reception mode:
If data 0x8EAA33 is received:

Figure 363. Receiving 0x8EAA33**Figure 364. I²S Philips standard (16-bit extended to 32-bit packet frame)**

When 16-bit data frame extended to 32-bit channel frame is selected during the I2S configuration phase, only one access to the SPIx_DR register is required. The 16 remaining bits are forced by hardware to 0x0000 to extend the data to 32-bit format.

If the data to transmit or the received data are 0x76A3 (0x76A30000 extended to 32-bit), the operation shown in [Figure 365](#) is required.

Figure 365. Example of 16-bit data frame extended to 32-bit channel frame

For transmission, each time an MSB is written to SPIx_DR, the TXE flag is set and its interrupt, if allowed, is generated to load the SPIx_DR register with the new value to send. This takes place even if 0x0000 have not yet been sent because it is done by hardware.

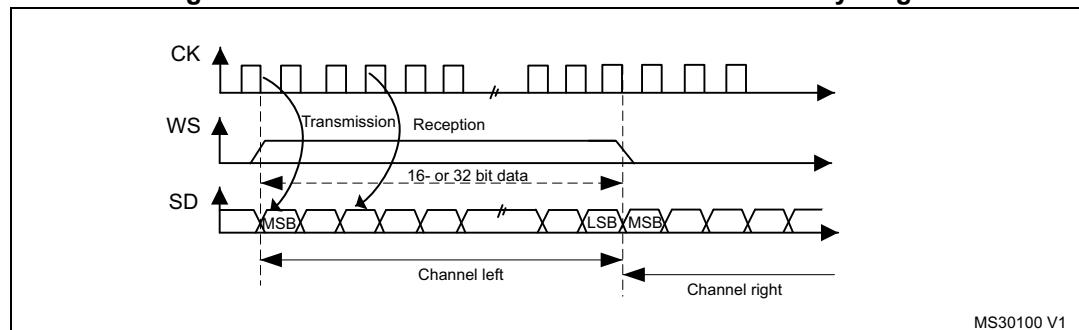
For reception, the RXNE flag is set and its interrupt, if allowed, is generated when the first 16 MSB half-word is received.

In this way, more time is provided between two write or read operations, which prevents underrun or overrun conditions (depending on the direction of the data transfer).

MSB justified standard

For this standard, the WS signal is generated at the same time as the first data bit, which is the MSBit.

Figure 366. MSB Justified 16-bit or 32-bit full-accuracy length



Data are latched on the falling edge of CK (for transmitter) and are read on the rising edge (for the receiver).

Figure 367. MSB justified 24-bit frame length

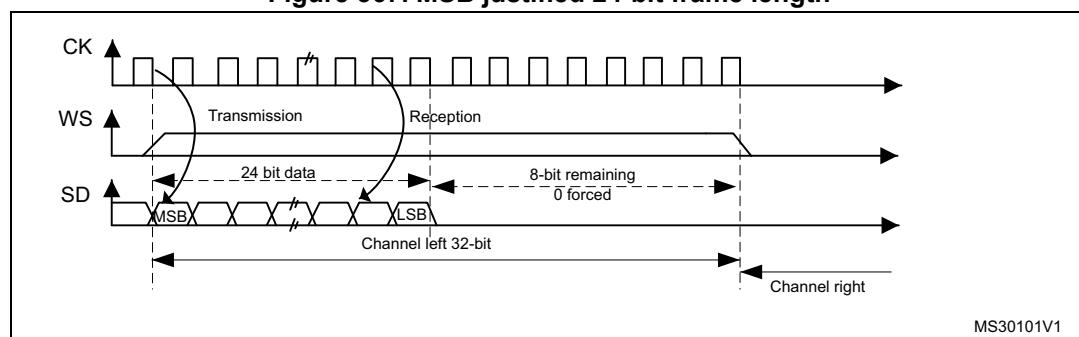
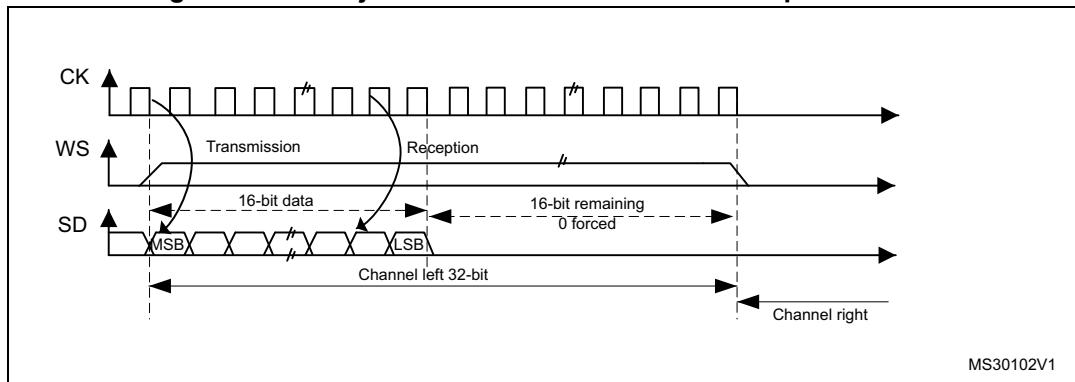
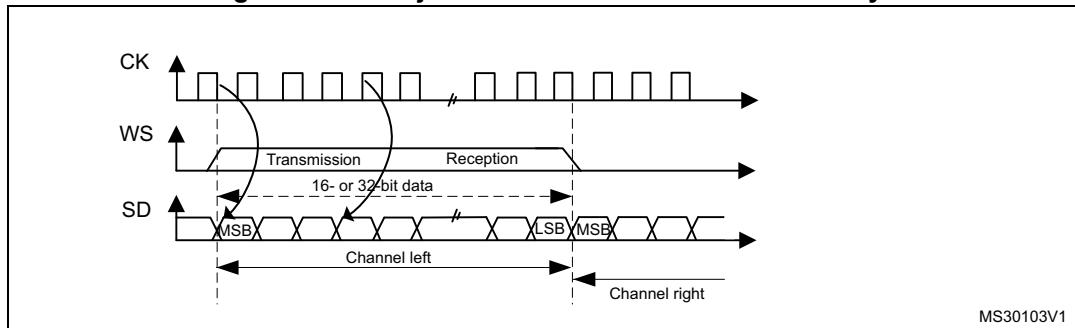
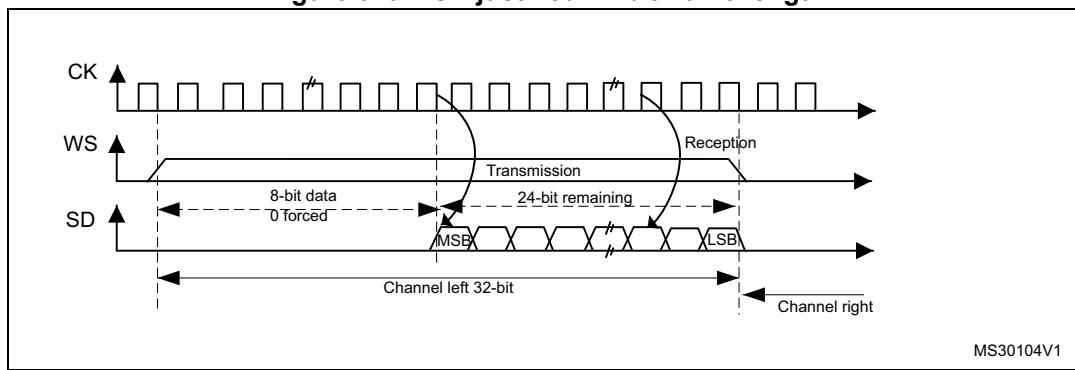


Figure 368. MSB justified 16-bit extended to 32-bit packet frame**LSB justified standard**

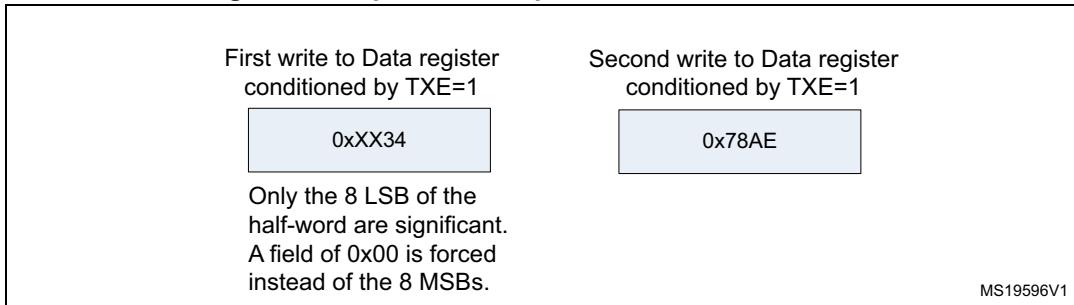
This standard is similar to the MSB justified standard (no difference for the 16-bit and 32-bit full-accuracy frame formats).

The sampling of the input and output signals is the same as for the I²S Philips standard.

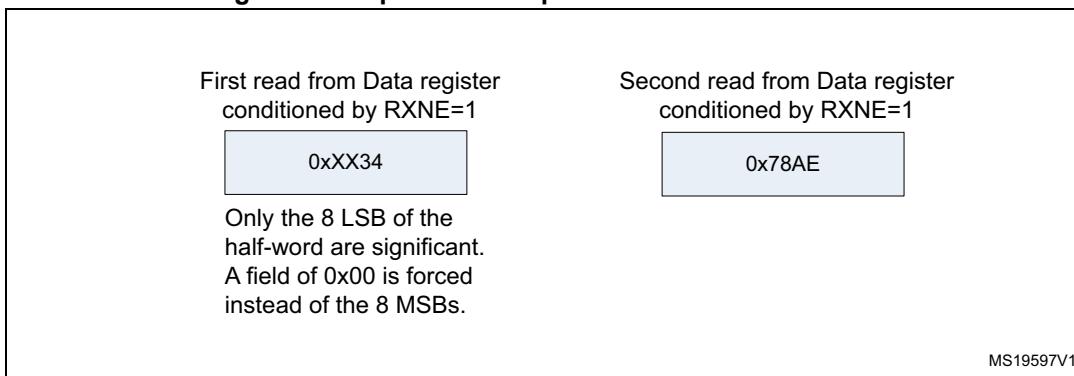
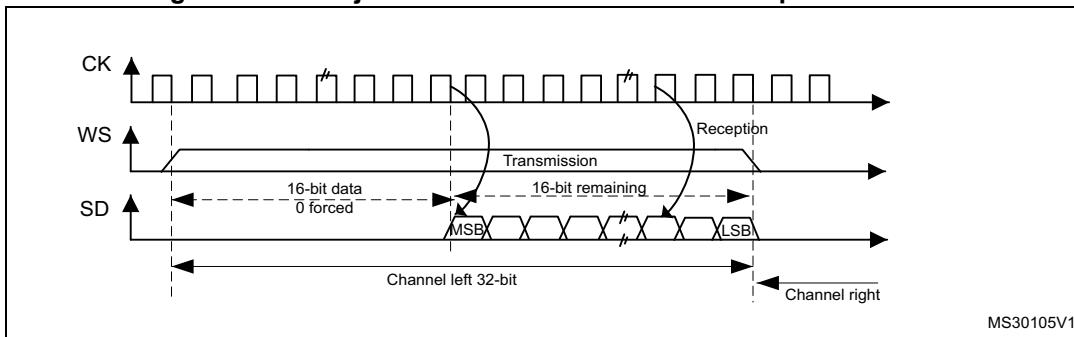
Figure 369. LSB justified 16-bit or 32-bit full-accuracy**Figure 370. LSB justified 24-bit frame length**

- In transmission mode:

If data 0x3478AE have to be transmitted, two write operations to the SPIx_DR register are required by software or by DMA. The operations are shown below.

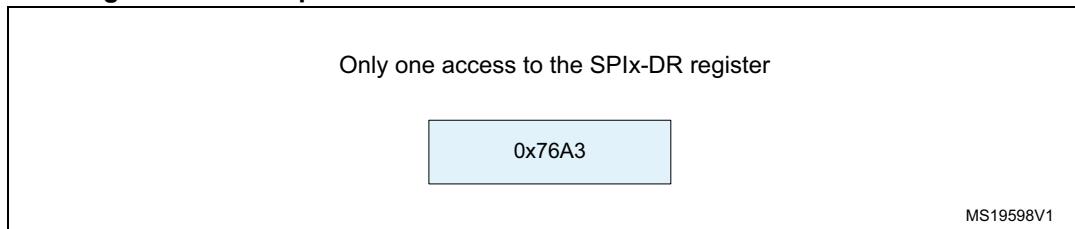
Figure 371. Operations required to transmit 0x3478AE

- In reception mode:
If data 0x3478AE are received, two successive read operations from the SPIx_DR register are required on each RXNE event.

Figure 372. Operations required to receive 0x3478AE**Figure 373. LSB justified 16-bit extended to 32-bit packet frame**

When 16-bit data frame extended to 32-bit channel frame is selected during the I2S configuration phase, Only one access to the SPIx_DR register is required. The 16 remaining bits are forced by hardware to 0x0000 to extend the data to 32-bit format. In this case it corresponds to the half-word MSB.

If the data to transmit or the received data are 0x76A3 (0x0000 76A3 extended to 32-bit), the operation shown in [Figure 374](#) is required.

Figure 374. Example of 16-bit data frame extended to 32-bit channel frame

In transmission mode, when a TXE event occurs, the application has to write the data to be transmitted (in this case 0x76A3). The 0x000 field is transmitted first (extension on 32-bit). The TXE flag is set again as soon as the effective data (0x76A3) is sent on SD.

In reception mode, RXNE is asserted as soon as the significant half-word is received (and not the 0x0000 field).

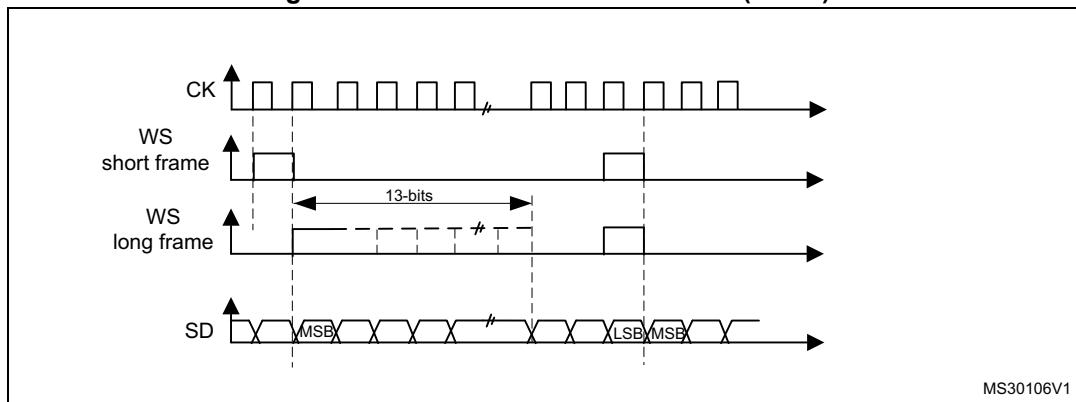
In this way, more time is provided between two write or read operations to prevent underrun or overrun conditions.

PCM standard

For the PCM standard, there is no need to use channel-side information. The two PCM modes (short and long frame) are available and configurable using the PCMSYNC bit in SPIx_I2SCFGR register.

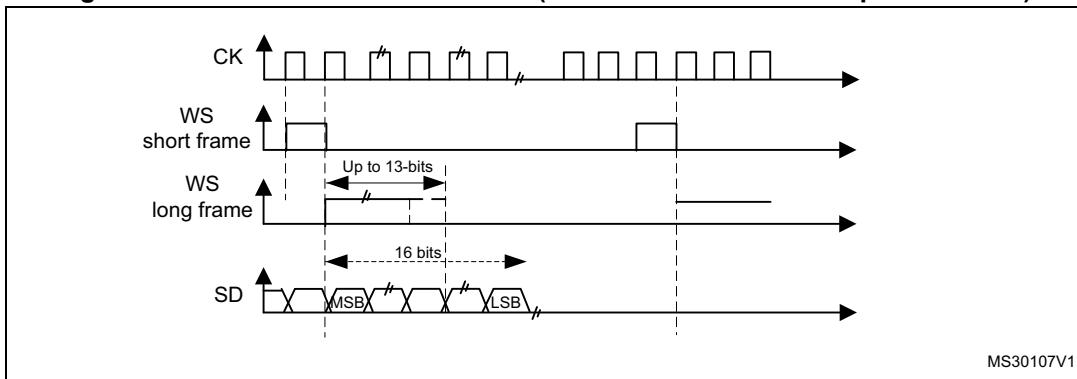
In PCM mode, the output signals (WS, SD) are sampled on the rising edge of CK signal. The input signals (WS, SD) are captured on the falling edge of CK.

Note that CK and WS are configured as output in MASTER mode.

Figure 375. PCM standard waveforms (16-bit)

For long frame synchronization, the WS signal assertion time is fixed to 13 bits in master mode.

For short frame synchronization, the WS synchronization signal is only one cycle long.

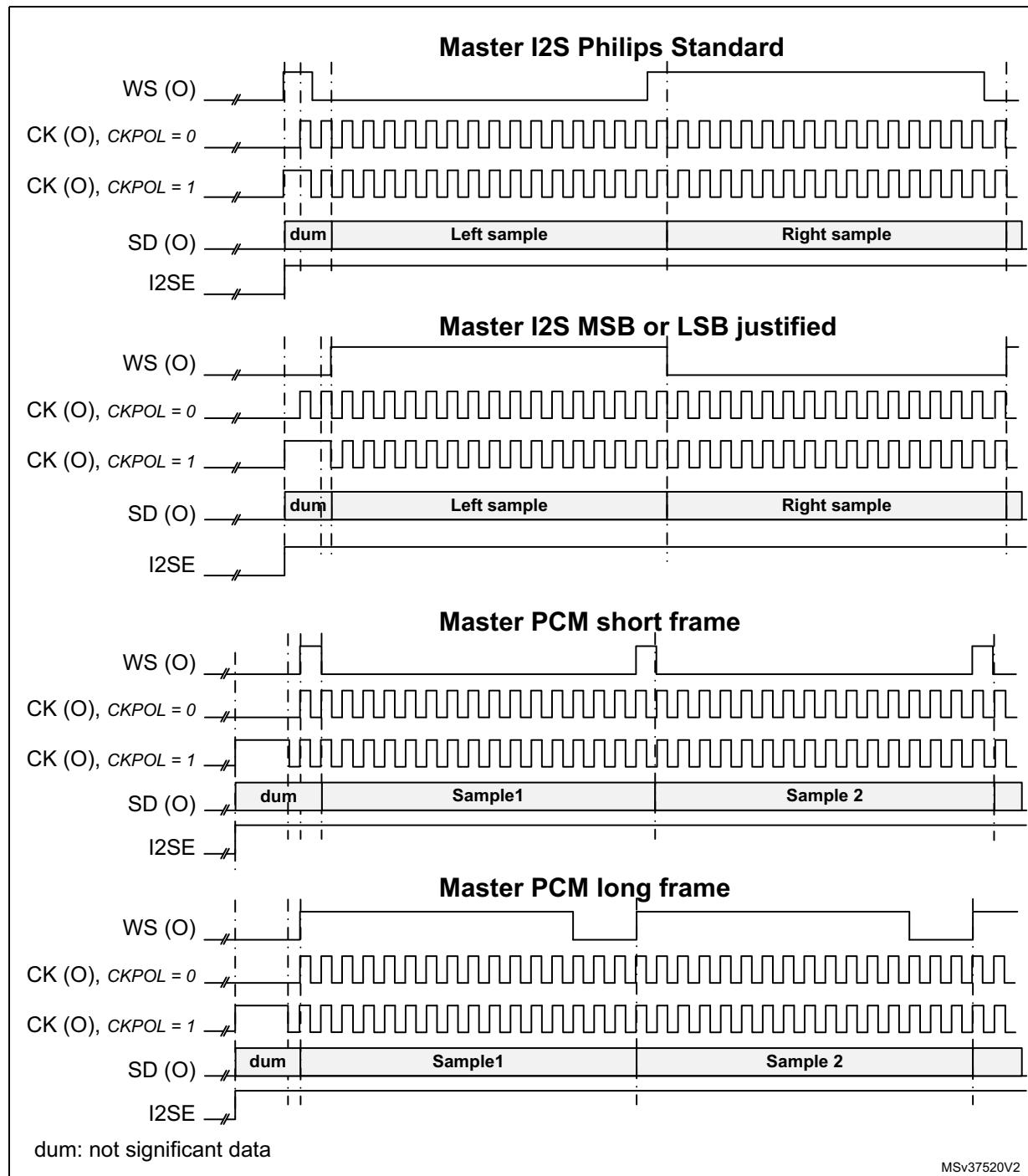
Figure 376. PCM standard waveforms (16-bit extended to 32-bit packet frame)

Note: For both modes (master and slave) and for both synchronizations (short and long), the number of bits between two consecutive pieces of data (and so two synchronization signals) needs to be specified (DATLEN and CHLEN bits in the SPIx_I2SCFGR register) even in slave mode.

28.7.4 Start-up description

The [Figure 377](#) shows how the serial interface is handled in MASTER mode, when the SPI/I2S is enabled (via I2SE bit). It shows as well the effect of CKPOL on the generated signals.

Figure 377. Start sequence in master mode



In slave mode, the way the frame synchronization is detected, depends on the value of ASTRTEN bit.

If ASTRTEN = 0, when the audio interface is enabled (I2SE = 1), then the hardware waits for the appropriate transition on the incoming WS signal, using the CK signal.

The appropriate transition is a falling edge on WS signal when I²S Philips Standard is used, or a rising edge for other standards. The falling edge is detected by sampling first WS to 1 and then to 0, and vice-versa for the rising edge detection.

If ASTRTEN = 1, the user has to enable the audio interface before the WS becomes active. This means that the I2SE bit must be set to 1 when WS = 1 for I²S Philips standard, or when WS = 0 for other standards.

28.7.5 Clock generator

The I²S bit rate determines the data flow on the I²S data line and the I²S clock signal frequency.

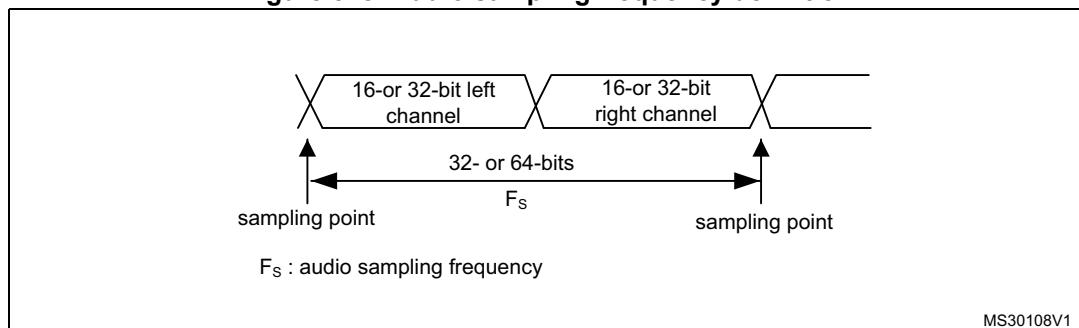
I²S bit rate = number of bits per channel × number of channels × sampling audio frequency

For a 16-bit audio, left and right channel, the I²S bit rate is calculated as follows:

$$\text{I}^2\text{S bit rate} = 16 \times 2 \times f_S$$

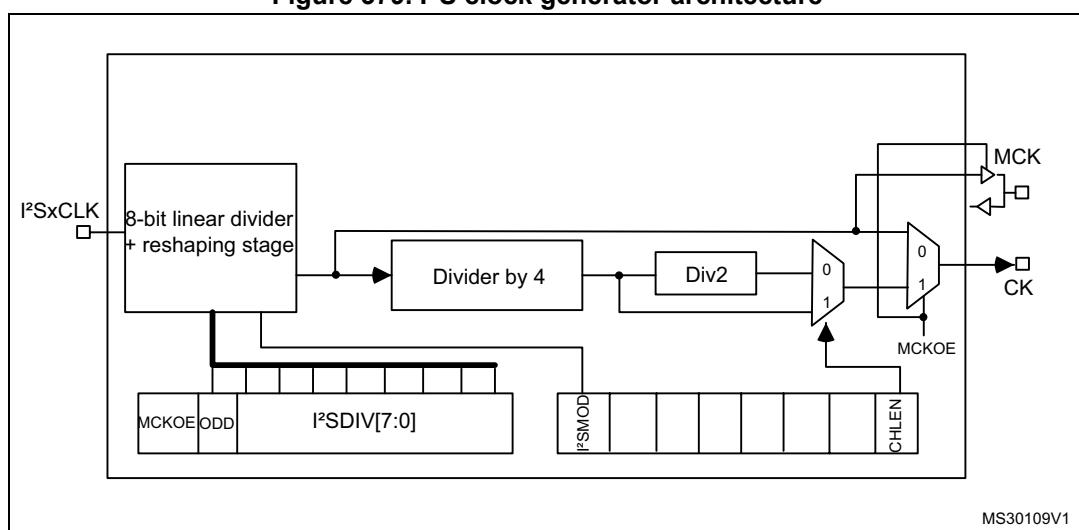
It will be: I²S bit rate = 32 × 2 × f_S if the packet length is 32-bit wide.

Figure 378. Audio sampling frequency definition



When the master mode is configured, a specific action needs to be taken to properly program the linear divider in order to communicate with the desired audio frequency.

Figure 379. I²S clock generator architecture



- Where x can be 2 or 3.

Figure 379 presents the communication clock architecture. The I2SxCLK clock is provided by the reset and clock controller (RCC) of the product. The I2SxCLK clock can be asynchronous with respect to the SPI/I2S APB clock.

Warning: In addition, it is mandatory to keep the I2SxCLK frequency higher or equal to the APB clock used by the SPI/I2S block. If this condition is not respected, the SPI/I2S will not work properly.

The audio sampling frequency may be 192 kHz, 96 kHz, 48 kHz, 44.1 kHz, 32 kHz, 22.05 kHz, 16 kHz, 11.025 kHz or 8 kHz (or any other value within this range).

In order to reach the desired frequency, the linear divider needs to be programmed according to the formulas below:

For I²S modes:

When the master clock is generated (MCKOE in the SPIx_I2SPR register is set):

$$F_S = \frac{F_{I2SxCLK}}{256 \times ((2 \times I2SDIV) + ODD)}$$

When the master clock is disabled (MCKOE bit cleared):

$$F_S = \frac{F_{I2SxCLK}}{32 \times (CHLEN + 1) \times ((2 \times I2SDIV) + ODD)}$$

CHLEN = 0 when the channel frame is 16-bit wide and,

CHLEN = 1 when the channel frame is 32-bit wide.

For PCM modes:

When the master clock is generated (MCKOE in the SPIx_I2SPR register is set):

$$F_S = \frac{F_{I2SxCLK}}{128 \times ((2 \times I2SDIV) + ODD)}$$

When the master clock is disabled (MCKOE bit cleared):

$$F_S = \frac{F_{I2SxCLK}}{16 \times (CHLEN + 1) \times ((2 \times I2SDIV) + ODD)}$$

CHLEN = 0 when the channel frame is 16-bit wide and,

CHLEN = 1 when the channel frame is 32-bit wide.

Where F_S is the audio sampling frequency, and $F_{I2SxCLK}$ is the frequency of the kernel clock provided to the SPI/I2S block.

Note: Note that I2SDIV must be strictly higher than 1.

[Table 172](#) provides example precision values for different clock configurations.

Note: Other configurations are possible that allow optimum clock precision.

Table 172. Audio-frequency precision using standard 8 MHz HSE⁽¹⁾

SYSCLK (MHz)	Data length	I2SDIV	I2SODD	MCLK	Target fs (Hz)	Real fs (kHz)	Error
48	16	8	0	No	96000	93750	2.3438%
48	32	4	0	No	96000	93750	2.3438%
48	16	15	1	No	48000	48387.0968	0.8065%
48	32	8	0	No	48000	46875	2.3438%
48	16	17	0	No	44100	44117.647	0.0400%
48	32	8	1	No	44100	44117.647	0.0400%
48	16	23	1	No	32000	31914.8936	0.2660%
48	32	11	1	No	32000	32608.696	1.9022%
48	16	34	0	No	22050	22058.8235	0.0400%
48	32	17	0	No	22050	22058.8235	0.0400%
48	16	47	0	No	16000	15957.4468	0.2660%
48	32	23	1	No	16000	15957.447	0.2660%
48	16	68	0	No	11025	11029.4118	0.0400%
48	32	34	0	No	11025	11029.412	0.0400%
48	16	94	0	No	8000	7978.7234	0.2660%
48	32	47	0	No	8000	7978.7234	0.2660%
48	16	2	0	Yes	48000	46875	2.3430%
48	32	2	0	Yes	48000	46875	2.3430%
48	16	2	0	Yes	44100	46875	6.2925%
48	32	2	0	Yes	44100	46875	6.2925%
48	16	3	0	Yes	32000	31250	2.3438%
48	32	3	0	Yes	32000	31250	2.3438%
48	16	4	1	Yes	22050	20833.333	5.5178%
48	32	4	1	Yes	22050	20833.333	5.5178%
48	16	6	0	Yes	16000	15625	2.3438%
48	32	6	0	Yes	16000	15625	2.3438%
48	16	8	1	Yes	11025	11029.4118	0.0400%
48	32	8	1	Yes	11025	11029.4118	0.0400%
48	16	11	1	Yes	8000	8152.17391	1.9022%
48	32	11	1	Yes	8000	8152.17391	1.9022%

1. This table gives only example values for different clock configurations. Other configurations allowing optimum clock precision are possible.

28.7.6 I²S master mode

The I²S can be configured in master mode. This means that the serial clock is generated on the CK pin as well as the Word Select signal WS. Master clock (MCK) may be output or not, controlled by the MCKOE bit in the SPIx_I2SPR register.

Procedure

1. Select the I2SDIV[7:0] bits in the SPIx_I2SPR register to define the serial clock baud rate to reach the proper audio sample frequency. The ODD bit in the SPIx_I2SPR register also has to be defined.
2. Select the CKPOL bit to define the steady level for the communication clock. Set the MCKOE bit in the SPIx_I2SPR register if the master clock MCK needs to be provided to the external DAC/ADC audio component (the I2SDIV and ODD values should be computed depending on the state of the MCK output, for more details refer to [Section 28.7.5: Clock generator](#)).
3. Set the I2SMOD bit in the SPIx_I2SCFGR register to activate the I²S functions and choose the I²S standard through the I2SSTD[1:0] and PCMSYNC bits, the data length through the DATLEN[1:0] bits and the number of bits per channel by configuring the CHLEN bit. Select also the I²S master mode and direction (Transmitter or Receiver) through the I2SCFG[1:0] bits in the SPIx_I2SCFGR register.
4. If needed, select all the potential interrupt sources and the DMA capabilities by writing the SPIx_CR2 register.
5. The I2SE bit in SPIx_I2SCFGR register must be set.

WS and CK are configured in output mode. MCK is also an output, if the MCKOE bit in SPIx_I2SPR is set.

Transmission sequence

The transmission sequence begins when a half-word is written into the Tx buffer.

Lets assume the first data written into the Tx buffer corresponds to the left channel data. When data are transferred from the Tx buffer to the shift register, TXE is set and data corresponding to the right channel have to be written into the Tx buffer. The CHSIDE flag indicates which channel is to be transmitted. It has a meaning when the TXE flag is set because the CHSIDE flag is updated when TXE goes high.

A full frame has to be considered as a left channel data transmission followed by a right channel data transmission. It is not possible to have a partial frame where only the left channel is sent.

The data half-word is parallel loaded into the 16-bit shift register during the first bit transmission, and then shifted out, serially, to the MOSI/SD pin, MSB first. The TXE flag is set after each transfer from the Tx buffer to the shift register and an interrupt is generated if the TXIE bit in the SPIx_CR2 register is set.

For more details about the write operations depending on the I²S standard mode selected, refer to [Section 28.7.3: Supported audio protocols](#)).

To ensure a continuous audio data transmission, it is mandatory to write the SPIx_DR register with the next data to transmit before the end of the current transmission.

To switch off the I²S, by clearing I2SE, it is mandatory to wait for TXE = 1 and BSY = 0.

Reception sequence

The operating mode is the same as for transmission mode except for the point 3 (refer to the procedure described in [Section 28.7.6: I²S master mode](#)), where the configuration should set the master reception mode through the I2SCFG[1:0] bits.

Whatever the data or channel length, the audio data are received by 16-bit packets. This means that each time the Rx buffer is full, the RXNE flag is set and an interrupt is generated if the RXNEIE bit is set in SPIx_CR2 register. Depending on the data and channel length configuration, the audio value received for a right or left channel may result from one or two receptions into the Rx buffer.

Clearing the RXNE bit is performed by reading the SPIx_DR register.

CHSIDE is updated after each reception. It is sensitive to the WS signal generated by the I²S cell.

For more details about the read operations depending on the I²S standard mode selected, refer to [Section 28.7.3: Supported audio protocols](#).

If data are received while the previously received data have not been read yet, an overrun is generated and the OVR flag is set. If the ERRIE bit is set in the SPIx_CR2 register, an interrupt is generated to indicate the error.

To switch off the I²S, specific actions are required to ensure that the I²S completes the transfer cycle properly without initiating a new data transfer. The sequence depends on the configuration of the data and channel lengths, and on the audio protocol mode selected. In the case of:

- 16-bit data length extended on 32-bit channel length (DATLEN = 00 and CHLEN = 1) using the LSB justified mode (I2SSTD = 10)
 - a) Wait for the second to last RXNE = 1 ($n - 1$)
 - b) Then wait 17 I²S clock cycles (using a software loop)
 - c) Disable the I²S (I2SE = 0)
- 16-bit data length extended on 32-bit channel length (DATLEN = 00 and CHLEN = 1) in MSB justified, I²S or PCM modes (I2SSTD = 00, I2SSTD = 01 or I2SSTD = 11, respectively)
 - a) Wait for the last RXNE
 - b) Then wait 1 I²S clock cycle (using a software loop)
 - c) Disable the I²S (I2SE = 0)
- For all other combinations of DATLEN and CHLEN, whatever the audio mode selected through the I2SSTD bits, carry out the following sequence to switch off the I²S:
 - a) Wait for the second to last RXNE = 1 ($n - 1$)
 - b) Then wait one I²S clock cycle (using a software loop)
 - c) Disable the I²S (I2SE = 0)

Note: The BSY flag is kept low during transfers.

28.7.7 I²S slave mode

For the slave configuration, the I²S can be configured in transmission or reception mode. The operating mode is following mainly the same rules as described for the I²S master

configuration. In slave mode, there is no clock to be generated by the I2S interface. The clock and WS signals are input from the external master connected to the I2S interface. There is then no need, for the user, to configure the clock.

The configuration steps to follow are listed below:

1. Set the I2SMOD bit in the SPIx_I2SCFGR register to select I²S mode and choose the I²S standard through the I2SSTD[1:0] bits, the data length through the DATLEN[1:0] bits and the number of bits per channel for the frame configuring the CHLEN bit. Select also the mode (transmission or reception) for the slave through the I2SCFG[1:0] bits in SPIx_I2SCFGR register.
2. If needed, select all the potential interrupt sources and the DMA capabilities by writing the SPIx_CR2 register.
3. The I2SE bit in SPIx_I2SCFGR register must be set.

Transmission sequence

The transmission sequence begins when the external master device sends the clock and when the NSS_WS signal requests the transfer of data. The slave has to be enabled before the external master starts the communication. The I2S data register has to be loaded before the master initiates the communication.

For the I2S, MSB justified and LSB justified modes, the first data item to be written into the data register corresponds to the data for the left channel. When the communication starts, the data are transferred from the Tx buffer to the shift register. The TXE flag is then set in order to request the right channel data to be written into the I2S data register.

The CHSIDE flag indicates which channel is to be transmitted. Compared to the master transmission mode, in slave mode, CHSIDE is sensitive to the WS signal coming from the external master. This means that the slave needs to be ready to transmit the first data before the clock is generated by the master. WS assertion corresponds to left channel transmitted first.

Note: *The I2SE has to be written at least two PCLK cycles before the first clock of the master comes on the CK line.*

The data half-word is parallel-loaded into the 16-bit shift register (from the internal bus) during the first bit transmission, and then shifted out serially to the MOSI/SD pin MSB first. The TXE flag is set after each transfer from the Tx buffer to the shift register and an interrupt is generated if the TXEIE bit in the SPIx_CR2 register is set.

Note that the TXE flag should be checked to be at 1 before attempting to write the Tx buffer.

For more details about the write operations depending on the I²S standard mode selected, refer to [Section 28.7.3: Supported audio protocols](#).

To secure a continuous audio data transmission, it is mandatory to write the SPIx_DR register with the next data to transmit before the end of the current transmission. An underrun flag is set and an interrupt may be generated if the data are not written into the SPIx_DR register before the first clock edge of the next data communication. This indicates to the software that the transferred data are wrong. If the ERRIE bit is set into the SPIx_CR2 register, an interrupt is generated when the UDR flag in the SPIx_SR register goes high. In this case, it is mandatory to switch off the I2S and to restart a data transfer starting from the left channel.

To switch off the I2S, by clearing the I2SE bit, it is mandatory to wait for TXE = 1 and BSY = 0.

Reception sequence

The operating mode is the same as for the transmission mode except for the point 1 (refer to the procedure described in [Section 28.7.7: I2S slave mode](#)), where the configuration should set the master reception mode using the I2SCFG[1:0] bits in the SPIx_I2SCFGR register.

Whatever the data length or the channel length, the audio data are received by 16-bit packets. This means that each time the RX buffer is full, the RXNE flag in the SPIx_SR register is set and an interrupt is generated if the RXNEIE bit is set in the SPIx_CR2 register. Depending on the data length and channel length configuration, the audio value received for a right or left channel may result from one or two receptions into the RX buffer.

The CHSIDE flag is updated each time data are received to be read from the SPIx_DR register. It is sensitive to the external WS line managed by the external master component.

Clearing the RXNE bit is performed by reading the SPIx_DR register.

For more details about the read operations depending the I²S standard mode selected, refer to [Section 28.7.3: Supported audio protocols](#).

If data are received while the preceding received data have not yet been read, an overrun is generated and the OVR flag is set. If the bit ERRIE is set in the SPIx_CR2 register, an interrupt is generated to indicate the error.

To switch off the I2S in reception mode, I2SE has to be cleared immediately after receiving the last RXNE = 1.

Note: *The external master components should have the capability of sending/receiving data in 16-bit or 32-bit packets via an audio channel.*

28.7.8 I2S status flags

Three status flags are provided for the application to fully monitor the state of the I2S bus.

Busy flag (BSY)

The BSY flag is set and cleared by hardware (writing to this flag has no effect). It indicates the state of the communication layer of the I2S.

When BSY is set, it indicates that the I2S is busy communicating. There is one exception in master receive mode (I2SCFG = 11) where the BSY flag is kept low during reception.

The BSY flag is useful to detect the end of a transfer if the software needs to disable the I2S. This avoids corrupting the last transfer. For this, the procedure described below must be strictly respected.

The BSY flag is set when a transfer starts, except when the I2S is in master receiver mode.

The BSY flag is cleared:

- When a transfer completes (except in master transmit mode, in which the communication is supposed to be continuous)
- When the I2S is disabled

When communication is continuous:

- In master transmit mode, the BSY flag is kept high during all the transfers
- In slave mode, the BSY flag goes low for one I2S clock cycle between each transfer

Note: *Do not use the BSY flag to handle each data transmission or reception. It is better to use the TXE and RXNE flags instead.*

Tx buffer empty flag (TXE)

When set, this flag indicates that the Tx buffer is empty and the next data to be transmitted can then be loaded into it. The TXE flag is reset when the Tx buffer already contains data to be transmitted. It is also reset when the I2S is disabled (I2SE bit is reset).

RX buffer not empty (RXNE)

When set, this flag indicates that there are valid received data in the RX Buffer. It is reset when SPIx_DR register is read.

Channel Side flag (CHSIDE)

In transmission mode, this flag is refreshed when TXE goes high. It indicates the channel side to which the data to transfer on SD has to belong. In case of an underrun error event in slave transmission mode, this flag is not reliable and I2S needs to be switched off and switched on before resuming the communication.

In reception mode, this flag is refreshed when data are received into SPIx_DR. It indicates from which channel side data have been received. Note that in case of error (like OVR) this flag becomes meaningless and the I2S should be reset by disabling and then enabling it (with configuration if it needs changing).

This flag has no meaning in the PCM standard (for both Short and Long frame modes).

When the OVR or UDR flag in the SPIx_SR is set and the ERRIE bit in SPIx_CR2 is also set, an interrupt is generated. This interrupt can be cleared by reading the SPIx_SR status register (once the interrupt source has been cleared).

28.7.9 I2S error flags

There are three error flags for the I2S cell.

Underrun flag (UDR)

In slave transmission mode this flag is set when the first clock for data transmission appears while the software has not yet loaded any value into SPIx_DR. It is available when the I2SMOD bit in the SPIx_I2SCFGR register is set. An interrupt may be generated if the ERRIE bit in the SPIx_CR2 register is set.

The UDR bit is cleared by a read operation on the SPIx_SR register.

Overrun flag (OVR)

This flag is set when data are received and the previous data have not yet been read from the SPIx_DR register. As a result, the incoming data are lost. An interrupt may be generated if the ERRIE bit is set in the SPIx_CR2 register.

In this case, the receive buffer contents are not updated with the newly received data from the transmitter device. A read operation to the SPIx_DR register returns the previous correctly received data. All other subsequently transmitted half-words are lost.

Clearing the OVR bit is done by a read operation on the SPIx_DR register followed by a read access to the SPIx_SR register.

Frame error flag (FRE)

This flag can be set by hardware only if the I2S is configured in Slave mode. It is set if the external master is changing the WS line while the slave is not expecting this change. If the

synchronization is lost, the following steps are required to recover from this state and resynchronize the external master device with the I2S slave device:

1. Disable the I2S.
2. Re-enable the I2S interface again (Keeping ASTRTEN=0).

Desynchronization between master and slave devices may be due to noisy environment on the CK communication clock or on the WS frame synchronization line. An error interrupt can be generated if the ERRIE bit is set. The desynchronization flag (FRE) is cleared by software when the status register is read.

28.7.10 DMA features

In I²S mode, the DMA works in exactly the same way as it does in SPI mode. There is no difference except that the CRC feature is not available in I²S mode since there is no data transfer protection system.

28.8 I2S interrupts

Table 173 provides the list of I2S interrupts.

Table 173. I2S interrupt requests

Interrupt event	Event flag	Enable control bit
Transmit buffer empty flag	TXE	TXEIE
Receive buffer not empty flag	RXNE	RXNEIE
Overrun error	OVR	ERRIE
Underrun error	UDR	
Frame error flag	FRE	

28.9 SPI and I2S registers

The peripheral registers can be accessed by half-words (16-bit) or words (32-bit). SPI_DR in addition can be accessed by 8-bit access.

28.9.1 SPI control register 1 (SPIx_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIDIMODE	BIDIOE	CRCE_N	CRCNEXT	CRCL	RXONLY	SSM	SSI	LSBFIRST	SPE	BR[2:0]	MSTR	CPOL	CPHA		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 **BIDIMODE**: Bidirectional data mode enable.

This bit enables half-duplex communication using common single bidirectional data line.
Keep RXONLY bit clear when bidirectional mode is active.

0: 2-line unidirectional data mode selected

1: 1-line bidirectional data mode selected

Note: This bit is not used in I²S mode.

Bit 14 **BIDIOE**: Output enable in bidirectional mode

This bit combined with the BIDIMODE bit selects the direction of transfer in bidirectional mode.

0: Output disabled (receive-only mode)

1: Output enabled (transmit-only mode)

Note: In master mode, the MOSI pin is used and in slave mode, the MISO pin is used.

This bit is not used in I²S mode.

Bit 13 **CRCEN**: Hardware CRC calculation enable

0: CRC calculation disabled

1: CRC calculation enabled

Note: This bit should be written only when SPI is disabled (SPE = '0') for correct operation.

This bit is not used in I²S mode.

Bit 12 **CRCNEXT**: Transmit CRC next

0: Next transmit value is from Tx buffer.

1: Next transmit value is from Tx CRC register.

Note: This bit has to be written as soon as the last data is written in the SPIx_DR register.

This bit is not used in I²S mode.

Bit 11 **CRCL**: CRC length

This bit is set and cleared by software to select the CRC length.

0: 8-bit CRC length

1: 16-bit CRC length

Note: This bit should be written only when SPI is disabled (SPE = '0') for correct operation.

This bit is not used in I²S mode.

Bit 10 **RXONLY:** Receive only mode enabled.

This bit enables simplex communication using a single unidirectional line to receive data exclusively. Keep BIDIMODE bit clear when receive only mode is active. This bit is also useful in a multislave system in which this particular slave is not accessed, the output from the accessed slave is not corrupted.

- 0: Full-duplex (Transmit and receive)
- 1: Output disabled (Receive-only mode)

Note: This bit is not used in I²S mode.

Bit 9 **SSM:** Software slave management

When the SSM bit is set, the NSS pin input is replaced with the value from the SSI bit.

- 0: Software slave management disabled
- 1: Software slave management enabled

Note: This bit is not used in I²S mode and SPI TI mode.

Bit 8 **SSI:** Internal slave select

This bit has an effect only when the SSM bit is set. The value of this bit is forced onto the NSS pin and the I/O value of the NSS pin is ignored.

Note: This bit is not used in I²S mode and SPI TI mode.

Bit 7 **LSBFIRST:** Frame format

- 0: data is transmitted / received with the MSB first
- 1: data is transmitted / received with the LSB first

Note: 1. This bit should not be changed when communication is ongoing.

2. This bit is not used in I²S mode and SPI TI mode.

Bit 6 **SPE:** SPI enable

- 0: Peripheral disabled
- 1: Peripheral enabled

Note: When disabling the SPI, follow the procedure described in [Procedure for disabling the SPI on page 963](#).

This bit is not used in I²S mode.

Bits 5:3 **BR[2:0]:** Baud rate control

- 000: f_{PCLK}/2
- 001: f_{PCLK}/4
- 010: f_{PCLK}/8
- 011: f_{PCLK}/16
- 100: f_{PCLK}/32
- 101: f_{PCLK}/64
- 110: f_{PCLK}/128
- 111: f_{PCLK}/256

Note: These bits should not be changed when communication is ongoing.

This bit is not used in I²S mode.

Bit 2 **MSTR:** Master selection

- 0: Slave configuration
- 1: Master configuration

Note: This bit should not be changed when communication is ongoing.

This bit is not used in I²S mode.

Bit 1 **CPOL:** Clock polarity

- 0: CK to 0 when idle
- 1: CK to 1 when idle

Note: This bit should not be changed when communication is ongoing.

This bit is not used in SPI TI mode except the case when CRC is applied at TI mode.

Bit 0 **CPHA:** Clock phase

- 0: The first clock transition is the first data capture edge
- 1: The second clock transition is the first data capture edge

Note: This bit should not be changed when communication is ongoing.

This bit is not used in SPI TI mode except the case when CRC is applied at TI mode.

28.9.2 SPI control register 2 (SPIx_CR2)

Address offset: 0x04

Reset value: 0x0700

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.		LDMA_TX	LDMA_RX	FRXTH	DS[3:0]				TXEIE	RXNEIE	ERRIE	FRF	NSSP	SSOE	TXDMAEN	RXDMAEN
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 Reserved, must be kept at reset value.

Bit 14 **LDMA_TX:** Last DMA transfer for transmission

This bit is used in data packing mode, to define if the total number of data to transmit by DMA is odd or even. It has significance only if the TXDMAEN bit in the SPIx_CR2 register is set and if packing mode is used (data length <= 8-bit and write access to SPIx_DR is 16-bit wide). It has to be written when the SPI is disabled (SPE = 0 in the SPIx_CR1 register).

- 0: Number of data to transfer is even
- 1: Number of data to transfer is odd

Note: Refer to [Procedure for disabling the SPI on page 963](#) if the CRCEN bit is set.

This bit is not used in I²S mode.

Bit 13 **LDMA_RX:** Last DMA transfer for reception

This bit is used in data packing mode, to define if the total number of data to receive by DMA is odd or even. It has significance only if the RXDMAEN bit in the SPIx_CR2 register is set and if packing mode is used (data length <= 8-bit and write access to SPIx_DR is 16-bit wide). It has to be written when the SPI is disabled (SPE = 0 in the SPIx_CR1 register).

- 0: Number of data to transfer is even
- 1: Number of data to transfer is odd

Note: Refer to [Procedure for disabling the SPI on page 963](#) if the CRCEN bit is set.

This bit is not used in I²S mode.

Bit 12 **FRXTH:** FIFO reception threshold

This bit is used to set the threshold of the RXFIFO that triggers an RXNE event

- 0: RXNE event is generated if the FIFO level is greater than or equal to 1/2 (16-bit)
- 1: RXNE event is generated if the FIFO level is greater than or equal to 1/4 (8-bit)

Note: This bit is not used in I²S mode.

Bits 11:8 **DS[3:0]**: Data size

These bits configure the data length for SPI transfers.

- 0000: Not used
- 0001: Not used
- 0010: Not used
- 0011: 4-bit
- 0100: 5-bit
- 0101: 6-bit
- 0110: 7-bit
- 0111: 8-bit
- 1000: 9-bit
- 1001: 10-bit
- 1010: 11-bit
- 1011: 12-bit
- 1100: 13-bit
- 1101: 14-bit
- 1110: 15-bit
- 1111: 16-bit

If software attempts to write one of the “Not used” values, they are forced to the value “0111” (8-bit)

Note: This bit is not used in I²S mode.

Bit 7 **TXEIE**: Tx buffer empty interrupt enable

0: TXE interrupt masked

1: TXE interrupt not masked. Used to generate an interrupt request when the TXE flag is set.

Bit 6 **RXNEIE**: RX buffer not empty interrupt enable

0: RXNE interrupt masked

1: RXNE interrupt not masked. Used to generate an interrupt request when the RXNE flag is set.

Bit 5 **ERRIE**: Error interrupt enable

This bit controls the generation of an interrupt when an error condition occurs (CRCERR, OVR, MODF in SPI mode, FRE at TI mode and UDR, OVR, and FRE in I²S mode).

0: Error interrupt is masked

1: Error interrupt is enabled

Bit 4 **FRF**: Frame format

0: SPI Motorola mode

1 SPI TI mode

Note: This bit must be written only when the SPI is disabled (SPE=0).

This bit is not used in I²S mode.

Bit 3 **NSSP**: NSS pulse management

This bit is used in master mode only. It allows the SPI to generate an NSS pulse between two consecutive data when doing continuous transfers. In the case of a single data transfer, it forces the NSS pin high level after the transfer.

It has no meaning if CPHA = '1', or FRF = '1'.

0: No NSS pulse

1: NSS pulse generated

Note: 1. This bit must be written only when the SPI is disabled (SPE=0).

2. This bit is not used in I²S mode and SPI TI mode.

Bit 2 **SSOE:** SS output enable

0: SS output is disabled in master mode and the SPI interface can work in multimaster configuration

1: SS output is enabled in master mode and when the SPI interface is enabled. The SPI interface cannot work in a multimaster environment.

Note: This bit is not used in I²S mode and SPI TI mode.

Bit 1 **TXDMAEN:** Tx buffer DMA enable

When this bit is set, a DMA request is generated whenever the TXE flag is set.

0: Tx buffer DMA disabled

1: Tx buffer DMA enabled

Bit 0 **RXDMAEN:** Rx buffer DMA enable

When this bit is set, a DMA request is generated whenever the RXNE flag is set.

0: Rx buffer DMA disabled

1: Rx buffer DMA enabled

28.9.3 SPI status register (SPIx_SR)

Address offset: 0x08

Reset value: 0x0002

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	FTLVL[1:0]		FRLVL[1:0]		FRE	BSY	OVR	MODF	CRCE RR	UDR	CHSIDE	TXE	RXNE
			r	r	r	r	r	r	r	r	rc_w0	r	r	r	r

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

Bits 12:11 **FTLVL[1:0]:** FIFO transmission level

These bits are set and cleared by hardware.

00: FIFO empty

01: 1/4 FIFO

10: 1/2 FIFO

11: FIFO full (considered as FULL when the FIFO threshold is greater than 1/2)

Note: This bit is not used in I²S mode.

Bits 10:9 **FRLVL[1:0]:** FIFO reception level

These bits are set and cleared by hardware.

00: FIFO empty

01: 1/4 FIFO

10: 1/2 FIFO

11: FIFO full

Note: These bits are not used in I²S mode and in SPI receive-only mode while CRC calculation is enabled.

Bit 8 **FRE:** Frame format error

This flag is used for SPI in TI slave mode and I²S slave mode. Refer to [Section 28.5.11: SPI error flags](#) and [Section 28.7.9: I2S error flags](#).

This flag is set by hardware and reset when SPIx_SR is read by software.

0: No frame format error

1: A frame format error occurred

Bit 7 BSY: Busy flag

- 0: SPI (or I2S) not busy
 - 1: SPI (or I2S) is busy in communication or Tx buffer is not empty
- This flag is set and cleared by hardware.

Note: The BSY flag must be used with caution: refer to [Section 28.5.10: SPI status flags and Procedure for disabling the SPI](#) on page 963.

Bit 6 OVR: Overrun flag

- 0: No overrun occurred
 - 1: Overrun occurred
- This flag is set by hardware and reset by a software sequence. Refer to [I2S error flags on page 996](#) for the software sequence.

Bit 5 MODF: Mode fault

- 0: No mode fault occurred
- 1: Mode fault occurred

This flag is set by hardware and reset by a software sequence. Refer to [Section : Mode fault \(MODF\) on page 973](#) for the software sequence.

Note: This bit is not used in I²S mode.

Bit 4 CRCERR: CRC error flag

- 0: CRC value received matches the SPIx_RXCRCR value
- 1: CRC value received does not match the SPIx_RXCRCR value

Note: This flag is set by hardware and cleared by software writing 0.

This bit is not used in I²S mode.

Bit 3 UDR: Underrun flag

- 0: No underrun occurred
- 1: Underrun occurred

This flag is set by hardware and reset by a software sequence. Refer to [I2S error flags on page 996](#) for the software sequence.

Note: This bit is not used in SPI mode.

Bit 2 CHSIDE: Channel side

- 0: Channel Left has to be transmitted or has been received
- 1: Channel Right has to be transmitted or has been received

Note: This bit is not used in SPI mode. It has no significance in PCM mode.

Bit 1 TXE: Transmit buffer empty

- 0: Tx buffer not empty
- 1: Tx buffer empty

Bit 0 RXNE: Receive buffer not empty

- 0: Rx buffer empty
- 1: Rx buffer not empty

28.9.4 SPI data register (SPIx_DR)

Address offset: 0x0C

Reset value: 0x0000

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

Bits 15:0 **DR[15:0]**: Data register

Data received or to be transmitted

The data register serves as an interface between the Rx and Tx FIFOs. When the data register is read, RxFIFO is accessed while the write to data register accesses TxFIFO (See [Section 28.5.9: Data transmission and reception procedures](#)).

Note: Data is always right-aligned. Unused bits are ignored when writing to the register, and read as zero when the register is read. The Rx threshold setting must always correspond with the read access currently used.

28.9.5 SPI CRC polynomial register (SPIx_CRCPR)

Address offset: 0x10

Reset value: 0x0007

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

Bits 15:0 **CRCPOLY[15:0]**: CRC polynomial register

This register contains the polynomial for the CRC calculation.

The CRC polynomial (0x0007) is the reset value of this register. Another polynomial can be configured as required.

Note: The polynomial value should be odd only. No even value is supported.

28.9.6 SPI Rx CRC register (SPIx_RXCRCR)

Address offset: 0x14

Reset value: 0x0000

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

Bits 15:0 RXCRC[15:0]: Rx CRC register

When CRC calculation is enabled, the RXCRC[15:0] bits contain the computed CRC value of the subsequently received bytes. This register is reset when the CRCEN bit in SPIx_CR1 register is written to 1. The CRC is calculated serially using the polynomial programmed in the SPIx_CRCPR register.

Only the 8 LSB bits are considered when the CRC frame format is set to be 8-bit length (CRCL bit in the SPIx_CR1 is cleared). CRC calculation is done based on any CRC8 standard.

The entire 16-bits of this register are considered when a 16-bit CRC frame format is selected (CRCL bit in the SPIx_CR1 register is set). CRC calculation is done based on any CRC16 standard.

Note: A read to this register when the BSY Flag is set could return an incorrect value.

These bits are not used in I²S mode.

28.9.7 SPI Tx CRC register (SPIx_TXCRCR)

Address offset: 0x18

Reset value: 0x0000

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

Bits 15:0 TXCRC[15:0]: Tx CRC register

When CRC calculation is enabled, the TXCRC[7:0] bits contain the computed CRC value of the subsequently transmitted bytes. This register is reset when the CRCEN bit of SPIx_CR1 is written to 1. The CRC is calculated serially using the polynomial programmed in the SPIx_CRCPR register.

Only the 8 LSB bits are considered when the CRC frame format is set to be 8-bit length (CRCL bit in the SPIx_CR1 is cleared). CRC calculation is done based on any CRC8 standard.

The entire 16-bits of this register are considered when a 16-bit CRC frame format is selected (CRCL bit in the SPIx_CR1 register is set). CRC calculation is done based on any CRC16 standard.

Note: A read to this register when the BSY flag is set could return an incorrect value.

These bits are not used in I²S mode.

28.9.8 SPIx_I2S configuration register (SPIx_I2SCFGR)

Address offset: 0x1C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	ASTR TEN	I2SMOD	I2SE	I2SCFG[1:0]		PCMSYNC	Res.	I2SSSTD[1:0]		CKPOL	DATLEN[1:0]		CHLEN
			rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw

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

Bit 12 **ASTRTEN**: Asynchronous start enable.

0: The Asynchronous start is disabled.

When the I2S is enabled in slave mode, the hardware starts the transfer when the I2S clock is received and an appropriate transition is detected on the WS signal.

1: The Asynchronous start is enabled.

When the I2S is enabled in slave mode, the hardware starts the transfer when the I2S clock is received and the appropriate level is detected on the WS signal.

Note: The appropriate transition is a falling edge on WS signal when I²S Philips Standard is used, or a rising edge for other standards.

The appropriate level is a low level on WS signal when I²S Philips Standard is used, or a high level for other standards.

Please refer to [Section 28.7.4: Start-up description](#) for additional information.

Bit 11 **I2SMOD**: I2S mode selection

0: SPI mode is selected

1: I2S mode is selected

Note: This bit should be configured when the SPI is disabled.

Bit 10 **I2SE**: I2S enable

0: I2S peripheral is disabled

1: I2S peripheral is enabled

Note: This bit is not used in SPI mode.

Bits 9:8 **I2SCFG[1:0]**: I2S configuration mode

00: Slave - transmit

01: Slave - receive

10: Master - transmit

11: Master - receive

Note: These bits should be configured when the I2S is disabled.

They are not used in SPI mode.

Bit 7 **PCMSYNC**: PCM frame synchronization

0: Short frame synchronization

1: Long frame synchronization

Note: This bit has a meaning only if I2SSTD = 11 (PCM standard is used).

It is not used in SPI mode.

Bit 6 Reserved, must be kept at reset value.

Bits 5:4 **I2SSTD[1:0]**: I2S standard selection

00: I²S Philips standard

01: MSB justified standard (left justified)

10: LSB justified standard (right justified)

11: PCM standard

For more details on I²S standards, refer to [Section 28.7.3 on page 980](#)

Note: For correct operation, these bits should be configured when the I2S is disabled.

They are not used in SPI mode.

Bit 3 **CKPOL**: Inactive state clock polarity

- 0: I2S clock inactive state is low level
- 1: I2S clock inactive state is high level

Note: For correct operation, this bit should be configured when the I2S is disabled.

It is not used in SPI mode.

The bit CKPOL does not affect the CK edge sensitivity used to receive or transmit the SD and WS signals.

Bits 2:1 **DATLEN[1:0]**: Data length to be transferred

- 00: 16-bit data length
- 01: 24-bit data length
- 10: 32-bit data length
- 11: Not allowed

Note: For correct operation, these bits should be configured when the I2S is disabled.

They are not used in SPI mode.

Bit 0 **CHLEN**: Channel length (number of bits per audio channel)

- 0: 16-bit wide
- 1: 32-bit wide

The bit write operation has a meaning only if DATLEN = 00 otherwise the channel length is fixed to 32-bit by hardware whatever the value filled in.

Note: For correct operation, this bit should be configured when the I2S is disabled.

It is not used in SPI mode.

28.9.9 SPIx_I2S prescaler register (SPIx_I2SPR)

Address offset: 0x20

Reset value: 0x0002

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	MCKOE	ODD	I2SDIV[7:0]							

Bits 15:10 Reserved, must be kept at reset value.

Bit 9 **MCKOE**: Master clock output enable

- 0: Master clock output is disabled
- 1: Master clock output is enabled

Note: This bit should be configured when the I2S is disabled. It is used only when the I2S is in master mode.

It is not used in SPI mode.

Bit 8 **ODD**: Odd factor for the prescaler

- 0: Real divider value is = I2SDIV *2
 - 1: Real divider value is = (I2SDIV * 2) + 1
- Refer to [Section 28.7.4 on page 987](#).

Note: This bit should be configured when the I2S is disabled. It is used only when the I2S is in master mode.

It is not used in SPI mode.

Bits 7:0 **I2SDIV[7:0]**: I2S linear prescaler

I2SDIV [7:0] = 0 or I2SDIV [7:0] = 1 are forbidden values.

Refer to [Section 28.7.4 on page 987](#).

Note: These bits should be configured when the I2S is disabled. They are used only when the I2S is in master mode.

They are not used in SPI mode.

28.9.10 SPI/I2S register map

Table 174 shows the SPI/I2S register map and reset values.

Table 174. SPI register map and reset values

Offset	Register	Reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7
0x00	SPIx_CR1		Res.																								
	Reset value		Res.																								
0x04	SPIx_CR2		Res.																								
	Reset value		Res.																								
0x08	SPIx_SR		Res.																								
	Reset value		Res.																								
0x0C	SPIx_DR		Res.																								
	Reset value		Res.																								
0x10	SPIx_CRCPR		Res.																								
	Reset value		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	SPIx_RXCRCR		Res.																								
	Reset value		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	SPIx_TXCRCR		Res.																								
	Reset value		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1C	SPIx_I2SCFGR		Res.																								
	Reset value		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x20	SPIx_I2SPR		Res.																								
	Reset value		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Refer to [Section 2.2.2](#) for the register boundary addresses.

29 Serial audio interface (SAI)

29.1 Introduction

The SAI interface (Serial Audio Interface) offers a wide set of audio protocols due to its flexibility and wide range of configurations. Many stereo or mono audio applications may be targeted. I2S standards, LSB or MSB-justified, PCM/DSP, TDM, and AC'97 protocols may be addressed for example. SPDIF output is offered when the audio block is configured as a transmitter.

To bring this level of flexibility and reconfigurability, the SAI contains two independent audio sub-blocks. Each block has its own clock generator and I/O line controller.

The SAI can work in master or slave configuration. The audio sub-blocks can be either receiver or transmitter and can work synchronously or not (with respect to the other one).

The SAI can be connected with other SAIs to work synchronously.

29.2 SAI main features

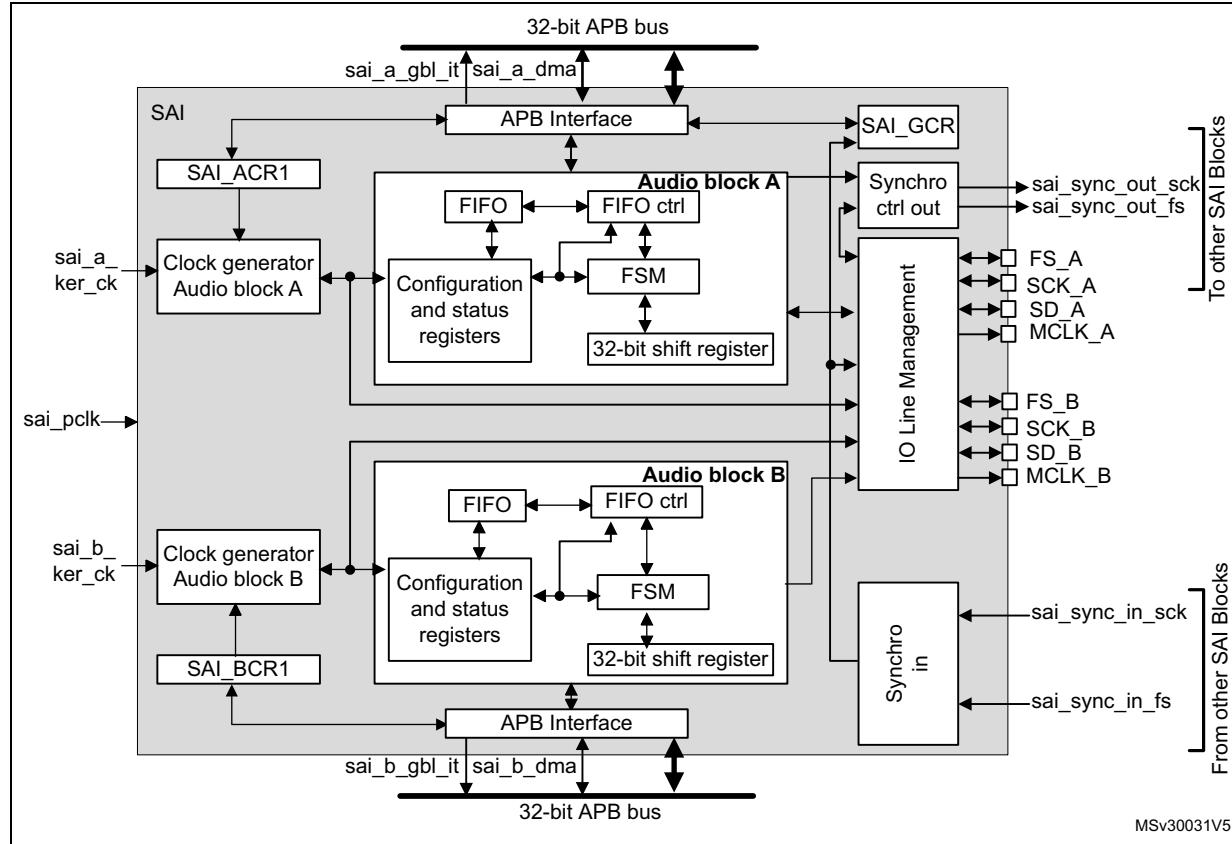
- Two independent audio sub-blocks which can be transmitters or receivers with their respective FIFO.
- 8-word integrated FIFOs for each audio sub-block.
- Synchronous or asynchronous mode between the audio sub-blocks.
- Possible synchronization between multiple SAIs.
- Master or slave configuration independent for both audio sub-blocks.
- Clock generator for each audio block to target independent audio frequency sampling when both audio sub-blocks are configured in master mode.
- Data size configurable: 8-, 10-, 16-, 20-, 24-, 32-bit.
- Audio protocol: I2S, LSB or MSB-justified, PCM/DSP, TDM, AC'97
- SPDIF output available if required.
- Up to 16 slots available with configurable size.
- Number of bits by frame can be configurable.
- Frame synchronization active level configurable (offset, bit length, level).
- First active bit position in the slot is configurable.
- LSB first or MSB first for data transfer.
- Mute mode.
- Stereo/Mono audio frame capability.
- Communication clock strobing edge configurable (SCK).
- Error flags with associated interrupts if enabled respectively.
 - Overrun and underrun detection,
 - Anticipated frame synchronization signal detection in slave mode,
 - Late frame synchronization signal detection in slave mode,
 - Codec not ready for the AC'97 mode in reception.
- Interruption sources when enabled:
 - Errors,
 - FIFO requests.
- 2-channel DMA interface.

29.3 SAI functional description

29.3.1 SAI block diagram

Figure 380 shows the SAI block diagram while *Table 175* and *Table 176* list SAI internal and external signals.

Figure 380. SAI functional block diagram



The SAI is mainly composed of two audio sub-blocks with their own clock generator. Each audio block integrates a 32-bit shift register controlled by their own functional state machine. Data are stored or read from the dedicated FIFO. FIFO may be accessed by the CPU, or by DMA in order to leave the CPU free during the communication. Each audio block is independent. They can be synchronous with each other.

An I/O line controller manages a set of 4 dedicated pins (SD, SCK, FS, MCLK) for a given audio block in the SAI. Some of these pins can be shared if the two sub-blocks are declared as synchronous to leave some free to be used as general purpose I/Os. The MCLK pin can be output, or not, depending on the application, the decoder requirement and whether the audio block is configured as the master.

If one SAI is configured to operate synchronously with another one, even more I/Os can be freed (except for pins SD_x).

The functional state machine can be configured to address a wide range of audio protocols. Some registers are present to set-up the desired protocols (audio frame waveform generator).

The audio sub-block can be a transmitter or receiver, in master or slave mode. The master mode means the SCK_x bit clock and the frame synchronization signal are generated from the SAI, whereas in slave mode, they come from another external or internal master. There is a particular case for which the FS signal direction is not directly linked to the master or slave mode definition. In AC'97 protocol, it will be an SAI output even if the SAI (link controller) is set-up to consume the SCK clock (and so to be in Slave mode).

Note: *For ease of reading of this section, the notation SAI_x refers to SAI_A or SAI_B, where 'x' represents the SAI A or B sub-block.*

29.3.2 SAI pins and internal signals

Table 175. SAI internal input/output signals

Internal signal name	Signal type	Description
sai_a_gbl_it/ sai_b_gbl_it	Output	Audio block A and B global interrupts.
sai_a_dma, sai_b_dma	Input/output	Audio block A and B DMA acknowledges and requests.
sai_sync_out_sck, sai_sync_out_fs	Output	Internal clock and frame synchronization output signals exchanged with other SAI blocks.
sai_sync_in_sck, sai_sync_in_fs	Input	Internal clock and frame synchronization input signals exchanged with other SAI blocks.
sai_a_ker_ck/ sai_b_ker_ck	Input	Audio block A/B kernel clock.
sai_pclk	Input	APB clock.

Table 176. SAI input/output pins

Name	Signal type	Comments
SAI_SCK_A/B	Input/output	Audio block A/B bit clock.
SAI_MCLK_A/B	Output	Audio block A/B master clock.
SAI_SD_A/B	Input/output	Data line for block A/B.
SAI_FS_A/B	Input/output	Frame synchronization line for audio block A/B.

29.3.3 Main SAI modes

Each audio sub-block of the SAI can be configured to be master or slave via MODE bits in the SAI_xCR1 register of the selected audio block.

Master mode

In master mode, the SAI delivers the timing signals to the external connected device:

- The bit clock and the frame synchronization are output on pin SCK_x and FS_x, respectively.
- If needed, the SAI can also generate a master clock on MCLK_x pin.

Both SCK_x, FS_x and MCLK_x are configured as outputs.

Slave mode

The SAI expects to receive timing signals from an external device.

- If the SAI sub-block is configured in asynchronous mode, then SCK_x and FS_x pins are configured as inputs.
- If the SAI sub-block is configured to operate synchronously with another SAI interface or with the second audio sub-block, the corresponding SCK_x and FS_x pins are left free to be used as general purpose I/Os.

In slave mode, MCLK_x pin is not used and can be assigned to another function.

It is recommended to enable the slave device before enabling the master.

Configuring and enabling SAI modes

Each audio sub-block can be independently defined as a transmitter or receiver through the MODE bit in the SAI_xCR1 register of the corresponding audio block. As a result, SAI_SD_x pin will be respectively configured as an output or an input.

Two master audio blocks in the same SAI can be configured with two different MCLK and SCK clock frequencies. In this case they have to be configured in asynchronous mode.

Each of the audio blocks in the SAI are enabled by SAIEN bit in the SAI_xCR1 register. As soon as this bit is active, the transmitter or the receiver is sensitive to the activity on the clock line, data line and synchronization line in slave mode.

In master TX mode, enabling the audio block immediately generates the bit clock for the external slaves even if there is no data in the FIFO. However FS signal generation is conditioned by the presence of data in the FIFO. After the FIFO receives the first data to transmit, this data is output to external slaves. If there is no data to transmit in the FIFO, 0 values are then sent in the audio frame with an underrun flag generation.

In slave mode, the audio frame starts when the audio block is enabled and when a start of frame is detected.

In Slave TX mode, no underrun event is possible on the first frame after the audio block is enabled, because the mandatory operating sequence in this case is:

1. Write into the SAI_xDR (by software or by DMA).
2. Wait until the FIFO threshold (FLH) flag is different from 0b000 (FIFO empty).
3. Enable the audio block in slave transmitter mode.

29.3.4 SAI synchronization mode

There are two levels of synchronization, either at audio sub-block level or at SAI level.

Internal synchronization

An audio sub-block can be configured to operate synchronously with the second audio sub-block in the same SAI. In this case, the bit clock and the frame synchronization signals are shared to reduce the number of external pins used for the communication. The audio block configured in synchronous mode sees its own SCK_x, FS_x, and MCLK_x pins released back as GPIOs while the audio block configured in asynchronous mode is the one for which FS_x and SCK_x ad MCLK_x I/O pins are relevant (if the audio block is considered as master).

Typically, the audio block in synchronous mode can be used to configure the SAI in full duplex mode. One of the two audio blocks can be configured as a master and the other as slave, or both as slaves with one asynchronous block (corresponding SYNCEN[1:0] bits set to 00 in SAI_xCR1) and one synchronous block (corresponding SYNCEN[1:0] bits set to 01 in the SAI_xCR1).

Note: Due to internal resynchronization stages, PCLK APB frequency must be higher than twice the bit rate clock frequency.

External synchronization

The audio sub-blocks can also be configured to operate synchronously with another SAI. This can be done as follow:

1. The SAI, which is configured as the source from which the other SAI is synchronized, has to define which of its audio sub-block is supposed to provide the FS and SCK signals to other SAI. This is done by programming SYNCOUT[1:0] bits.
2. The SAI which shall receive the synchronization signals has to select which SAI will provide the synchronization by setting the proper value on SYNCIN[1:0] bits. For each of the two SAI audio sub-blocks, the user must then specify if it operates synchronously with the other SAI via the SYNCEN bit.

Note: SYNCIN[1:0] and SYNCOUT[1:0] bits are located into the SAI_GCR register, and SYNCEN bits into SAI_xCR1 register.

If both audio sub-blocks in a given SAI need to be synchronized with another SAI, it is possible to choose one of the following configurations:

- Configure each audio block to be synchronous with another SAI block through the SYNCEN[1:0] bits.
- Configure one audio block to be synchronous with another SAI through the SYNCEN[1:0] bits. The other audio block is then configured as synchronous with the second SAI audio block through SYNCEN[1:0] bits.

The following table shows how to select the proper synchronization signal depending on the SAI block used. For example SAI2 can select the synchronization from SAI1 by setting SAI2 SYNCIN to 0. If SAI1 wants to select the synchronization coming from SAI2, SAI1 SYNCIN must be set to 1. Positions noted as 'Res.' shall not be used.

Table 177. External synchronization selection

Block instance	SYNCIN= 3	SYNCIN= 2	SYNCIN= 1	SYNCIN= 0
SAI1	Res.	Res.	SAI2 sync	Res.
SAI2	Res.	Res.	Res.	SAI1 sync

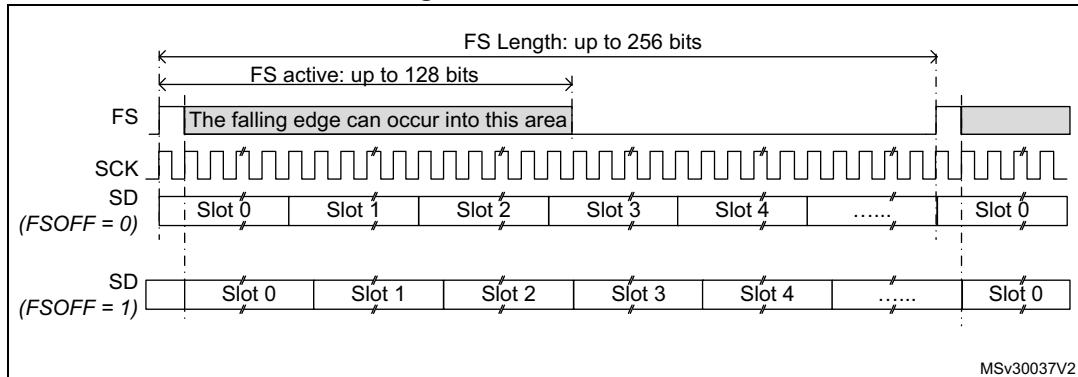
29.3.5 Audio data size

The audio frame can target different data sizes by configuring bit DS[2:0] in the SAI_xCR1 register. The data sizes may be 8, 10, 16, 20, 24 or 32 bits. During the transfer, either the MSB or the LSB of the data are sent first, depending on the configuration of bit LSBFIRST in the SAI_xCR1 register.

29.3.6 Frame synchronization

The FS signal acts as the Frame synchronization signal in the audio frame (start of frame). The shape of this signal is completely configurable in order to target the different audio protocols with their own specificities concerning this Frame synchronization behavior. This reconfigurability is done using register SAI_xFRCR. [Figure 381](#) illustrates this flexibility.

Figure 381. Audio frame



In AC'97 mode or in SPDIF mode (bit PRTCFG[1:0] = 10 or PRTCFG[1:0] = 01 in the SAI_xCR1 register), the frame synchronization shape is forced to match the AC'97 protocol. The SAI_xFRCR register value is ignored.

Each audio block is independent and consequently each one requires a specific configuration.

Frame length

- Master mode

The audio frame length can be configured to up to 256 bit clock cycles, by setting FRL[7:0] field in the SAI_xFRCR register.

If the frame length is greater than the number of declared slots for the frame, the remaining bits to transmit will be extended to 0 or the SD line will be released to HI-z depending the state of bit TRIS in the SAI_xCR2 register (refer to [FS signal role](#)). In reception mode, the remaining bit is ignored.

If bit NODIV is cleared, (FRL+1) must be equal to a power of 2, from 8 to 256, to ensure that an audio frame contains an integer number of MCLK pulses per bit clock cycle.

If bit NODIV is set, the (FRL+1) field can take any value from 8 to 256. Refer to [Section 29.3.8: SAI clock generator](#).

- Slave mode

The audio frame length is mainly used to specify to the slave the number of bit clock cycles per audio frame sent by the external master. It is used mainly to detect from the master any anticipated or late occurrence of the Frame synchronization signal during an on-going audio frame. In this case an error will be generated. For more details refer to [Section 29.3.13: Error flags](#).

In slave mode, there are no constraints on the FRL[7:0] configuration in the SAI_xFRCR register.

The number of bits in the frame is equal to FRL[7:0] + 1.

The minimum number of bits to transfer in an audio frame is 8.

Frame synchronization polarity

FSPOL bit in the SAI_xFRCR register sets the active polarity of the FS pin from which a frame is started. The start of frame is edge sensitive.

In slave mode, the audio block waits for a valid frame to start transmitting or receiving. Start of frame is synchronized to this signal. It is effective only if the start of frame is not detected during an ongoing communication and assimilated to an anticipated start of frame (refer to [Section 29.3.13: Error flags](#)).

In master mode, the frame synchronization is sent continuously each time an audio frame is complete until the SAIEN bit in the SAI_xCR1 register is cleared. If no data are present in the FIFO at the end of the previous audio frame, an underrun condition will be managed as described in [Section 29.3.13: Error flags](#), but the audio communication flow will not be interrupted.

Frame synchronization active level length

The FSALL[6:0] bits of the SAI_xFRCR register allow configuring the length of the active level of the Frame synchronization signal. The length can be set from 1 to 128 bit clock cycles.

As an example, the active length can be half of the frame length in I2S, LSB or MSB-justified modes, or one-bit wide for PCM/DSP or TDM mode.

Frame synchronization offset

Depending on the audio protocol targeted in the application, the Frame synchronization signal can be asserted when transmitting the last bit or the first bit of the audio frame (this is the case in I2S standard protocol and in MSB-justified protocol, respectively). FSOFF bit in the SAI_xFRCR register allows to choose one of the two configurations.

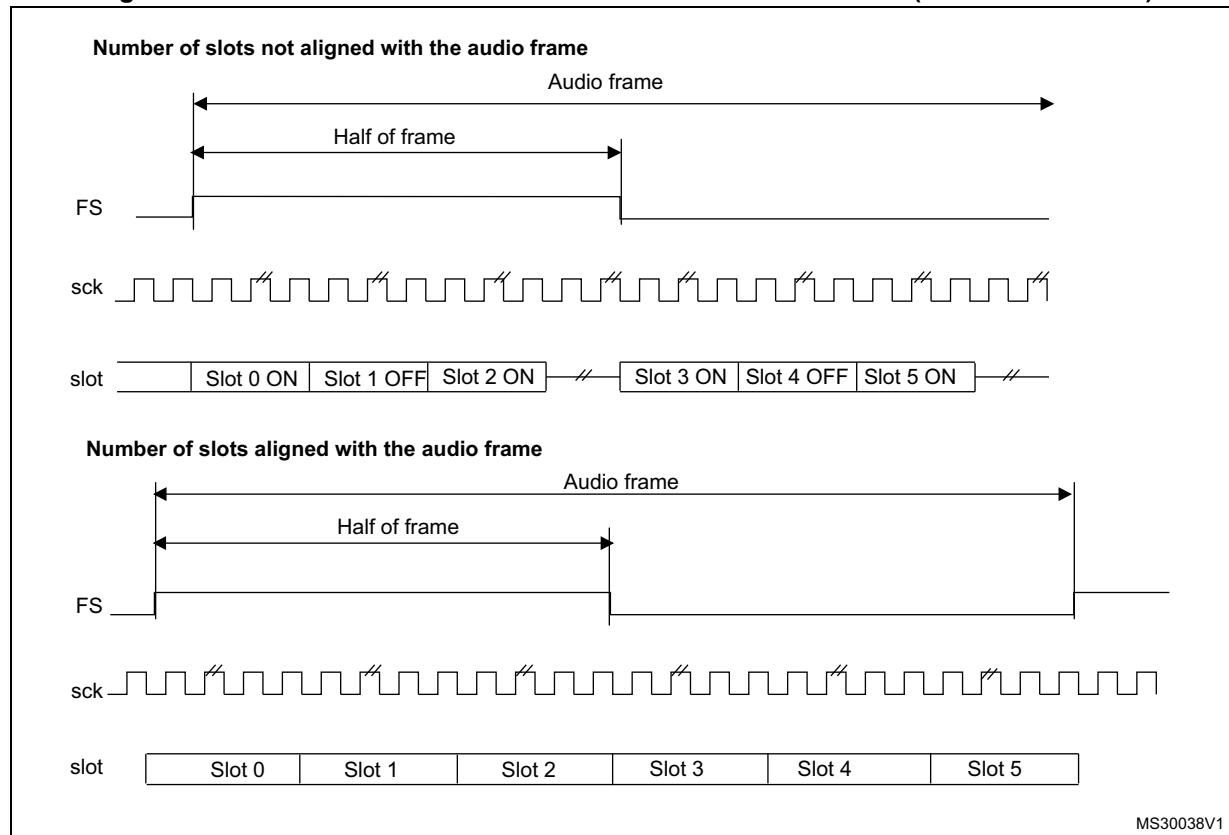
FS signal role

The FS signal can have a different meaning depending on the FS function. FSDEF bit in the SAI_xFRCR register selects which meaning it will have:

- 0: start of frame, like for instance the PCM/DSP, TDM, AC'97, audio protocols,
- 1: start of frame and channel side identification within the audio frame like for the I2S, the MSB or LSB-justified protocols.

When the FS signal is considered as a start of frame and channel side identification within the frame, the number of declared slots must be considered to be half the number for the left channel and half the number for the right channel. If the number of bit clock cycles on half audio frame is greater than the number of slots dedicated to a channel side, and TRIS = 0, 0 is sent for transmission for the remaining bit clock cycles in the SAI_xCR2 register.

Otherwise if TRIS = 1, the SD line is released to HI-Z. In reception mode, the remaining bit clock cycles are not considered until the channel side changes.

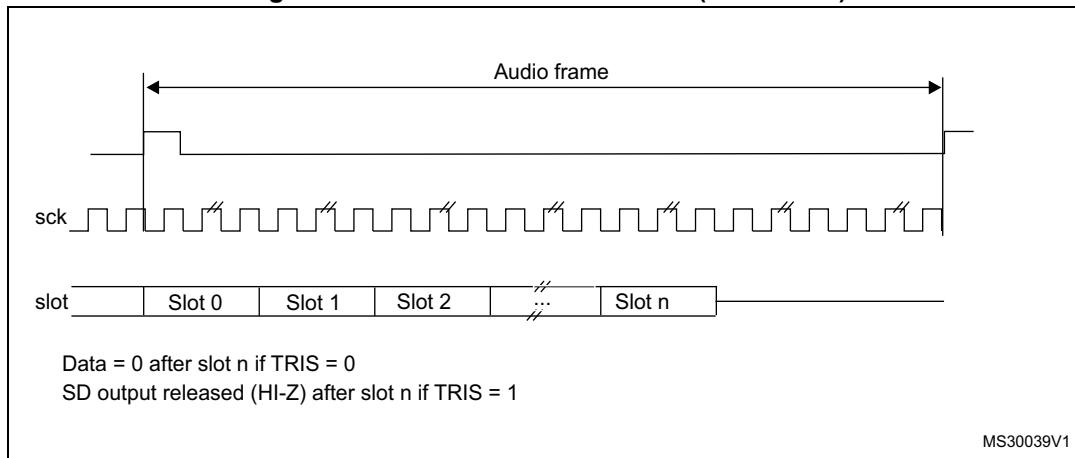
Figure 382. FS role is start of frame + channel side identification (FSDEF = TRIS = 1)

MS30038V1

1. The frame length should be even.

If FSDEF bit in SAI_xFRCR is kept clear, so FS signal is equivalent to a start of frame, and if the number of slots defined in NBSLOT[3:0] in SAI_xSLOTR multiplied by the number of bits by slot configured in SLOTSZ[1:0] in SAI_xSLOTR is less than the frame size (bit FRL[7:0] in the SAI_xFRCR register), then:

- if TRIS = 0 in the SAI_xCR2 register, the remaining bit after the last slot will be forced to 0 until the end of frame in case of transmitter,
- if TRIS = 1, the line will be released to HI-Z during the transfer of these remaining bits. In reception mode, these bits are discarded.

Figure 383. FS role is start of frame (FSDEF = 0)

The FS signal is not used when the audio block in transmitter mode is configured to get the SPDIF output on the SD line. The corresponding FS I/O will be released and left free for other purposes.

29.3.7 Slot configuration

The slot is the basic element in the audio frame. The number of slots in the audio frame is equal to NBSLOT[3:0] + 1.

The maximum number of slots per audio frame is fixed at 16.

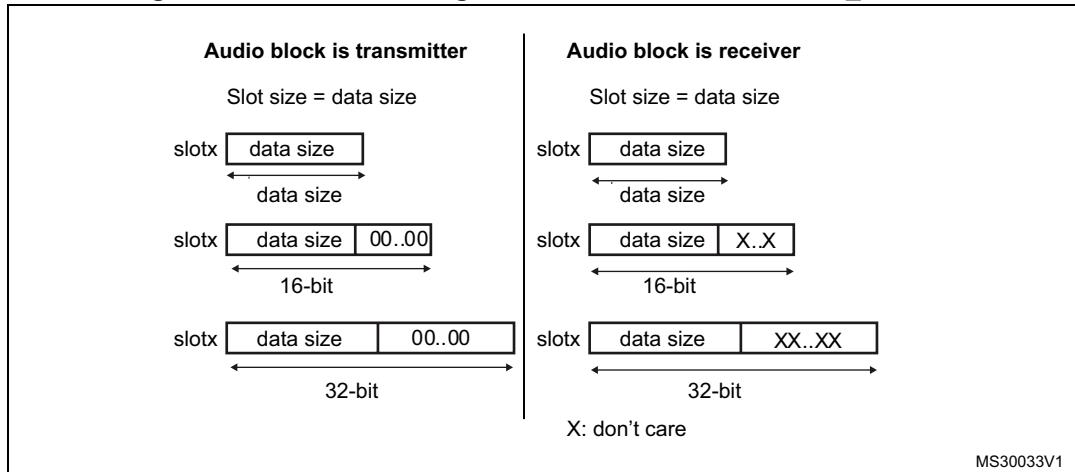
For AC'97 protocol or SPDIF (when bit PRTCFG[1:0] = 10 or PRTCFG[1:0] = 01), the number of slots is automatically set to target the protocol specification, and the value of NBSLOT[3:0] is ignored.

Each slot can be defined as a valid slot, or not, by setting SLOTEN[15:0] bits of the SAI_xSLOTR register.

When an invalid slot is transferred, the SD data line is either forced to 0 or released to HI-Z depending on TRIS bit configuration (refer to [Output data line management on an inactive slot](#)) in transmitter mode. In receiver mode, the received value from the end of this slot is ignored. Consequently, there will be no FIFO access and so no request to read or write the FIFO linked to this inactive slot status.

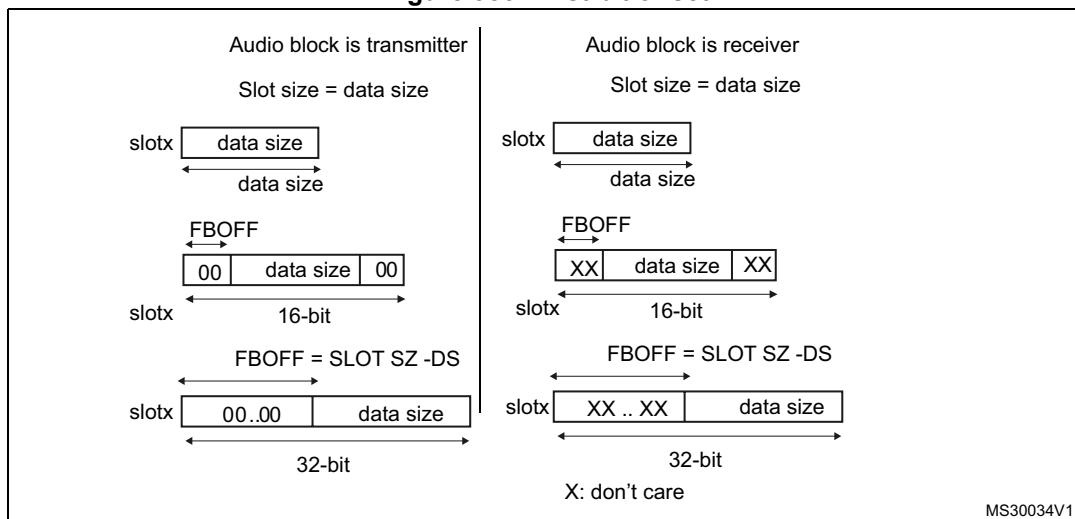
The slot size is also configurable as shown in [Figure 384](#). The size of the slots is selected by setting SLOTSZ[1:0] bits in the SAI_xSLOTR register. The size is applied identically for each slot in an audio frame.

Figure 384. Slot size configuration with FBOFF = 0 in SAI_xSLOTR



It is possible to choose the position of the first data bit to transfer within the slots. This offset is configured by FBOFF[4:0] bits in the SAI_xSLOTR register. 0 values will be injected in transmitter mode from the beginning of the slot until this offset position is reached. In reception, the bit in the offset phase is ignored. This feature targets the LSB justified protocol (if the offset is equal to the slot size minus the data size).

Figure 385. First bit offset



It is mandatory to respect the following conditions to avoid bad SAI behavior:

- FBOFF \leq (SLOTSZ - DS),
- DS \leq SLOTSZ,
- NBSLOT \times SLOTSZ \leq FRL (frame length),

The number of slots must be even when bit FSDEF in the SAI_xFRCR register is set.

In AC'97 and SPDIF protocol (bit PRTCFG[1:0] = 10 or PRTCFG[1:0] = 01), the slot size is automatically set as defined in [Section 29.3.10: AC'97 link controller](#).

29.3.8 SAI clock generator

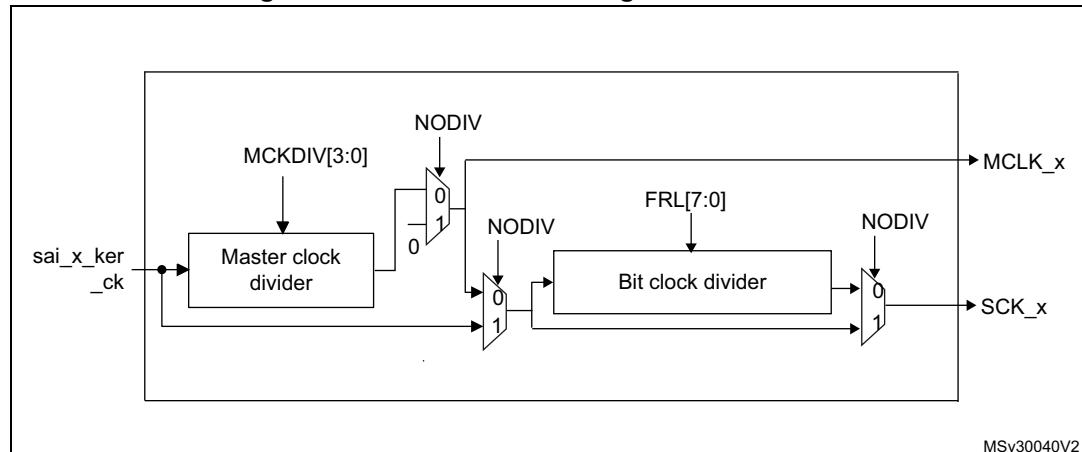
Each audio block has its own clock generator that makes these two blocks completely independent. There is no difference in terms of functionality between these two clock generators.

When the audio block is configured as Master, the clock generator provides the communication clock (the bit clock) and the master clock for external decoders.

When the audio block is defined as slave, the clock generator is OFF.

Figure 386 illustrates the architecture of the audio block clock generator.

Figure 386. Audio block clock generator overview



Note: If NODIV is set to 1, the MCLK_x signal will be set at 0 level if this pin is configured as the SAI pin in GPIO peripherals.

The clock source for the clock generator comes from the product clock controller. The sai_x_ker_ck clock is equivalent to the master clock which can be divided for the external decoders using bit MCKDIV[3:0]:

$$\text{MCLK}_x = \text{sai_x_ker_ck} / (\text{MCKDIV}[3:0] * 2), \text{ if MCKDIV}[3:0] \text{ is not equal to 0000.}$$

$$\text{MCLK}_x = \text{sai_x_ker_ck}, \text{ if MCKDIV}[3:0] \text{ is equal to 0000.}$$

MCLK_x signal is used only in TDM.

The division must be even in order to keep 50% on the Duty cycle on the MCLK output and on the SCK_x clock. If bit MCKDIV[3:0] = 0000, division by one is applied to obtain MCLK_x equal to sai_x_ker_ck.

In the SAI, the single ratio MCLK/FS = 256 is considered. Mostly, three frequency ranges will be encountered as illustrated in *Table 178*.

Table 178. Example of possible audio frequency sampling range

Input sai_x_ker_ck clock frequency	Most usual audio frequency sampling achievable	MCKDIV[3:0]
192 kHz x 256	192 kHz	MCKDIV[3:0] = 0000
	96 kHz	MCKDIV[3:0] = 0001
	48 kHz	MCKDIV[3:0] = 0010
	16 kHz	MCKDIV[3:0] = 0110
	8 kHz	MCKDIV[3:0] = 1100
44.1 kHz x 256	44.1 kHz	MCKDIV[3:0] = 0000
	22.05 kHz	MCKDIV[3:0] = 0001
	11.025 kHz	MCKDIV[3:0] = 0010
sai_x_ker_ck = MCLK ⁽¹⁾	MCLK	MCKDIV[3:0] = 0000

1. This may happen when the product clock controller selects an external clock source, instead of PLL clock.

The master clock can be generated externally on an I/O pad for external decoders if the corresponding audio block is declared as master with bit NODIV = 0 in the SAI_xCR1 register. In slave, the value set in this last bit is ignored since the clock generator is OFF, and the MCLK_x I/O pin is released for use as a general purpose I/O.

The bit clock is derived from the master clock. The bit clock divider sets the divider factor between the bit clock (SCK_x) and the master clock (MCLK_x) following the formula:

$$\text{SCK}_x = \text{MCLK}_x \times (\text{FRL}[7:0] + 1) / 256$$

where:

256 is the fixed ratio between MCLK and the audio frequency sampling.

FRL[7:0] is the number of bit clock cycles- 1 in the audio frame, configured in the SAI_xFRCR register.

In master mode it is mandatory that ($\text{FRL}[7:0] + 1$) is equal to a number with a power of 2 (refer to [Section 29.3.6: Frame synchronization](#)) to obtain an even integer number of MCLK_x pulses by bit clock cycle. The 50% duty cycle is guaranteed on the bit clock (SCK_x).

The sai_x_ker_ck clock can also be equal to the bit clock frequency. In this case, NODIV bit in the SAI_xCR1 register should be set and the value inside the MCKDIV divider and the bit clock divider will be ignored. In this case, the number of bits per frame is fully configurable without the need to be equal to a power of two.

The bit clock strobing edge on SCK can be configured by bit CKSTR in the SAI_xCR1 register.

Refer to [Section 29.3.11: SPDIF output](#) for details on clock generator programming in SPDIF mode.

29.3.9 Internal FIFOs

Each audio block in the SAI has its own FIFO. Depending if the block is defined to be a transmitter or a receiver, the FIFO can be written or read, respectively. There is therefore only one FIFO request linked to FREQ bit in the SAI_xSR register.

An interrupt is generated if FREQIE bit is enabled in the SAI_xIM register. This depends on:

- FIFO threshold setting (FLVL bits in SAI_xCR2)
- Communication direction (transmitter or receiver). Refer to [Interrupt generation in transmitter mode](#) and [Interrupt generation in reception mode](#).

Interrupt generation in transmitter mode

The interrupt generation depends on the FIFO configuration in transmitter mode:

- When the FIFO threshold bits in SAI_xCR2 register are configured as FIFO empty (FTH[2:0] set to 0b000), an interrupt is generated (FREQ bit set by hardware to 1 in SAI_xSR register) if no data are available in SAI_xDR register (FLVL[2:0] bits in SAI_xSR is less than 001b). This Interrupt (FREQ bit in SAI_xSR register) is cleared by hardware when the FIFO is no more empty (FLVL[2:0] bits in SAI_xSR are different from 0b000) i.e one or more data are stored in the FIFO.
- When the FIFO threshold bits in SAI_xCR2 register are configured as FIFO quarter full (FTH[2:0] set to 001b), an interrupt is generated (FREQ bit set by hardware to 1 in SAI_xSR register) if less than a quarter of the FIFO contains data (FLVL[2:0] bits in SAI_xSR are less than 0b010). This Interrupt (FREQ bit in SAI_xSR register) is cleared by hardware when at least a quarter of the FIFO contains data (FLVL[2:0] bits in SAI_xSR are higher or equal to 0b010).
- When the FIFO threshold bits in SAI_xCR2 register are configured as FIFO half full (FTH[2:0] set to 0b010), an interrupt is generated (FREQ bit set by hardware to 1 in SAI_xSR register) if less than half of the FIFO contains data (FLVL[2:0] bits in SAI_xSR are less than 011b). This Interrupt (FREQ bit in SAI_xSR register) is cleared by hardware when at least half of the FIFO contains data (FLVL[2:0] bits in SAI_xSR are higher or equal to 011b).
- When the FIFO threshold bits in SAI_xCR2 register are configured as FIFO three quarter (FTH[2:0] set to 011b), an interrupt is generated (FREQ bit is set by hardware to 1 in SAI_xSR register) if less than three quarters of the FIFO contain data (FLVL[2:0] bits in SAI_xSR are less than 0b100). This Interrupt (FREQ bit in SAI_xSR register) is cleared by hardware when at least three quarters of the FIFO contain data (FLVL[2:0] bits in SAI_xSR are higher or equal to 0b100).
- When the FIFO threshold bits in SAI_xCR2 register are configured as FIFO full (FTH[2:0] set to 0b100), an interrupt is generated (FREQ bit is set by hardware to 1 in SAI_xSR register) if the FIFO is not full (FLVL[2:0] bits in SAI_xSR is less than 101b). This Interrupt (FREQ bit in SAI_xSR register) is cleared by hardware when the FIFO is full (FLVL[2:0] bits in SAI_xSR is equal to 101b value).

Interrupt generation in reception mode

The interrupt generation depends on the FIFO configuration in reception mode:

- When the FIFO threshold bits in SAI_xCR2 register are configured as FIFO empty (FTH[2:0] set to 0b000), an interrupt is generated (FREQ bit is set by hardware to 1 in SAI_xSR register) if at least one data is available in SAI_xDR register(FLVL[2:0] bits in SAI_xSR is higher or equal to 001b). This Interrupt (FREQ bit in SAI_xSR register) is

cleared by hardware when the FIFO becomes empty (FLVL[2:0] bits in SAI_xSR is equal to 0b000) i.e no data are stored in FIFO.

- When the FIFO threshold bits in SAI_xCR2 register are configured as FIFO quarter fully (FTH[2:0] set to 001b), an interrupt is generated (FREQ bit is set by hardware to 1 in SAI_xSR register) if at least one quarter of the FIFO data locations are available (FLVL[2:0] bits in SAI_xSR is higher or equal to 0b010). This Interrupt (FREQ bit in SAI_xSR register) is cleared by hardware when less than a quarter of the FIFO data locations become available (FLVL[2:0] bits in SAI_xSR is less than 0b010).
- When the FIFO threshold bits in SAI_xCR2 register are configured as FIFO half fully (FTH[2:0] set to 0b010 value), an interrupt is generated (FREQ bit is set by hardware to 1 in SAI_xSR register) if at least half of the FIFO data locations are available (FLVL[2:0] bits in SAI_xSR is higher or equal to 011b). This Interrupt (FREQ bit in SAI_xSR register) is cleared by hardware when less than half of the FIFO data locations become available (FLVL[2:0] bits in SAI_xSR is less than 011b).
- When the FIFO threshold bits in SAI_xCR2 register are configured as FIFO three quarter full(FTH[2:0] set to 011b value), an interrupt is generated (FREQ bit is set by hardware to 1 in SAI_xSR register) if at least three quarters of the FIFO data locations are available (FLVL[2:0] bits in SAI_xSR is higher or equal to 0b100). This Interrupt (FREQ bit in SAI_xSR register) is cleared by hardware when the FIFO has less than three quarters of the FIFO data locations available(FLVL[2:0] bits in SAI_xSR is less than 0b100).
- When the FIFO threshold bits in SAI_xCR2 register are configured as FIFO full(FTH[2:0] set to 0b100), an interrupt is generated (FREQ bit is set by hardware to 1 in SAI_xSR register) if the FIFO is full (FLVL[2:0] bits in SAI_xSR is equal to 101b). This Interrupt (FREQ bit in SAI_xSR register) is cleared by hardware when the FIFO is not full (FLVL[2:0] bits in SAI_xSR is less than 101b).

Like interrupt generation, the SAI can use the DMA if DMAEN bit in the SAI_xCR1 register is set. The FREQ bit assertion mechanism is the same as the interruption generation mechanism described above for FREQIE.

Each FIFO is an 8-word FIFO. Each read or write operation from/to the FIFO targets one word FIFO location whatever the access size. Each FIFO word contains one audio slot. FIFO pointers are incremented by one word after each access to the SAI_xDR register.

Data should be right aligned when it is written in the SAI_xDR.

Data received will be right aligned in the SAI_xDR.

The FIFO pointers can be reinitialized when the SAI is disabled by setting bit FFLUSH in the SAI_xCR2 register. If FFLUSH is set when the SAI is enabled the data present in the FIFO will be lost automatically.

29.3.10 AC'97 link controller

The SAI is able to work as an AC'97 link controller. In this protocol:

- The slot number and the slot size are fixed.
- The frame synchronization signal is perfectly defined and has a fixed shape.

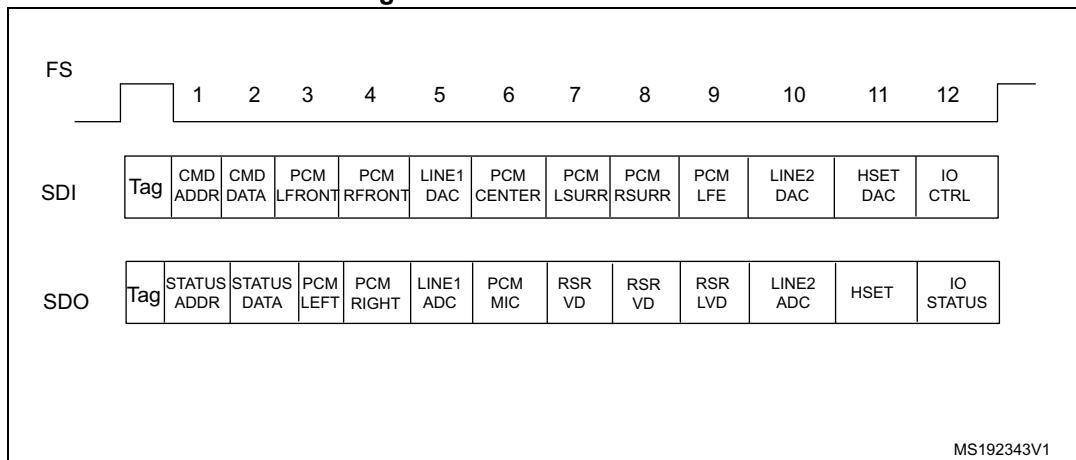
To select this protocol, set PRTC_{FG}[1:0] bits in the SAI_xCR1 register to 10. When AC'97 mode is selected, only data sizes of 16 or 20 bits can be used, otherwise the SAI behavior is not guaranteed.

- NBSLOT[3:0] and SLOTSZ[1:0] bits are consequently ignored.
- The number of slots is fixed to 13 slots. The first one is 16-bit wide and all the others are 20-bit wide (data slots).
- FBOFF[4:0] bits in the SAI_xSLOTR register are ignored.
- The SAI_xFRCR register is ignored.
- The MCLK is not used.

The FS signal from the block defined as asynchronous is configured automatically as an output, since the AC'97 controller link drives the FS signal whatever the master or slave configuration.

Figure 387 shows an AC'97 audio frame structure.

Figure 387. AC'97 audio frame



Note:

In AC'97 protocol, bit 2 of the tag is reserved (always 0), so bit 2 of the TAG is forced to 0 level whatever the value written in the SAI FIFO.

For more details about tag representation, refer to the AC'97 protocol standard.

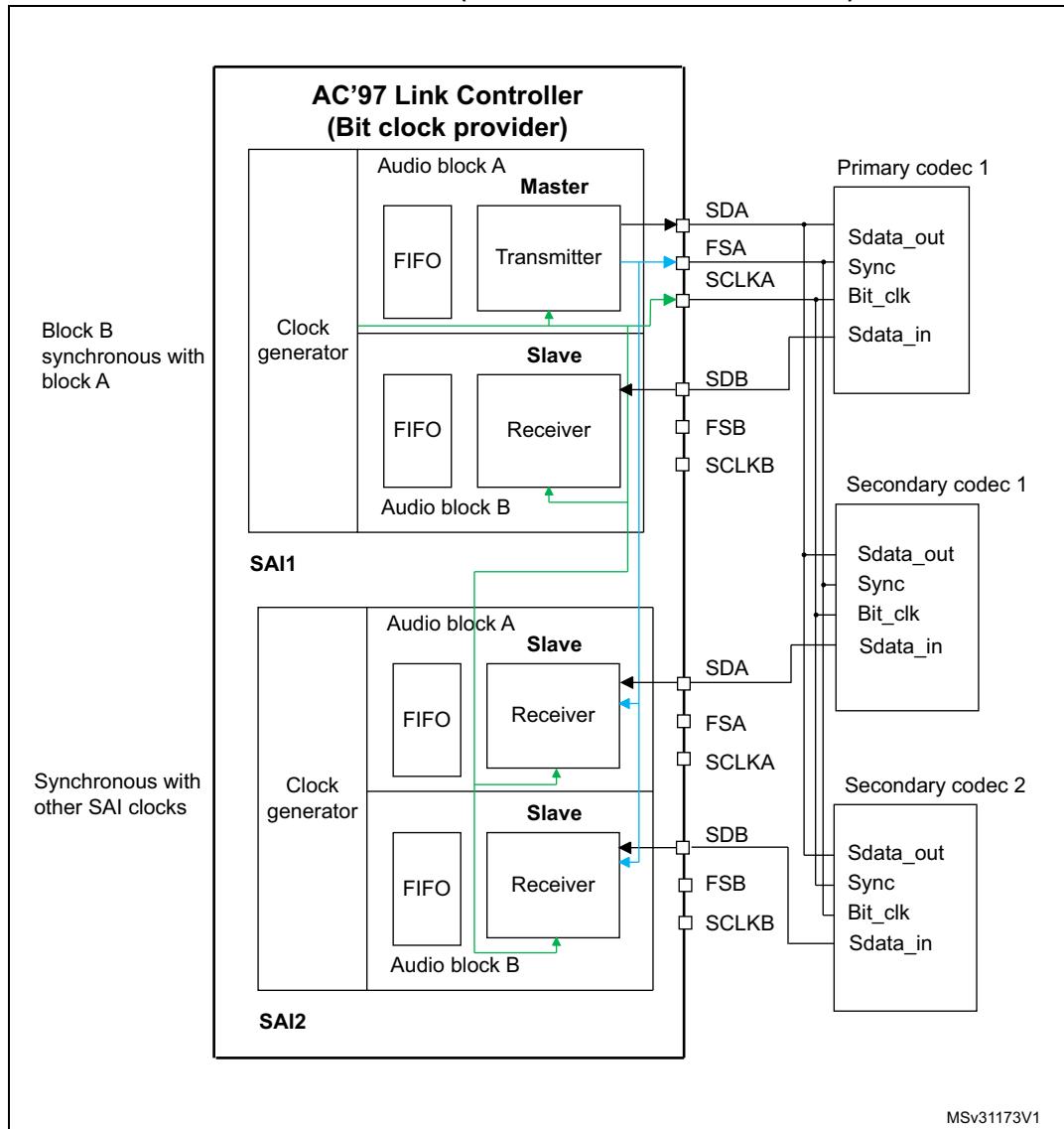
One SAI can be used to target an AC'97 point-to-point communication.

Using two SAIs (for devices featuring two embedded SAIs) allows controlling three external AC'97 decoders as illustrated in *Figure 388*.

In SAI1, the audio block A must be declared as asynchronous master transmitter whereas the audio block B is defined to be slave receiver and internally synchronous to the audio block A.

The SAI2 is configured for audio block A and B both synchronous with the external SAI1 in slave receiver mode.

Figure 388. Example of typical AC'97 configuration on devices featuring at least 2 embedded SAI (three external AC'97 decoders)



In receiver mode, the SAI acting as an AC'97 link controller requires no FIFO request and so no data storage in the FIFO when the Codec ready bit in the slot 0 is decoded low. If bit CNRDYIE is enabled in the SAI_xIM register, flag CNRDY will be set in the SAI_xSR register and an interrupt is generated. This flag is dedicated to the AC'97 protocol.

Clock generator programming in AC'97 mode

In AC'97 mode, the frame length is fixed at 256 bits, and its frequency shall be set to 48 kHz. The formulas given in [Section 29.3.8: SAI clock generator](#) shall be used with $F_{RL} = 255$, in order to generate the proper frame rate (F_{FS_x}).

29.3.11 SPDIF output

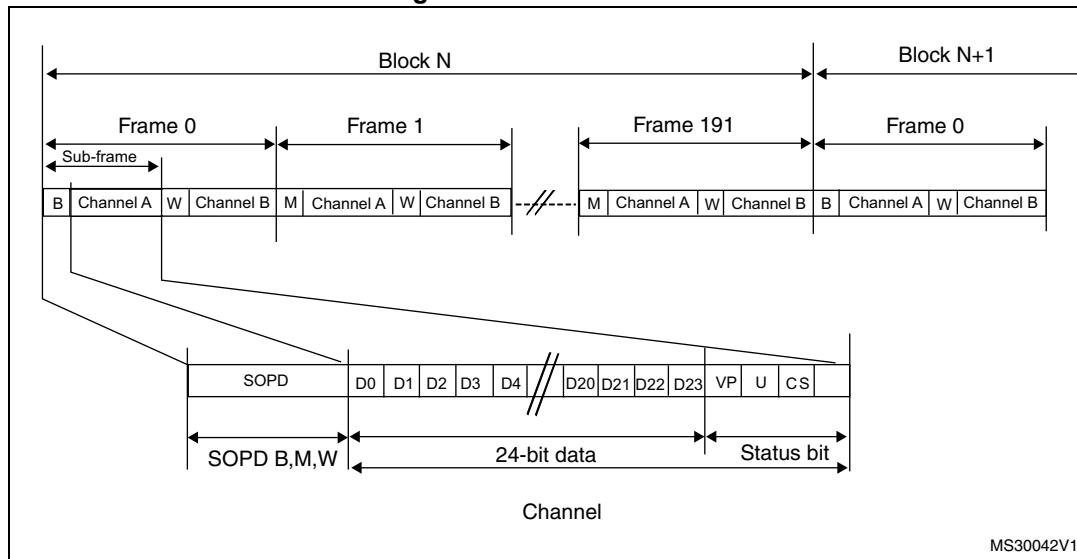
The SPDIF interface is available in transmitter mode only. It supports the audio IEC60958.

To select SPDIF mode, set PRTCFG[1:0] bit to 01 in the SAI_xCR1 register.

For SPDIF protocol:

- Only SD data line is enabled.
- FS, SCK, MCLK I/Os pins are left free.
- MODE[1] bit is forced to 0 to select the master mode in order to enable the clock generator of the SAI and manage the data rate on the SD line.
- The data size is forced to 24 bits. The value set in DS[2:0] bits in the SAI_xCR1 register is ignored.
- The clock generator must be configured to define the symbol-rate, knowing that the bit clock should be twice the symbol-rate. The data is coded in Manchester protocol.
- The SAI_xFRCR and SAI_xSLOTR registers are ignored. The SAI is configured internally to match the SPDIF protocol requirements as shown in [Figure 389](#).

Figure 389. SPDIF format



A SPDIF block contains 192 frames. Each frame is composed of two 32-bit sub-frames, generally one for the left channel and one for the right channel. Each sub-frame is composed of a SOPD pattern (4-bit) to specify if the sub-frame is the start of a block (and so is identifying a channel A) or if it is identifying a channel A somewhere in the block, or if it is referring to channel B (see [Table 179](#)). The next 28 bits of channel information are composed of 24 bits data + 4 status bits.

Table 179. SOPD pattern

SOPD	Preamble coding		Description
	last bit is 0	last bit is 1	
B	11101000	00010111	Channel A data at the start of block
W	11100100	00011011	Channel B data somewhere in the block
M	11100010	00011101	Channel A data

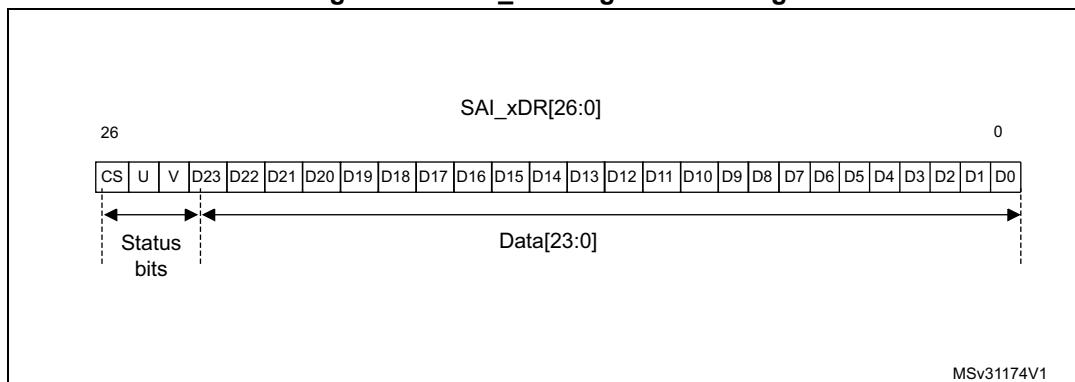
The data stored in SAI_xDR has to be filled as follows:

- SAI_xDR[26:24] contain the Channel status, User and Validity bits.
- SAI_xDR[23:0] contain the 24-bit data for the considered channel.

If the data size is 20 bits, then data shall be mapped on SAI_xDR[23:4].

If the data size is 16 bits, then data shall be mapped on SAI_xDR[23:8].

SAI_xDR[23] always represents the MSB.

Figure 390. SAI_xDR register ordering

Note:

The transfer is performed always with LSB first.

The SAI first sends the adequate preamble for each sub-frame in a block. The SAI_xDR is then sent on the SD line (manchester coded). The SAI ends the sub-frame by transferring the Parity bit calculated as described in [Table 180](#).

Table 180. Parity bit calculation

SAI_xDR[26:0]	Parity bit P value transferred
odd number of 0	0
odd number of 1	1

The underrun is the only error flag available in the SAI_xSR register for SPDIF mode since the SAI can only operate in transmitter mode. As a result, the following sequence should be

executed to recover from an underrun error detected via the underrun interrupt or the underrun status bit:

1. Disable the DMA stream (via the DMA peripheral) if the DMA is used.
2. Disable the SAI and check that the peripheral is physically disabled by polling the SAIEN bit in SAI_xCR1 register.
3. Clear the COVRUNDR flag in the SAI_xCLRFR register.
4. Flush the FIFO by setting the FFLUSH bit in SAI_xCR2.

The software needs to point to the address of the future data corresponding to a start of new block (data for preamble B). If the DMA is used, the DMA source base address pointer should be updated accordingly.

5. Enable again the DMA stream (DMA peripheral) if the DMA used to manage data transfers according to the new source base address.
6. Enable again the SAI by setting SAIEN bit in SAI_xCR1 register.

Clock generator programming in SPDIF generator mode

For the SPDIF generator, the SAI shall provide a bit clock twice faster as the symbol-rate. The table hereafter shows usual examples of symbol rates with respect to the audio sampling rate.

Table 181. Audio sampling frequency versus symbol rates

Audio Sampling Frequencies (F_S)	Symbol-rate
44.1 kHz	2.8224 MHz
48 kHz	3.072 MHz
96 kHz	6.144 MHz
192 kHz	12.288 MHz

More generally, the relationship between the audio sampling frequency (F_S) and the bit clock rate (F_{SCK_x}) is given by the formula:

$$F_S = \frac{F_{SCK_x}}{128}$$

And the bit clock rate is obtained as follow:

$$F_{SCK_x} = F_{SAI_CK_x}$$

29.3.12 Specific features

The SAI interface embeds specific features which can be useful depending on the audio protocol selected. These functions are accessible through specific bits of the SAI_xCR2 register.

Mute mode

The mute mode can be used when the audio sub-block is a transmitter or a receiver.

Audio sub-block in transmission mode

In transmitter mode, the mute mode can be selected at anytime. The mute mode is active for entire audio frames. The MUTE bit in the SAI_xCR2 register enables the mute mode when it is set during an ongoing frame.

The mute mode bit is strobed only at the end of the frame. If it is set at this time, the mute mode is active at the beginning of the new audio frame and for a complete frame, until the next end of frame. The bit is then strobed to determine if the next frame will still be a mute frame.

If the number of slots set through NBSLOT[3:0] bits in the SAI_xSLOTR register is lower than or equal to 2, it is possible to specify if the value sent in mute mode is 0 or if it is the last value of each slot. The selection is done via MUTEVAL bit in the SAI_xCR2 register.

If the number of slots set in NBSLOT[3:0] bits in the SAI_xSLOTR register is greater than 2, MUTEVAL bit in the SAI_xCR2 is meaningless as 0 values are sent on each bit on each slot.

The FIFO pointers are still incremented in mute mode. This means that data present in the FIFO and for which the mute mode is requested are discarded.

Audio sub-block in reception mode

In reception mode, it is possible to detect a mute mode sent from the external transmitter when all the declared and valid slots of the audio frame receive 0 for a given consecutive number of audio frames (MUTECNT[5:0] bits in the SAI_xCR2 register).

When the number of MUTE frames is detected, the MUTEDET flag in the SAI_xSR register is set and an interrupt can be generated if MUTEDETIE bit is set in SAI_xCR2.

The mute frame counter is cleared when the audio sub-block is disabled or when a valid slot receives at least one data in an audio frame. The interrupt is generated just once, when the counter reaches the value specified in MUTECNT[5:0] bits. The interrupt event is then reinitialized when the counter is cleared.

Note:

The mute mode is not available for SPDIF audio blocks.

Mono/stereo mode

In transmitter mode, the mono mode can be addressed, without any data preprocessing in memory, assuming the number of slots is equal to 2 (NBSLOT[3:0] = 0001 in SAI_xSLOTR). In this case, the access time to and from the FIFO will be reduced by 2 since the data for slot 0 is duplicated into data slot 1.

To enable the mono mode,

1. Set MONO bit to 1 in the SAI_xCR1 register.
2. Set NBSLOT to 1 and SLOTEN to 3 in SAI_xSLOTR.

In reception mode, the MONO bit can be set and is meaningful only if the number of slots is equal to 2 as in transmitter mode. When it is set, only slot 0 data will be stored in the FIFO. The data belonging to slot 1 will be discarded since, in this case, it is supposed to be the same as the previous slot. If the data flow in reception mode is a real stereo audio flow with a distinct and different left and right data, the MONO bit is meaningless. The conversion from the output stereo file to the equivalent mono file is done by software.

Companding mode

Telecommunication applications can require to process the data to be transmitted or received using a data companding algorithm.

Depending on the COMP[1:0] bits in the SAI_xCR2 register (used only when TDM mode is selected), the application software can choose to process or not the data before sending it on SD serial output line (compression) or to expand the data after the reception on SD serial input line (expansion) as illustrated in [Figure 391](#). The two companding modes supported are the μ -Law and the A-Law log which are a part of the CCITT G.711 recommendation.

The companding standard used in the United States and Japan is the μ -Law. It supports 14 bits of dynamic range (COMP[1:0] = 10 in the SAI_xCR2 register).

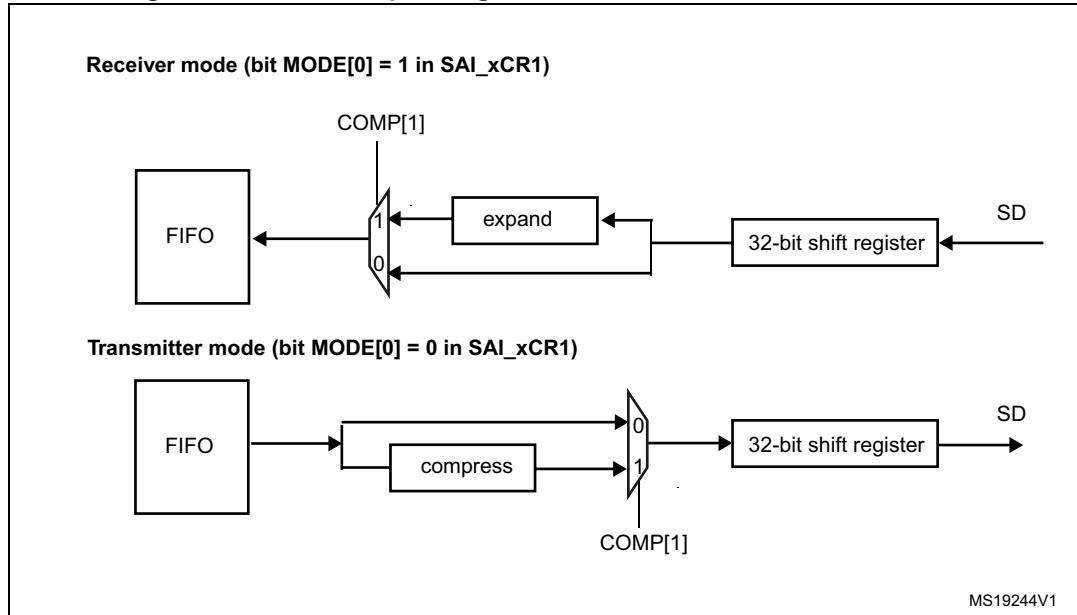
The European companding standard is A-Law and supports 13 bits of dynamic range (COMP[1:0] = 11 in the SAI_xCR2 register).

Both μ -Law or A-Law companding standard can be computed based on 1's complement or 2's complement representation depending on the CPL bit setting in the SAI_xCR2 register.

In μ -Law and A-Law standards, data are coded as 8 bits with MSB alignment. Companded data are always 8-bit wide. For this reason, DS[2:0] bits in the SAI_xCR1 register will be forced to 010 when the SAI audio block is enabled (SAIEN bit = 1 in the SAI_xCR1 register) and when one of these two companding modes selected through the COMP[1:0] bits.

If no companding processing is required, COMP[1:0] bits should be kept clear.

Figure 391. Data companding hardware in an audio block in the SAI



1. Not applicable when AC'97 or SPDIF are selected.

Expansion and compression mode are automatically selected through the SAI_xCR2:

- If the SAI audio block is configured to be a transmitter, and if the COMP[1] bit is set in the SAI_xCR2 register, the compression mode will be applied.
- If the SAI audio block is declared as a receiver, the expansion algorithm will be applied.

Output data line management on an inactive slot

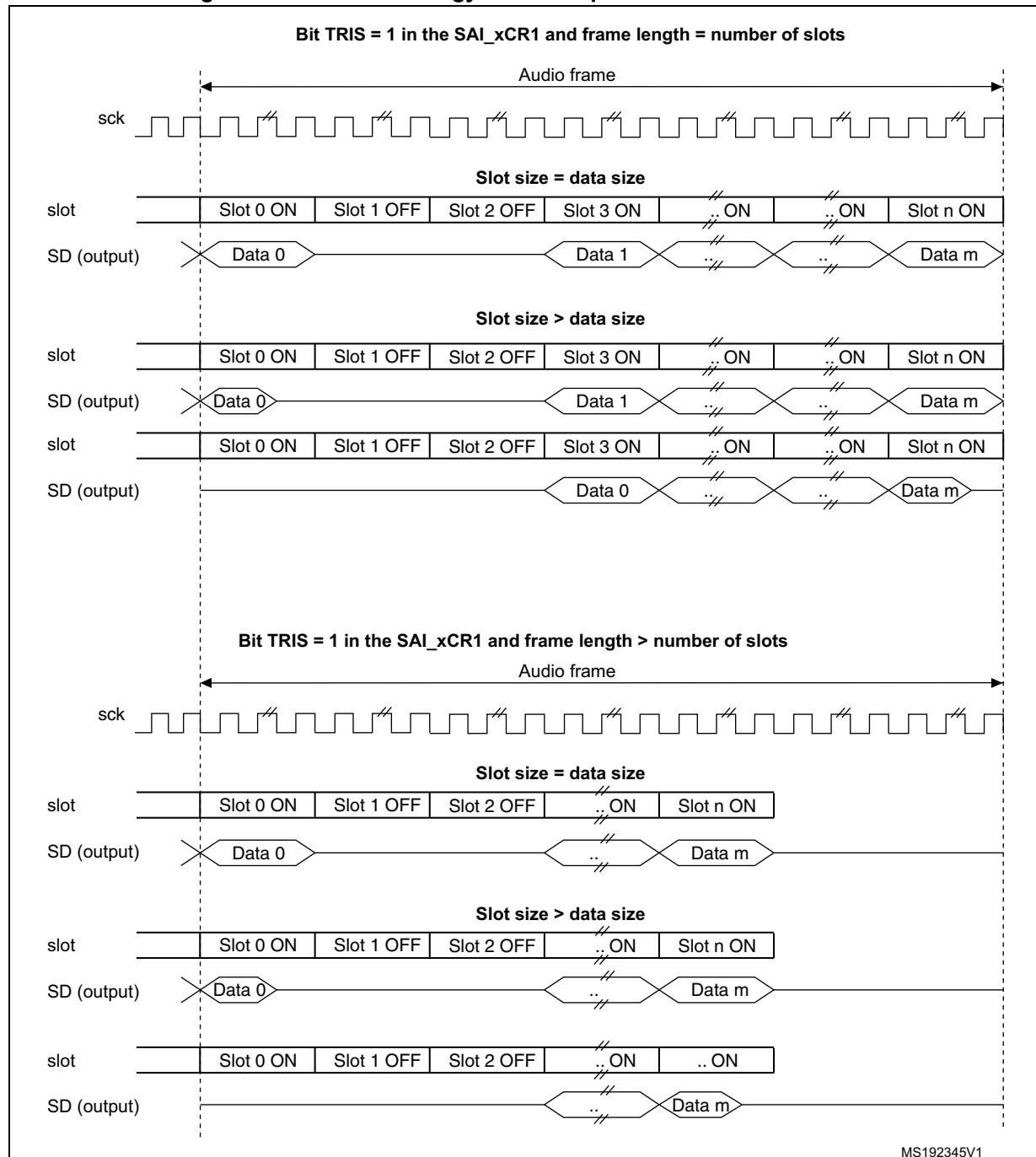
In transmitter mode, it is possible to choose the behavior of the SD line output when an inactive slot is sent on the data line (via TRIS bit).

- Either the SAI forces 0 on the SD output line when an inactive slot is transmitted, or
- The line is released in HI-z state at the end of the last bit of data transferred, to release the line for other transmitters connected to this node.

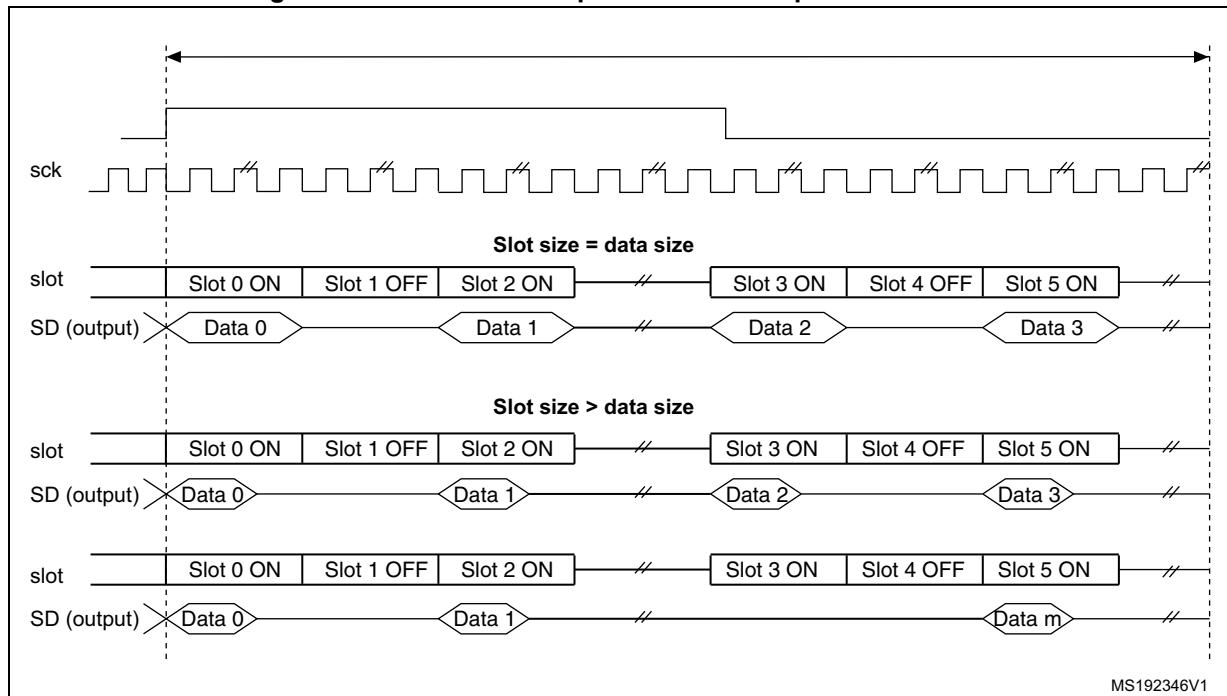
It is important to note that the two transmitters cannot attempt to drive the same SD output pin simultaneously, which could result in a short circuit. To ensure a gap between transmissions, if the data is lower than 32-bit, the data can be extended to 32-bit by setting bit SLOTSZ[1:0] = 10 in the SAI_xSLOTR register. The SD output pin will then be tri-stated at the end of the LSB of the active slot (during the padding to 0 phase to extend the data to 32-bit) if the following slot is declared inactive.

In addition, if the number of slots multiplied by the slot size is lower than the frame length, the SD output line will be tri-stated when the padding to 0 is done to complete the audio frame.

Figure 392 illustrates these behaviors.

Figure 392. Tristate strategy on SD output line on an inactive slot

When the selected audio protocol uses the FS signal as a start of frame and a channel side identification (bit FSDEF = 1 in the SAI_xFRCR register), the tristate mode is managed according to [Figure 393](#) (where bit TRIS in the SAI_xCR1 register = 1, and FSDEF=1, and half frame length is higher than number of slots/2, and NBSLOT=6).

Figure 393. Tristate on output data line in a protocol like I2S

If the TRIS bit in the SAI_xCR2 register is cleared, all the High impedance states on the SD output line on [Figure 392](#) and [Figure 393](#) are replaced by a drive with a value of 0.

29.3.13 Error flags

The SAI implements the following error flags:

- FIFO overrun/underrun
- Anticipated frame synchronization detection
- Late frame synchronization detection
- Codec not ready (AC'97 exclusively)
- Wrong clock configuration in master mode.

FIFO overrun/underrun (OVRUDR)

The FIFO overrun/underrun bit is called OVRUDR in the SAI_xSR register.

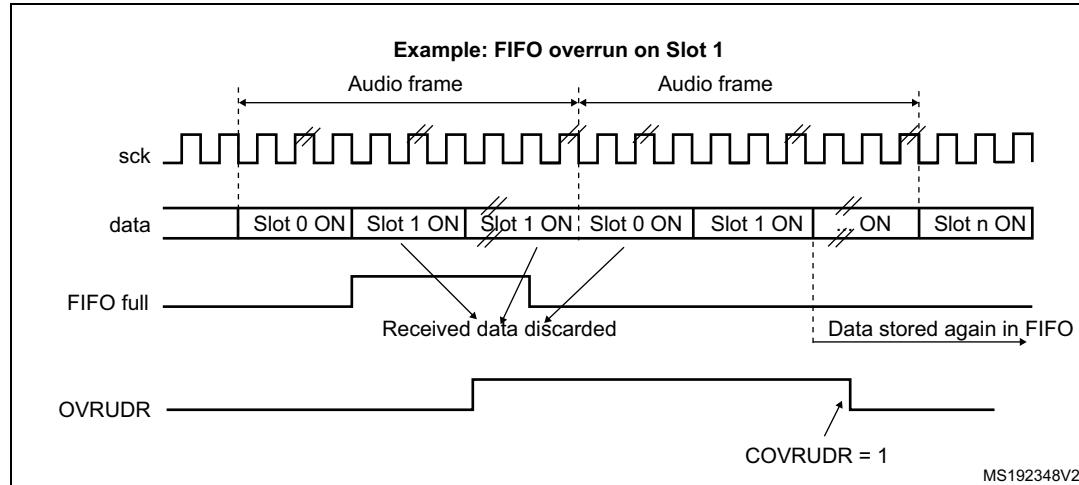
The overrun or underrun errors share the same bit since an audio block can be either receiver or transmitter and each audio block in a given SAI has its own SAI_xSR register.

Overrun

When the audio block is configured as receiver, an overrun condition may appear if data are received in an audio frame when the FIFO is full and not able to store the received data. In this case, the received data are lost, the flag OVRUDR in the SAI_xSR register is set and an interrupt is generated if OVRUDRIE bit is set in the SAI_xIM register. The slot number, from which the overrun occurs, is stored internally. No more data will be stored into the FIFO until it becomes free to store new data. When the FIFO has at least one data free, the SAI audio block receiver will store new data (from new audio frame) from the slot number which was stored internally when the overrun condition was detected. This avoids data slot de-alignment in the destination memory (refer to [Figure 394](#)).

The OVRUDR flag is cleared when COVRUDR bit is set in the SAI_xCLRFR register.

Figure 394. Overrun detection error



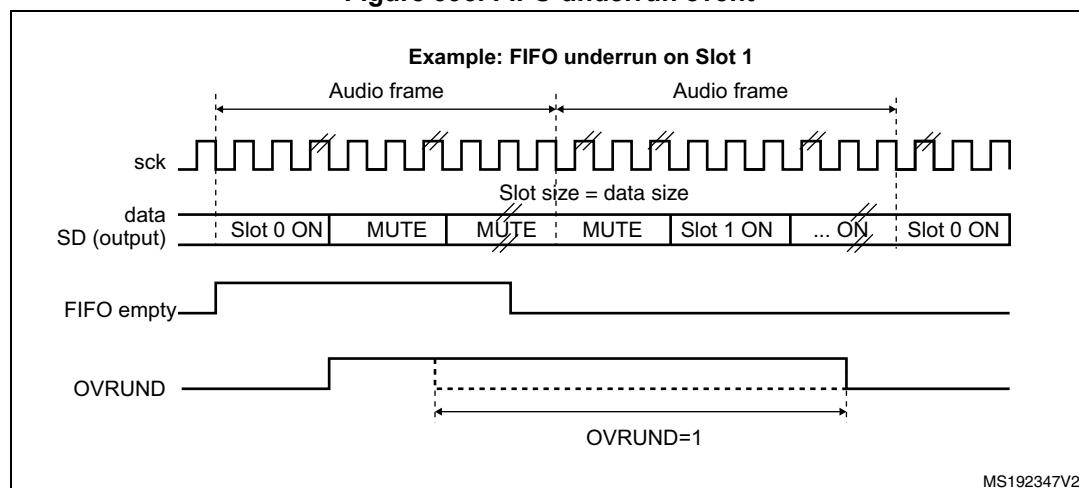
Underrun

An underrun may occur when the audio block in the SAI is a transmitter and the FIFO is empty when data need to be transmitted. If an underrun is detected, the slot number for which the event occurs is stored and MUTE value (00) is sent until the FIFO is ready to transmit the data corresponding to the slot for which the underrun was detected (refer to [Figure 395](#)). This avoids desynchronization between the memory pointer and the slot in the audio frame.

The underrun event sets the OVRUDR flag in the SAI_xSR register and an interrupt is generated if the OVRUDRIE bit is set in the SAI_xIM register. To clear this flag, set COVRUDR bit in the SAI_xCLRFR register.

The underrun event can occur when the audio sub-block is configured as master or slave.

Figure 395. FIFO underrun event



Anticipated frame synchronization detection (AFSDET)

The AFSDET flag is used only in slave mode. It is never asserted in master mode. It indicates that a frame synchronization (FS) has been detected earlier than expected since the frame length, the frame polarity, the frame offset are defined and known.

Anticipated frame detection sets the AFSDET flag in the SAI_xSR register.

This detection has no effect on the current audio frame which is not sensitive to the anticipated FS. This means that “parasitic” events on signal FS are flagged without any perturbation of the current audio frame.

An interrupt is generated if the AFSDETIE bit is set in the SAI_xIM register. To clear the AFSDET flag, CAFSDET bit must be set in the SAI_xCLRFR register.

To resynchronize with the master after an anticipated frame detection error, four steps are required:

1. Disable the SAI block by resetting SAIEN bit in SAI_xCR1 register. To make sure the SAI is disabled, read back the SAIEN bit and check it is set to 0.
2. Flush the FIFO via FFLUS bit in SAI_xCR2 register.
3. Enable again the SAI peripheral (SAIEN bit set to 1).
4. The SAI block will wait for the assertion on FS to restart the synchronization with master.

Note: *The AFSDET flag is not asserted in AC'97 mode since the SAI audio block acts as a link controller and generates the FS signal even when declared as slave. It has no meaning in SPDIF mode since the FS signal is not used.*

Late frame synchronization detection

The LFSDET flag in the SAI_xSR register can be set only when the SAI audio block operates as a slave. The frame length, the frame polarity and the frame offset configuration are known in register SAI_xFRCR.

If the external master does not send the FS signal at the expecting time thus generating the signal too late, the LFSDET flag is set and an interrupt is generated if LFSDETIE bit is set in the SAI_xIM register.

The LFSDET flag is cleared when CLFSDET bit is set in the SAI_xCLRFR register.

The late frame synchronization detection flag is set when the corresponding error is detected. The SAI needs to be resynchronized with the master (see sequence described in [Anticipated frame synchronization detection \(AFSDET\)](#)).

In a noisy environment, glitches on the SCK clock may be wrongly detected by the audio block state machine and shift the SAI data at a wrong frame position. This event can be detected by the SAI and reported as a late frame synchronization detection error.

There is no corruption if the external master is not managing the audio data frame transfer in continuous mode, which should not be the case in most applications. In this case, the LFSDET flag will be set.

Note: *The LFSDET flag is not asserted in AC'97 mode since the SAI audio block acts as a link controller and generates the FS signal even when declared as slave. It has no meaning in SPDIF mode since the signal FS is not used by the protocol.*

Codec not ready (CNRDY AC'97)

The CNRDY flag in the SAI_xSR register is relevant only if the SAI audio block is configured to operate in AC'97 mode (PRTCFG[1:0] = 10 in the SAI_xCR1 register). If CNRDYIE bit is set in the SAI_xIM register, an interrupt is generated when the CNRDY flag is set.

CNRDY is asserted when the Codec is not ready to communicate during the reception of the TAG 0 (slot0) of the AC'97 audio frame. In this case, no data will be automatically stored into the FIFO since the Codec is not ready, until the TAG 0 indicates that the Codec is ready. All the active slots defined in the SAI_xSLOTR register will be captured when the Codec is ready.

To clear CNRDY flag, CCNRDY bit must be set in the SAI_xCLRFR register.

Wrong clock configuration in master mode (with NODIV = 0)

When the audio block operates as a master (MODE[1] = 0) and NODIV bit is equal to 0, the WCKCFG flag is set as soon as the SAI is enabled if the following conditions are met:

- (FRL+1) is not a power of 2, and
- (FRL+1) is not between 8 and 256.

MODE, NODIV, and SAIEN bits belong to SAI_xCR1 register and FRL to SAI_xFRCR register.

If WCKCFGIE bit is set, an interrupt is generated when WCKCFG flag is set in the SAI_xSR register. To clear this flag, set CWCKCFG bit in the SAI_xCLRFR register.

When WCKCFG bit is set, the audio block is automatically disabled, thus performing a hardware clear of SAIEN bit.

29.3.14 Disabling the SAI

The SAI audio block can be disabled at any moment by clearing SAIEN bit in the SAI_xCR1 register. All the already started frames are automatically completed before the SAI is stopped working. SAIEN bit remains High until the SAI is completely switched-off at the end of the current audio frame transfer.

If an audio block in the SAI operates synchronously with the other one, the one which is the master must be disabled first.

29.3.15 SAI DMA interface

To free the CPU and to optimize bus bandwidth, each SAI audio block has an independent DMA interface to read/write from/to the SAI_xDR register (to access the internal FIFO).

There is one DMA channel per audio sub-block supporting basic DMA request/acknowledge protocol.

To configure the audio sub-block for DMA transfer, set DMAEN bit in the SAI_xCR1 register. The DMA request is managed directly by the FIFO controller depending on the FIFO threshold level (for more details refer to [Section 29.3.9: Internal FIFOs](#)). DMA transfer direction is linked to the SAI audio sub-block configuration:

- If the audio block operates as a transmitter, the audio block FIFO controller outputs a DMA request to load the FIFO with data written in the SAI_xDR register.
- If the audio block operates as a receiver, the DMA request is related to read operations from the SAI_xDR register.

Follow the sequence below to configure the SAI interface in DMA mode:

1. Configure SAI and FIFO threshold levels to specify when the DMA request will be launched.
2. Configure SAI DMA channel.
3. Enable the DMA.
4. Enable the SAI interface.

Note: *Before configuring the SAI block, the SAI DMA channel must be disabled.*

29.4 SAI interrupts

The SAI supports 7 interrupt sources as shown in [Table 182](#).

Table 182. SAI interrupt sources

Interrupt source	Interrupt group	Audio block mode	Interrupt enable	Interrupt clear
FREQ	FREQ	Master or slave Receiver or transmitter	FREQIE in SAI_xIM register	Depends on: – FIFO threshold setting (FLVL bits in SAI_xCR2) – Communication direction (transmitter or receiver) For more details refer to Section 29.3.9: Internal FIFOs
OVRUDR	ERROR	Master or slave Receiver or transmitter	OVRUDRIE in SAI_xIM register	COVRUDR = 1 in SAI_xCLRFR register
AFSDET	ERROR	Slave (not used in AC'97 mode and SPDIF mode)	AFSDETIE in SAI_xIM register	CAFSDET = 1 in SAI_xCLRFR register
LFSDET	ERROR	Slave (not used in AC'97 mode and SPDIF mode)	LFSDETIE in SAI_xIM register	CLFSDET = 1 in SAI_xCLRFR register
CNRDY	ERROR	Slave (only in AC'97 mode)	CNRDYIE in SAI_xIM register	CCNRDY = 1 in SAI_xCLRFR register
MUTEDET	MUTE	Master or slave Receiver mode only	MUTEDETIE in SAI_xIM register	CMUTEDET = 1 in SAI_xCLRFR register
WCKCFG	ERROR	Master with NODIV = 0 in SAI_xCR1 register	WCKCFGIE in SAI_xIM register	CWCKCFG = 1 in SAI_xCLRFR register

Follow the sequence below to enable an interrupt:

1. Disable SAI interrupt.
2. Configure SAI.
3. Configure SAI interrupt source.
4. Enable SAI.

29.5 SAI registers

29.5.1 Global configuration register (SAI_GCR)

Address offset: 0x000

Reset value: 0x0000 0000

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

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

Bits 5:4 **SYNCOUT[1:0]**: Synchronization outputs

These bits are set and cleared by software.

00: No synchronization output signals. SYNCOUT[1:0] should be configured as “No synchronization output signals” when audio block is configured as SPDIF

01: Block A used for further synchronization for others SAI

10: Block B used for further synchronization for others SAI

11: Reserved. These bits must be set when both audio block (A and B) are disabled.

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

Bits 1:0 **SYNCIN[1:0]**: Synchronization inputs

These bits are set and cleared by software.

Refer to [Table 177: External synchronization selection](#) for information on how to program this field.

These bits must be set when both audio blocks (A and B) are disabled.

They are meaningful if one of the two audio blocks is defined to operate in synchronous mode with an external SAI (SYNCEN[1:0] = 10 in SAI_ACR1 or in SAI_BCR1 registers).

29.5.2 Configuration register 1 (SAI_ACR1)

Address offset: 0x004

Reset value: 0x0000 0040

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	MCKDIV[3:0]				NODIV	Res.	DMAEN	SAIEN
								RW	RW	RW	RW	RW		RW	RW
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	OUTD RIV	MONO	SYNCEN[1:0]	CKSTR	LSBFIRST	DS[2:0]				Res.	PRTCFG[1:0]	MODE[1:0]		
		RW	RW	RW	RW	RW	RW	RW	RW	RW		RW	RW	RW	RW

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

Bits 23:20 MCKDIV[3:0]: Master clock divider

These bits are set and cleared by software. These bits are meaningless when the audio block operates in slave mode. They have to be configured when the audio block is disabled.

0000: Divides by 1 the master clock input.

Others: the master clock frequency is calculated accordingly to the following formula:

$$F_{SCK_x} = \frac{F_{sai_x_ker_ck}}{MCKDIV \times 2}$$

Bit 19 **NODIV**: No divider

This bit is set and cleared by software.

0: Master clock generator is enabled

1: No divider used in the clock generator (in this case Master Clock Divider bit has no effect)

Bit 18 Reserved, must be kept at reset value.

Bit 17 **DMAEN**: DMA enable

This bit is set and cleared by software.

0: DMA disabled

1: DMA enabled

Note: Since the audio block defaults to operate as a transmitter after reset, the MODE[1:0] bits must be configured before setting DMAEN to avoid a DMA request in receiver mode.

Bit 16 **SAIEN**: Audio block enable

This bit is set by software.

To switch off the audio block, the application software must program this bit to 0 and poll the bit till it reads back 0, meaning that the block is completely disabled. Before setting this bit to 1, check that it is set to 0, otherwise the enable command will not be taken into account.

This bit allows controlling the state of the SAI audio block. If it is disabled when an audio frame transfer is ongoing, the ongoing transfer completes and the cell is fully disabled at the end of this audio frame transfer.

0: SAI audio block disabled

1: SAI audio block enabled.

Note: When the SAI block (A or B) is configured in master mode, the clock must be present on the SAI block input before setting SAIEN bit.

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

Bit 13 **OUTDRIV**: Output drive

This bit is set and cleared by software.

0: Audio block output driven when SAIEN is set

1: Audio block output driven immediately after the setting of this bit.

Note: This bit has to be set before enabling the audio block and after the audio block configuration.

Bit 12 **MONO**: Mono mode

This bit is set and cleared by software. It is meaningful only when the number of slots is equal to 2.

When the mono mode is selected, slot 0 data are duplicated on slot 1 when the audio block operates as a transmitter. In reception mode, the slot1 is discarded and only the data received from slot 0 are stored. Refer to [Section : Mono/stereo mode](#) for more details.

0: Stereo mode

1: Mono mode.

Bits 11:10 SYNCEN[1:0]: Synchronization enable

These bits are set and cleared by software. They must be configured when the audio sub-block is disabled.

00: audio sub-block in asynchronous mode.

01: audio sub-block is synchronous with the other internal audio sub-block. In this case, the audio sub-block must be configured in slave mode

10: audio sub-block is synchronous with an external SAI embedded peripheral. In this case the audio sub-block should be configured in Slave mode.

11: Reserved

Note: The audio sub-block should be configured as asynchronous when SPDIF mode is enabled.

Bit 9 CKSTR: Clock strobing edge

This bit is set and cleared by software. It must be configured when the audio block is disabled. This bit has no meaning in SPDIF audio protocol.

0: Signals generated by the SAI change on SCK rising edge, while signals received by the SAI are sampled on the SCK falling edge.

1: Signals generated by the SAI change on SCK falling edge, while signals received by the SAI are sampled on the SCK rising edge.

Bit 8 LSBFIRST: Least significant bit first

This bit is set and cleared by software. It must be configured when the audio block is disabled. This bit has no meaning in AC'97 audio protocol since AC'97 data are always transferred with the MSB first. This bit has no meaning in SPDIF audio protocol since in SPDIF data are always transferred with LSB first.

0: Data are transferred with MSB first

1: Data are transferred with LSB first

Bits 7:5 DS[2:0]: Data size

These bits are set and cleared by software. These bits are ignored when the SPDIF protocols are selected (bit PRTCFS[1:0]), because the frame and the data size are fixed in such case. When the companding mode is selected through COMP[1:0] bits, DS[1:0] are ignored since the data size is fixed to 8 bits by the algorithm.

These bits must be configured when the audio block is disabled.

000: Reserved

001: Reserved

010: 8 bits

011: 10 bits

100: 16 bits

101: 20 bits

110: 24 bits

111: 32 bits

Bit 4 Reserved, must be kept at reset value.

Bits 3:2 PRTC_{FG}[1:0]: Protocol configuration

These bits are set and cleared by software. These bits have to be configured when the audio block is disabled.

00: Free protocol. Free protocol allows to use the powerful configuration of the audio block to address a specific audio protocol (such as I2S, LSB/MSB justified, TDM, PCM/DSP...) by setting most of the configuration register bits as well as frame configuration register.

01: SPDIF protocol

10: AC'97 protocol

11: Reserved

Bits 1:0 MODE[1:0]: SAI_x audio block mode

These bits are set and cleared by software. They must be configured when SAI_x audio block is disabled.

00: Master transmitter

01: Master receiver

10: Slave transmitter

11: Slave receiver

Note: When the audio block is configured in SPDIF mode, the master transmitter mode is forced (MODE[1:0] = 00).

29.5.3 Configuration register 1 (SAI_BCR1)

Address offset: 0x024

Reset value: 0x0000 0040

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	MCKDIV[3:0]				NODIV	Res.	DMAEN	SAIEN
								rw	rw	rw	rw	rw		rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	OUTD RIV	MONO	SYNCEN[1:0]		CKSTR	LSBFIRST	DS[2:0]			Res.	PRTC _{FG} [1:0]		MODE[1:0]	
		rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw

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

Bits 23:20 MCKDIV[3:0]: Master clock divider

These bits are set and cleared by software. These bits are meaningless when the audio block operates in slave mode. They have to be configured when the audio block is disabled.

0000: Divides by 1 the master clock input.

Others: the master clock frequency is calculated accordingly to the following formula:

$$F_{SCK_x} = \frac{F_{sai_x_ker_ck}}{MCKDIV \times 2}$$

Bit 19 NODIV: No divider

This bit is set and cleared by software.

0: Master clock generator is enabled

1: No divider used in the clock generator (in this case Master Clock Divider bit has no effect)

Bit 18 Reserved, must be kept at reset value.

Bit 17 DMAEN: DMA enable

This bit is set and cleared by software.

0: DMA disabled

1: DMA enabled

Note: Since the audio block defaults to operate as a transmitter after reset, the MODE[1:0] bits must be configured before setting DMAEN to avoid a DMA request in receiver mode.

Bit 16 SAIEN: Audio block enable

This bit is set by software.

To switch off the audio block, the application software must program this bit to 0 and poll the bit till it reads back 0, meaning that the block is completely disabled. Before setting this bit to 1, check that it is set to 0, otherwise the enable command will not be taken into account.

This bit allows controlling the state of the SAI audio block. If it is disabled when an audio frame transfer is ongoing, the ongoing transfer completes and the cell is fully disabled at the end of this audio frame transfer.

0: SAI audio block disabled

1: SAI audio block enabled.

Note: When the SAI block (A or B) is configured in master mode, the clock must be present on the SAI block input before setting SAIEN bit.

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

Bit 13 OUTDRV: Output drive

This bit is set and cleared by software.

0: Audio block output driven when SAIEN is set

1: Audio block output driven immediately after the setting of this bit.

Note: This bit has to be set before enabling the audio block and after the audio block configuration.

Bit 12 MONO: Mono mode

This bit is set and cleared by software. It is meaningful only when the number of slots is equal to 2. When the mono mode is selected, slot 0 data are duplicated on slot 1 when the audio block operates as a transmitter. In reception mode, the slot1 is discarded and only the data received from slot 0 are stored. Refer to [Section : Mono/stereo mode](#) for more details.

0: Stereo mode

1: Mono mode.

Bits 11:10 **SYNCEN[1:0]:** Synchronization enable

These bits are set and cleared by software. They must be configured when the audio sub-block is disabled.

00: audio sub-block in asynchronous mode.

01: audio sub-block is synchronous with the other internal audio sub-block. In this case, the audio sub-block must be configured in slave mode

10: audio sub-block is synchronous with an external SAI embedded peripheral. In this case the audio sub-block should be configured in Slave mode.

11: Reserved

Note: The audio sub-block should be configured as asynchronous when SPDIF mode is enabled.

Bit 9 CKSTR: Clock strobing edge

This bit is set and cleared by software. It must be configured when the audio block is disabled. This bit has no meaning in SPDIF audio protocol.

0: Signals generated by the SAI change on SCK rising edge, while signals received by the SAI are sampled on the SCK falling edge.

1: Signals generated by the SAI change on SCK falling edge, while signals received by the SAI are sampled on the SCK rising edge.

Bit 8 LSBFIRST: Least significant bit first

This bit is set and cleared by software. It must be configured when the audio block is disabled. This bit has no meaning in AC'97 audio protocol since AC'97 data are always transferred with the MSB first. This bit has no meaning in SPDIF audio protocol since in SPDIF data are always transferred with LSB first.

0: Data are transferred with MSB first

1: Data are transferred with LSB first

Bits 7:5 DS[2:0]: Data size

These bits are set and cleared by software. These bits are ignored when the SPDIF protocols are selected (bit PRTCFCFG[1:0]), because the frame and the data size are fixed in such case. When the companding mode is selected through COMP[1:0] bits, DS[1:0] are ignored since the data size is fixed to 8 bits by the algorithm.

These bits must be configured when the audio block is disabled.

000: Reserved

001: Reserved

010: 8 bits

011: 10 bits

100: 16 bits

101: 20 bits

110: 24 bits

111: 32 bits

Bit 4 Reserved, must be kept at reset value.**Bits 3:2 PRTCFCFG[1:0]: Protocol configuration**

These bits are set and cleared by software. These bits have to be configured when the audio block is disabled.

00: Free protocol. Free protocol allows to use the powerful configuration of the audio block to address a specific audio protocol (such as I2S, LSB/MSB justified, TDM, PCM/DSP...) by setting most of the configuration register bits as well as frame configuration register.

01: SPDIF protocol

10: AC'97 protocol

11: Reserved

Bits 1:0 MODE[1:0]: SAIx audio block mode

These bits are set and cleared by software. They must be configured when SAIx audio block is disabled.

00: Master transmitter

01: Master receiver

10: Slave transmitter

11: Slave receiver

Note: When the audio block is configured in SPDIF mode, the master transmitter mode is forced (MODE[1:0] = 00). In Master transmitter mode, the audio block starts generating the FS and the clocks immediately.

29.5.4 Configuration register 2 (SAI_ACR2)

Address offset: 0x008

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COMP[1:0]	CPL	MUTECNT[5:0]						MUTE VAL	MUTE	TRIS	F FLUSH	FTH[2:0]			
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	W	RW	RW	RW	

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

Bits 15:14 COMP[1:0]: Companding mode.

These bits are set and cleared by software. The μ -Law and the A-Law log are a part of the CCITT G.711 recommendation, the type of complement that will be used depends on CPL bit.

The data expansion or data compression are determined by the state of bit MODE[0].

The data compression is applied if the audio block is configured as a transmitter.

The data expansion is automatically applied when the audio block is configured as a receiver.

Refer to [Section : Companding mode](#) for more details.

00: No companding algorithm

01: Reserved.

10: μ -Law algorithm

11: A-Law algorithm

Note: Companding mode is applicable only when TDM is selected.

Bit 13 CPL: Complement bit.

This bit is set and cleared by software.

It defines the type of complement to be used for companding mode

0: 1's complement representation.

1: 2's complement representation.

Note: This bit has effect only when the companding mode is μ -Law algorithm or A-Law algorithm.

Bits 12:7 MUTECNT[5:0]: Mute counter.

These bits are set and cleared by software. They are used only in reception mode.

The value set in these bits is compared to the number of consecutive mute frames detected in reception. When the number of mute frames is equal to this value, the flag MUTEDET will be set and an interrupt will be generated if bit MUTEDETIE is set.

Refer to [Section : Mute mode](#) for more details.

Bit 6 MUTEVAL: Mute value.

This bit is set and cleared by software. It must be written before enabling the audio block: SAIEN.

This bit is meaningful only when the audio block operates as a transmitter, the number of slots is lower or equal to 2 and the MUTE bit is set.

If more slots are declared, the bit value sent during the transmission in mute mode is equal to 0, whatever the value of MUTEVAL.

If the number of slot is lower or equal to 2 and MUTEVAL = 1, the MUTE value transmitted for each slot is the one sent during the previous frame.

Refer to [Section : Mute mode](#) for more details.

0: Bit value 0 is sent during the mute mode.

1: Last values are sent during the mute mode.

Note: This bit is meaningless and should not be used for SPDIF audio blocks.

Bit 5 MUTE: Mute.

This bit is set and cleared by software. It is meaningful only when the audio block operates as a transmitter. The MUTE value is linked to value of MUTEVAL if the number of slots is lower or equal to 2, or equal to 0 if it is greater than 2.

Refer to [Section : Mute mode](#) for more details.

0: No mute mode.

1: Mute mode enabled.

Note: This bit is meaningless and should not be used for SPDIF audio blocks.

Bit 4 TRIS: Tristate management on data line.

This bit is set and cleared by software. It is meaningful only if the audio block is configured as a transmitter. This bit is not used when the audio block is configured in SPDIF mode. It should be configured when SAI is disabled.

Refer to [Section : Output data line management on an inactive slot](#) for more details.

0: SD output line is still driven by the SAI when a slot is inactive.

1: SD output line is released (Hi-Z) at the end of the last data bit of the last active slot if the next one is inactive.

Bit 3 FFLUSH: FIFO flush.

This bit is set by software. It is always read as 0. This bit should be configured when the SAI is disabled.

0: No FIFO flush.

1: FIFO flush. Programming this bit to 1 triggers the FIFO Flush. All the internal FIFO pointers (read and write) are cleared. In this case data still present in the FIFO are lost (no more transmission or received data lost). Before flushing, SAI DMA stream/interruption must be disabled

Bits 2:0 FTH[2:0]: FIFO threshold.

This bit is set and cleared by software.

000: FIFO empty

001: ¼ FIFO

010: ½ FIFO

011: ¾ FIFO

100: FIFO full

101: Reserved

110: Reserved

111: Reserved

29.5.5 Configuration register 2 (SAI_BCR2)

Address offset: 0x028

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COMP[1:0]	CPL	MUTECNT[5:0]						MUTEVAL	MUTE	TRIS	F FLUSH	FTH[2:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	w	rw	rw	rw	rw

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

Bits 15:14 **COMP[1:0]**: Companding mode.

These bits are set and cleared by software. The μ -Law and the A-Law log are a part of the CCITT G.711 recommendation, the type of complement that will be used depends on *CPL bit*.

The data expansion or data compression are determined by the state of bit MODE[0].

The data compression is applied if the audio block is configured as a transmitter.

The data expansion is automatically applied when the audio block is configured as a receiver.

Refer to [Section : Companding mode](#) for more details.

00: No companding algorithm

01: Reserved.

10: μ -Law algorithm

11: A-Law algorithm

Note: Companding mode is applicable only when TDM is selected.

Bit 13 **CPL**: Complement bit.

This bit is set and cleared by software.

It defines the type of complement to be used for companding mode

0: 1's complement representation.

1: 2's complement representation.

Note: This bit has effect only when the companding mode is μ -Law algorithm or A-Law algorithm.

Bits 12:7 **MUTECNT[5:0]**: Mute counter.

These bits are set and cleared by software. They are used only in reception mode.

The value set in these bits is compared to the number of consecutive mute frames detected in reception. When the number of mute frames is equal to this value, the flag MUTEDET will be set and an interrupt will be generated if bit MUTEDETIE is set.

Refer to [Section : Mute mode](#) for more details.

Bit 6 **MUTEVAL**: Mute value.

This bit is set and cleared by software. It must be written before enabling the audio block: SAIEN.

This bit is meaningful only when the audio block operates as a transmitter, the number of slots is lower or equal to 2 and the MUTE bit is set.

If more slots are declared, the bit value sent during the transmission in mute mode is equal to 0, whatever the value of MUTEVAL.

If the number of slot is lower or equal to 2 and MUTEVAL = 1, the MUTE value transmitted for each slot is the one sent during the previous frame.

Refer to [Section : Mute mode](#) for more details.

0: Bit value 0 is sent during the mute mode.

1: Last values are sent during the mute mode.

Note: This bit is meaningless and should not be used for SPDIF audio blocks.

Bit 5 **MUTE**: Mute.

This bit is set and cleared by software. It is meaningful only when the audio block operates as a transmitter. The MUTE value is linked to value of MUTEVAL if the number of slots is lower or equal to 2, or equal to 0 if it is greater than 2.

Refer to [Section : Mute mode](#) for more details.

0: No mute mode.

1: Mute mode enabled.

Note: This bit is meaningless and should not be used for SPDIF audio blocks.

Bit 4 **TRIS**: Tristate management on data line.

This bit is set and cleared by software. It is meaningful only if the audio block is configured as a transmitter. This bit is not used when the audio block is configured in SPDIF mode. It should be configured when SAI is disabled.

Refer to [Section : Output data line management on an inactive slot](#) for more details.

0: SD output line is still driven by the SAI when a slot is inactive.

1: SD output line is released (HI-Z) at the end of the last data bit of the last active slot if the next one is inactive.

Bit 3 **FFLUSH**: FIFO flush.

This bit is set by software. It is always read as 0. This bit should be configured when the SAI is disabled.

0: No FIFO flush.

1: FIFO flush. Programming this bit to 1 triggers the FIFO Flush. All the internal FIFO pointers (read and write) are cleared. In this case data still present in the FIFO are lost (no more transmission or received data lost). Before flushing, SAI DMA stream/interruption must be disabled

Bits 2:0 **FTH[2:0]**: FIFO threshold.

This bit is set and cleared by software.

000: FIFO empty

001: 1/4 FIFO

010: 1/2 FIFO

011: 3/4 FIFO

100: FIFO full

101: Reserved

110: Reserved

111: Reserved

29.5.6 Frame configuration register (SAI_AFRCR)

Address offset: 0x00C

Reset value: 0x0000 0007

Note: This register has no meaning in AC'97 and SPDIF audio protocol

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

Bit 18 FS OFF: Frame synchronization offset.

This bit is set and cleared by software. It is meaningless and is not used in AC'97 or SPDIF audio block configuration. This bit must be configured when the audio block is disabled.

0: FS is asserted on the first bit of the slot 0.

1: FS is asserted one bit before the first bit of the slot 0.

Bit 17 FSPOL: Frame synchronization polarity.

This bit is set and cleared by software. It is used to configure the level of the start of frame on the FS signal. It is meaningless and is not used in AC'97 or SPDIF audio block configuration.

This bit must be configured when the audio block is disabled.

0: FS is active low (falling edge)

1: FS is active high (rising edge)

Bit 16 FSDEF: Frame synchronization definition.

This bit is set and cleared by software.

0: FS signal is a start frame signal

1: FS signal is a start of frame signal + channel side identification

When the bit is set, the number of slots defined in the SAI_xSLOTR register has to be even. It means that half of this number of slots will be dedicated to the left channel and the other slots for the right channel (e.g: this bit has to be set for I2S or MSB/LSB-justified protocols...).

This bit is meaningless and is not used in AC'97 or SPDIF audio block configuration. It must be configured when the audio block is disabled.

Bit 15 Reserved, must be kept at reset value.

Bits 14:8 FSALL[6:0]: Frame synchronization active level length.

These bits are set and cleared by software. They specify the length in number of bit clock (SCK) + 1 (FSALL[6:0] + 1) of the active level of the FS signal in the audio frame

These bits are meaningless and are not used in AC'97 or SPDIF audio block configuration.

They must be configured when the audio block is disabled.

Bits 7:0 FRL[7:0]: Frame length.

These bits are set and cleared by software. They define the audio frame length expressed in number of SCK clock cycles: the number of bits in the frame is equal to FRL[7:0] + 1.

The minimum number of bits to transfer in an audio frame must be equal to 8, otherwise the audio block will behaves in an unexpected way. This is the case when the data size is 8 bits and only one slot 0 is defined in NBSLOT[4:0] of SAI_xSLOTR register (NBSLOT[3:0] = 0000).

In master mode, if the master clock (available on MCLK_x pin) is used, the frame length should be aligned with a number equal to a power of 2, ranging from 8 to 256. When the master clock is not used (NODIV = 1), it is recommended to program the frame length to an value ranging from 8 to 256. These bits are meaningless and are not used in AC'97 or SPDIF audio block configuration.

29.5.7 Frame configuration register (SAI_BFRCR)

Address offset: 0x02C

Reset value: 0x0000 0007

Note: This register has no meaning in AC'97 and SPDIF audio protocol

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

Bit 18 **FSOFF**: Frame synchronization offset.

This bit is set and cleared by software. It is meaningless and is not used in AC'97 or SPDIF audio block configuration. This bit must be configured when the audio block is disabled.

0: FS is asserted on the first bit of the slot 0.

1: FS is asserted one bit before the first bit of the slot 0.

Bit 17 **FSPOL**: Frame synchronization polarity.

This bit is set and cleared by software. It is used to configure the level of the start of frame on the FS signal. It is meaningless and is not used in AC'97 or SPDIF audio block configuration.

This bit must be configured when the audio block is disabled.

0: FS is active low (falling edge)

1: FS is active high (rising edge)

Bit 16 **FSDEF**: Frame synchronization definition.

This bit is set and cleared by software.

0: FS signal is a start frame signal

1: FS signal is a start of frame signal + channel side identification

When the bit is set, the number of slots defined in the SAI_xSLOTR register has to be even. It means that half of this number of slots will be dedicated to the left channel and the other slots for the right channel (e.g: this bit has to be set for I2S or MSB/LSB-justified protocols...).

This bit is meaningless and is not used in AC'97 or SPDIF audio block configuration. It must be configured when the audio block is disabled.

Bit 15 Reserved, must be kept at reset value.

Bits 14:8 **FSALL[6:0]**: Frame synchronization active level length.

These bits are set and cleared by software. They specify the length in number of bit clock (SCK) + 1 (FSALL[6:0] + 1) of the active level of the FS signal in the audio frame

These bits are meaningless and are not used in AC'97 or SPDIF audio block configuration.

They must be configured when the audio block is disabled.

Bits 7:0 **FRL[7:0]**: Frame length.

These bits are set and cleared by software. They define the audio frame length expressed in number of SCK clock cycles: the number of bits in the frame is equal to FRL[7:0] + 1.

The minimum number of bits to transfer in an audio frame must be equal to 8, otherwise the audio block will behaves in an unexpected way. This is the case when the data size is 8 bits and only one slot 0 is defined in NBSLOT[4:0] of SAI_xSLOTR register (NBSLOT[3:0] = 0000).

In master mode, if the master clock (available on MCLK_x pin) is used, the frame length should be aligned with a number equal to a power of 2, ranging from 8 to 256. When the master clock is not used (NODIV = 1), it is recommended to program the frame length to an value ranging from 8 to 256.

These bits are meaningless and are not used in AC'97 or SPDIF audio block configuration.

29.5.8 Slot register (SAI_ASLOTR)

Address offset: 0x010

Reset value: 0x0000 0000

Note: This register has no meaning in AC'97 and SPDIF audio protocol

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

Bits 31:16 **SLOTEN[15:0]**: Slot enable.

These bits are set and cleared by software.

Each SLOTEN bit corresponds to a slot position from 0 to 15 (maximum 16 slots).

0: Inactive slot.

1: Active slot.

The slot must be enabled when the audio block is disabled.

They are ignored in AC'97 or SPDIF mode.

Bits 15:12 Reserved, must be kept at reset value.

Bits 11:8 **NBSLOT[3:0]**: Number of slots in an audio frame.

These bits are set and cleared by software.

The value set in this bitfield represents the number of slots + 1 in the audio frame (including the number of inactive slots). The maximum number of slots is 16.

The number of slots should be even if FSDEF bit in the SAI_xFRCR register is set.

The number of slots must be configured when the audio block is disabled.

They are ignored in AC'97 or SPDIF mode.

Bits 7:6 **SLOTSZ[1:0]**: Slot size

This bits is set and cleared by software.

The slot size must be higher or equal to the data size. If this condition is not respected, the behavior of the SAI will be undetermined.

Refer to [Section : Output data line management on an inactive slot](#) for information on how to drive SD line.

These bits must be set when the audio block is disabled.

They are ignored in AC'97 or SPDIF mode.

00: The slot size is equivalent to the data size (specified in DS[3:0] in the SAI_xCR1 register).

01: 16-bit

10: 32-bit

11: Reserved

Bit 5 Reserved, must be kept at reset value.

Bits 4:0 **FBOFF[4:0]**: First bit offset

These bits are set and cleared by software.

The value set in this bitfield defines the position of the first data transfer bit in the slot. It represents an offset value. In transmission mode, the bits outside the data field are forced to 0. In reception mode, the extra received bits are discarded.

These bits must be set when the audio block is disabled.

They are ignored in AC'97 or SPDIF mode.

29.5.9 Slot register (SAI_BSLOTR)

Address offset: 0x030

Reset value: 0x0000 0000

Note: This register has no meaning in AC'97 and SPDIF audio protocol

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

Bits 31:16 **SLOTEN[15:0]**: Slot enable.

These bits are set and cleared by software.

Each SLOTEN bit corresponds to a slot position from 0 to 15 (maximum 16 slots).

0: Inactive slot.

1: Active slot.

The slot must be enabled when the audio block is disabled.

They are ignored in AC'97 or SPDIF mode.

Bits 15:12 Reserved, must be kept at reset value.

Bits 11:8 **NBSLOT[3:0]**: Number of slots in an audio frame.

These bits are set and cleared by software.

The value set in this bitfield represents the number of slots + 1 in the audio frame (including the number of inactive slots). The maximum number of slots is 16.

The number of slots should be even if FSDEF bit in the SAI_xFRCR register is set.

The number of slots must be configured when the audio block is disabled.

They are ignored in AC'97 or SPDIF mode.

Bits 7:6 **SLOTSZ[1:0]**: Slot size

This bits is set and cleared by software.

The slot size must be higher or equal to the data size. If this condition is not respected, the behavior of the SAI will be undetermined.

Refer to [Section : Output data line management on an inactive slot](#) for information on how to drive SD line.

These bits must be set when the audio block is disabled.

They are ignored in AC'97 or SPDIF mode.

00: The slot size is equivalent to the data size (specified in DS[3:0] in the SAI_xCR1 register).

01: 16-bit

10: 32-bit

11: Reserved

Bit 5 Reserved, must be kept at reset value.

Bits 4:0 **FBOFF[4:0]**: First bit offset

These bits are set and cleared by software.

The value set in this bitfield defines the position of the first data transfer bit in the slot. It represents an offset value. In transmission mode, the bits outside the data field are forced to 0. In reception mode, the extra received bits are discarded.

These bits must be set when the audio block is disabled.

They are ignored in AC'97 or SPDIF mode.

29.5.10 Interrupt mask register (SAI_AIM)

Address offset: 0x014

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LFSDET IE	AFSDET IE	CNRDY IE	FREQ IE	WCKCFG IE	MUTEDET IE	OVRUDR IE								
									rw	rw	rw	rw	rw	rw	rw

Bits 31:7 Reserved, must be kept at reset value.

Bit 6 **LFSDETIE**: Late frame synchronization detection interrupt enable.

This bit is set and cleared by software.

0: Interrupt is disabled

1: Interrupt is enabled

When this bit is set, an interrupt will be generated if the LFSDET bit is set in the SAI_xSR register.

This bit is meaningless in AC'97, SPDIF mode or when the audio block operates as a master.

Bit 5 **AFSDETIE**: Anticipated frame synchronization detection interrupt enable.

This bit is set and cleared by software.

0: Interrupt is disabled

1: Interrupt is enabled

When this bit is set, an interrupt will be generated if the AFSDET bit in the SAI_xSR register is set.

This bit is meaningless in AC'97, SPDIF mode or when the audio block operates as a master.

Bit 4 **CNRDYIE**: Codec not ready interrupt enable (AC'97).

This bit is set and cleared by software.

0: Interrupt is disabled

1: Interrupt is enabled

When the interrupt is enabled, the audio block detects in the slot 0 (tag0) of the AC'97 frame if the Codec connected to this line is ready or not. If it is not ready, the CNRDY flag in the SAI_xSR register is set and an interruption is generated.

This bit has a meaning only if the AC'97 mode is selected through PRTCFG[1:0] bits and the audio block is operating as a receiver.

Bit 3 **FREQIE**: FIFO request interrupt enable.

This bit is set and cleared by software.

0: Interrupt is disabled

1: Interrupt is enabled

When this bit is set, an interrupt is generated if the FREQ bit in the SAI_xSR register is set.

Since the audio block defaults to operate as a transmitter after reset, the MODE bit must be configured before setting FREQIE to avoid a parasitic interruption in receiver mode,

Bit 2 **WCKCFGIE**: Wrong clock configuration interrupt enable.

This bit is set and cleared by software.

0: Interrupt is disabled

1: Interrupt is enabled

This bit is taken into account only if the audio block is configured as a master (MODE[1] = 0) and NODIV = 0.

It generates an interrupt if the WCKCFG flag in the SAI_xSR register is set.

Note: This bit is used only in TDM mode and is meaningless in other modes.

Bit 1 **MUTEDETIE**: Mute detection interrupt enable.

This bit is set and cleared by software.

0: Interrupt is disabled

1: Interrupt is enabled

When this bit is set, an interrupt is generated if the MUTEDET bit in the SAI_xSR register is set.

This bit has a meaning only if the audio block is configured in receiver mode.

Bit 0 **OVRUDRIE**: Overrun/underrun interrupt enable.

This bit is set and cleared by software.

0: Interrupt is disabled

1: Interrupt is enabled

When this bit is set, an interrupt is generated if the OVRUDR bit in the SAI_xSR register is set.

29.5.11 Interrupt mask register (SAI_BIM)

Address offset: 0x034

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LFSDET IE	AFSDET IE	CNRDY IE	FREQ IE	WCKCFG IE	MUTEDET IE	OVRUDR IE								
									rw	rw	rw	rw	rw	rw	rw

Bits 31:7 Reserved, must be kept at reset value.

Bit 6 **LFSDETIE**: Late frame synchronization detection interrupt enable.

This bit is set and cleared by software.

0: Interrupt is disabled

1: Interrupt is enabled

When this bit is set, an interrupt will be generated if the LFSDET bit is set in the SAI_xSR register.

This bit is meaningless in AC'97, SPDIF mode or when the audio block operates as a master.

Bit 5 **AFSDETIE**: Anticipated frame synchronization detection interrupt enable.

This bit is set and cleared by software.

0: Interrupt is disabled

1: Interrupt is enabled

When this bit is set, an interrupt will be generated if the AFSDET bit in the SAI_xSR register is set.

This bit is meaningless in AC'97, SPDIF mode or when the audio block operates as a master.

Bit 4 **CNRDYIE**: Codec not ready interrupt enable (AC'97).

This bit is set and cleared by software.

0: Interrupt is disabled

1: Interrupt is enabled

When the interrupt is enabled, the audio block detects in the slot 0 (tag0) of the AC'97 frame if the Codec connected to this line is ready or not. If it is not ready, the CNRDY flag in the SAI_xSR register is set and an interruption is generated.

This bit has a meaning only if the AC'97 mode is selected through PRTC[1:0] bits and the audio block is operating as a receiver.

Bit 3 **FREQIE**: FIFO request interrupt enable.

This bit is set and cleared by software.

0: Interrupt is disabled

1: Interrupt is enabled

When this bit is set, an interrupt is generated if the FREQ bit in the SAI_xSR register is set.

Since the audio block defaults to operate as a transmitter after reset, the MODE bit must be configured before setting FREQIE to avoid a parasitic interruption in receiver mode,

Bit 2 **WCKCFGIE**: Wrong clock configuration interrupt enable.

This bit is set and cleared by software.

0: Interrupt is disabled

1: Interrupt is enabled

This bit is taken into account only if the audio block is configured as a master (MODE[1] = 0) and NODIV = 0.

It generates an interrupt if the WCKCFG flag in the SAI_xSR register is set.

Note: This bit is used only in TDM mode and is meaningless in other modes.

Bit 1 **MUTEDETIE**: Mute detection interrupt enable.

This bit is set and cleared by software.

0: Interrupt is disabled

1: Interrupt is enabled

When this bit is set, an interrupt is generated if the MUTEDET bit in the SAI_xSR register is set.

This bit has a meaning only if the audio block is configured in receiver mode.

Bit 0 **OVRUDRIE**: Overrun/underrun interrupt enable.

This bit is set and cleared by software.

0: Interrupt is disabled

1: Interrupt is enabled

When this bit is set, an interrupt is generated if the OVRUDR bit in the SAI_xSR register is set.

29.5.12 Status register (SAI_ASР)

Address offset: 0x018

Reset value: 0x0000 0008

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	FLVL[2:0]											
													r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LFSDET	AFSDET	CNRDY	FREQ	WCKCFG	MUTEDET	OVRUDR								
									r	r	r	r	r	r	r

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

Bits 18:16 **FLVL[2:0]**: FIFO level threshold.

This bit is read only. The FIFO level threshold flag is managed only by hardware and its setting depends on SAI block configuration (transmitter or receiver mode).

If the SAI block is configured as transmitter:

- 000: FIFO empty
- 001: FIFO $\leq \frac{1}{4}$ but not empty
- 010: $\frac{1}{4} < \text{FIFO} \leq \frac{1}{2}$
- 011: $\frac{1}{2} < \text{FIFO} \leq \frac{3}{4}$
- 100: $\frac{3}{4} < \text{FIFO}$ but not full
- 101: FIFO full

If SAI block is configured as receiver:

- 000: FIFO empty
- 001: FIFO $< \frac{1}{4}$ but not empty
- 010: $\frac{1}{4} \leq \text{FIFO} < \frac{1}{2}$
- 011: $\frac{1}{2} \leq \text{FIFO} < \frac{3}{4}$
- 100: $\frac{3}{4} \leq \text{FIFO}$ but not full
- 101: FIFO full

Bits 15:7 Reserved, must be kept at reset value.

Bit 6 **LFSDET**: Late frame synchronization detection.

This bit is read only.

0: No error.

1: Frame synchronization signal is not present at the right time.

This flag can be set only if the audio block is configured in slave mode.

It is not used in AC'97 or SPDIF mode.

It can generate an interrupt if LFSDETIE bit is set in the SAI_xIM register.

This flag is cleared when the software sets bit CLFSDET in SAI_xCLRFR register

Bit 5 **AFSDET**: Anticipated frame synchronization detection.

This bit is read only.

0: No error.

1: Frame synchronization signal is detected earlier than expected.

This flag can be set only if the audio block is configured in slave mode.

It is not used in AC'97 or SPDIF mode.

It can generate an interrupt if AFSDETIE bit is set in SAI_xIM register.

This flag is cleared when the software sets CAFSDET bit in SAI_xCLRFR register.

Bit 4 **CNRDY**: Codec not ready.

This bit is read only.

0: External AC'97 Codec is ready

1: External AC'97 Codec is not ready

This bit is used only when the AC'97 audio protocol is selected in the SAI_xCR1 register and configured in receiver mode.

It can generate an interrupt if CNRDYIE bit is set in SAI_xIM register.

This flag is cleared when the software sets CCNRDY bit in SAI_xCLRFR register.

Bit 3 **FREQ**: FIFO request.

This bit is read only.

0: No FIFO request.

1: FIFO request to read or to write the SAI_xDR.

The request depends on the audio block configuration:

- If the block is configured in transmission mode, the FIFO request is related to a write request operation in the SAI_xDR.
- If the block configured in reception, the FIFO request related to a read request operation from the SAI_xDR.

This flag can generate an interrupt if FREQIE bit is set in SAI_xIM register.

Bit 2 **WCKCFG**: Wrong clock configuration flag.

This bit is read only.

0: Clock configuration is correct

1: Clock configuration does not respect the rule concerning the frame length specification defined in [Section 29.3.6: Frame synchronization](#) (configuration of FRL[7:0] bit in the SAI_xFRCR register)

This bit is used only when the audio block operates in master mode (MODE[1] = 0) and NODIV = 0.

It can generate an interrupt if WCKCFGIE bit is set in SAI_xIM register.

This flag is cleared when the software sets CWCKCFG bit in SAI_xCLRFR register.

Bit 1 **MUTEDET**: Mute detection.

This bit is read only.

0: No MUTE detection on the SD input line

1: MUTE value detected on the SD input line (0 value) for a specified number of consecutive audio frame

This flag is set if consecutive 0 values are received in each slot of a given audio frame and for a consecutive number of audio frames (set in the MUTECNT bit in the SAI_xCR2 register).

It can generate an interrupt if MUTEDETIE bit is set in SAI_xIM register.

This flag is cleared when the software sets CMUTEDET in the SAI_xCLRFR register.

Bit 0 **OVRUDR**: Overrun / underrun.

This bit is read only.

0: No overrun/underrun error.

1: Overrun/underrun error detection.

The overrun and underrun conditions can occur only when the audio block is configured as a receiver and a transmitter, respectively.

It can generate an interrupt if OVRUDRIE bit is set in SAI_xIM register.

This flag is cleared when the software sets COVRUDR bit in SAI_xCLRFR register.

29.5.13 Status register (SAI_BSR)

Address offset: 0x038

Reset value: 0x0000 0008

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	FLVL[2:0]		
Res.	Res.	Res.	Res.	Res.	Res.	Res.	FLVL[2:0]											
																r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Res.	Res.	Res.
Res.	LFSDET	AFSDET	CNRDY	FREQ	WCKCFG	MUTEDET	OVRUDR											
									r	r	r	r	r	r	r			

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

Bits 18:16 **FLVL[2:0]**: FIFO level threshold.

This bit is read only. The FIFO level threshold flag is managed only by hardware and its setting depends on SAI block configuration (transmitter or receiver mode).

If the SAI block is configured as transmitter:

- 000: FIFO empty
- 001: FIFO $\leq \frac{1}{4}$ but not empty
- 010: $\frac{1}{4} < \text{FIFO} \leq \frac{1}{2}$
- 011: $\frac{1}{2} < \text{FIFO} \leq \frac{3}{4}$
- 100: $\frac{3}{4} < \text{FIFO}$ but not full
- 101: FIFO full

If SAI block is configured as receiver:

- 000: FIFO empty
- 001: FIFO $< \frac{1}{4}$ but not empty
- 010: $\frac{1}{4} \leq \text{FIFO} < \frac{1}{2}$
- 011: $\frac{1}{2} \leq \text{FIFO} < \frac{3}{4}$
- 100: $\frac{3}{4} \leq \text{FIFO}$ but not full
- 101: FIFO full

Bits 15:7 Reserved, must be kept at reset value.

Bit 6 **LFSDET**: Late frame synchronization detection.

This bit is read only.

0: No error.

1: Frame synchronization signal is not present at the right time.

This flag can be set only if the audio block is configured in slave mode.

It is not used in AC'97 or SPDIF mode.

It can generate an interrupt if LFSDETIE bit is set in the SAI_xIM register.

This flag is cleared when the software sets bit CLFSDET in SAI_xCLRFR register

Bit 5 **AFSDET**: Anticipated frame synchronization detection.

This bit is read only.

0: No error.

1: Frame synchronization signal is detected earlier than expected.

This flag can be set only if the audio block is configured in slave mode.

It is not used in AC'97 or SPDIF mode.

It can generate an interrupt if AFSDETIE bit is set in SAI_xIM register.

This flag is cleared when the software sets CAFSDET bit in SAI_xCLRFR register.

Bit 4 **CNRDY**: Codec not ready.

This bit is read only.

0: External AC'97 Codec is ready

1: External AC'97 Codec is not ready

This bit is used only when the AC'97 audio protocol is selected in the SAI_xCR1 register and configured in receiver mode.

It can generate an interrupt if CNRDYIE bit is set in SAI_xIM register.

This flag is cleared when the software sets CCNRDY bit in SAI_xCLRFR register.

Bit 3 **FREQ**: FIFO request.

This bit is read only.

0: No FIFO request.

1: FIFO request to read or to write the SAI_xDR.

The request depends on the audio block configuration:

- If the block is configured in transmission mode, the FIFO request is related to a write request operation in the SAI_xDR.
- If the block configured in reception, the FIFO request related to a read request operation from the SAI_xDR.

This flag can generate an interrupt if FREQIE bit is set in SAI_xIM register.

Bit 2 **WCKCFG**: Wrong clock configuration flag.

This bit is read only.

0: Clock configuration is correct

1: Clock configuration does not respect the rule concerning the frame length specification defined in [Section 29.3.6: Frame synchronization](#) (configuration of FRL[7:0] bit in the SAI_xFRCR register)

This bit is used only when the audio block operates in master mode (MODE[1] = 0) and NODIV = 0.

It can generate an interrupt if WCKCFGIE bit is set in SAI_xIM register.

This flag is cleared when the software sets CWCKCFG bit in SAI_xCLRFR register.

Bit 1 **MUTEDET**: Mute detection.

This bit is read only.

0: No MUTE detection on the SD input line

1: MUTE value detected on the SD input line (0 value) for a specified number of consecutive audio frame

This flag is set if consecutive 0 values are received in each slot of a given audio frame and for a consecutive number of audio frames (set in the MUTECNT bit in the SAI_xCR2 register).

It can generate an interrupt if MUTEDETIE bit is set in SAI_xIM register.

This flag is cleared when the software sets bit CMUTEDET in the SAI_xCLRFR register.

Bit 0 **OVRUDR**: Overrun / underrun.

This bit is read only.

0: No overrun/underrun error.

1: Overrun/underrun error detection.

The overrun and underrun conditions can occur only when the audio block is configured as a receiver and a transmitter, respectively.

It can generate an interrupt if OVRUDRIE bit is set in SAI_xIM register.

This flag is cleared when the software sets COVRUDR bit in SAI_xCLRFR register.

29.5.14 Clear flag register (SAI_ACLRFR)

Address offset: 0x01C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CLFSDET	CAFSDET	CCNRDY	Res.	CWCKCFG	CMUTEDET	COVRUDR								
									w	w	w		w	w	w

Bits 31:7 Reserved, must be kept at reset value.

Bit 6 **CLFSDET**: Clear late frame synchronization detection flag.

This bit is write only.

Programming this bit to 1 clears the LFSDET flag in the SAI_xSR register.

This bit is not used in AC'97 or SPDIF mode

Reading this bit always returns the value 0.

Bit 5 **CAFSDET**: Clear anticipated frame synchronization detection flag.

This bit is write only.

Programming this bit to 1 clears the AFSDET flag in the SAI_xSR register.

It is not used in AC'97 or SPDIF mode.

Reading this bit always returns the value 0.

Bit 4 **CCNRDY**: Clear Codec not ready flag.

This bit is write only.

Programming this bit to 1 clears the CNRDY flag in the SAI_xSR register.

This bit is used only when the AC'97 audio protocol is selected in the SAI_xCR1 register.

Reading this bit always returns the value 0.

Bit 3 Reserved, must be kept at reset value.

Bit 2 **CWCKCFG**: Clear wrong clock configuration flag.

This bit is write only.

Programming this bit to 1 clears the WCKCFG flag in the SAI_xSR register.

This bit is used only when the audio block is set as master (MODE[1] = 0) and NODIV = 0 in the SAI_xCR1 register.

Reading this bit always returns the value 0.

Bit 1 **CMUTEDET**: Mute detection flag.

This bit is write only.

Programming this bit to 1 clears the MUTEDET flag in the SAI_xSR register.

Reading this bit always returns the value 0.

Bit 0 **COVRUDR**: Clear overrun / underrun.

This bit is write only.

Programming this bit to 1 clears the OVRUDR flag in the SAI_xSR register.

Reading this bit always returns the value 0.

29.5.15 Clear flag register (SAI_BCLRFR)

Address offset: 0x03C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CLFSDET	CAFSDET	CCNRDY	Res.	CWCKCFG	CMUTEDET	COVRUDR								
									w	w	w		w	w	w

Bits 31:7 Reserved, must be kept at reset value.

Bit 6 **CLFSDET**: Clear late frame synchronization detection flag.

This bit is write only.

Programming this bit to 1 clears the LFSDET flag in the SAI_xSR register.

This bit is not used in AC'97 or SPDIF mode

Reading this bit always returns the value 0.

Bit 5 **CAFSDET**: Clear anticipated frame synchronization detection flag.

This bit is write only.

Programming this bit to 1 clears the AFSDET flag in the SAI_xSR register.

It is not used in AC'97 or SPDIF mode.

Reading this bit always returns the value 0.

Bit 4 **CCNRDY**: Clear Codec not ready flag.

This bit is write only.

Programming this bit to 1 clears the CNRDY flag in the SAI_xSR register.

This bit is used only when the AC'97 audio protocol is selected in the SAI_xCR1 register.

Reading this bit always returns the value 0.

Bit 3 Reserved, must be kept at reset value.

Bit 2 **CWCKCFG**: Clear wrong clock configuration flag.

This bit is write only.

Programming this bit to 1 clears the WCKCFG flag in the SAI_xSR register.

This bit is used only when the audio block is set as master (MODE[1] = 0) and NODIV = 0 in the SAI_xCR1 register.

Reading this bit always returns the value 0.

Bit 1 **CMUTEDET**: Mute detection flag.

This bit is write only.

Programming this bit to 1 clears the MUTEDET flag in the SAI_xSR register.

Reading this bit always returns the value 0.

Bit 0 **COVRUDR**: Clear overrun / underrun.

This bit is write only.

Programming this bit to 1 clears the OVRUDR flag in the SAI_xSR register.

Reading this bit always returns the value 0.

29.5.16 Data register (SAI_ADR)

Address offset: 0x020

Reset value: 0x0000 0000

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

Bits 31:0 **DATA[31:0]**: Data

A write to this register loads the FIFO provided the FIFO is not full.

A read from this register empties the FIFO if the FIFO is not empty.

29.5.17 Data register (SAI_BDR)

Address offset: 0x040

Reset value: 0x0000 0000

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

Bits 31:0 **DATA[31:0]**: Data

A write to this register loads the FIFO provided the FIFO is not full.

A read from this register empties the FIFO if the FIFO is not empty.

29.5.18 SAI register map

The following table summarizes the SAI registers.

Table 183. SAI register map and reset values

Offset	Register name	Reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10
0x0000	SAI_GCR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0004 or 0x0024	SAI_xCR1	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0008 or 0x0028	SAI_xCR2	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x000C or 0x002C	SAI_xFRCR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0010 or 0x0030	SAI_xSLOTR	SLOTEN[15:0]															MUTECN[5:0]							
	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
0x0014 or 0x0034	SAI_xIM	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	COMP[1:0]	0	0	0	0	0	0
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CPL	0	0	0	0	0	0
0x0018 or 0x0038	SAI_xSR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	FSALL[6:0]						FRL[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
0x001C or 0x003C	SAI_xCLRFR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	NBSLOT[3:0]	0	0	0	0	0	0
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SLOTSZ[1:0]	0	0	0	0	0	0
0x0020 or 0x0040	SAI_xDR	DATA[31:0]																						
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MUTE VAL	0	0	0	0	0	0

Refer to [Section 1.5 on page 55](#) for the register boundary addresses.

30 SD/SDIO/MMC card host interface (SDMMC)

30.1 SDMMC main features

The SD/SDIO MMC card host interface (SDMMC) provides an interface between the APB2 peripheral bus and MultiMediaCards (MMCs), SD memory cards and SDIO cards.

The MultiMediaCard system specifications are available through the MultiMediaCard Association website, published by the MMCA technical committee.

SD memory card and SD I/O card system specifications are available through the SD card Association website.

The SDMMC features include the following:

- Full compliance with *MultiMediaCard System Specification Version 4.2*. Card support for three different databus modes: 1-bit (default), 4-bit and 8-bit
- Full compatibility with previous versions of MultiMediaCards (forward compatibility)
- Full compliance with *SD Memory Card Specifications Version 2.0*
- Full compliance with *SD I/O Card Specification Version 2.0*: card support for two different databus modes: 1-bit (default) and 4-bit
- Data transfer up to 50 MHz for the 8 bit mode
- Data and command output enable signals to control external bidirectional drivers.

- Note: 1 *The SDMMC does not have an SPI-compatible communication mode.*
- 2 *The SD memory card protocol is a superset of the MultiMediaCard protocol as defined in the MultiMediaCard system specification V2.11. Several commands required for SD memory devices are not supported by either SD I/O-only cards or the I/O portion of combo cards. Some of these commands have no use in SD I/O devices, such as erase commands, and thus are not supported in the SDIO protocol. In addition, several commands are different between SD memory cards and SD I/O cards and thus are not supported in the SDIO protocol. For details refer to SD I/O card Specification Version 1.0.*

The MultiMediaCard/SD bus connects cards to the controller.

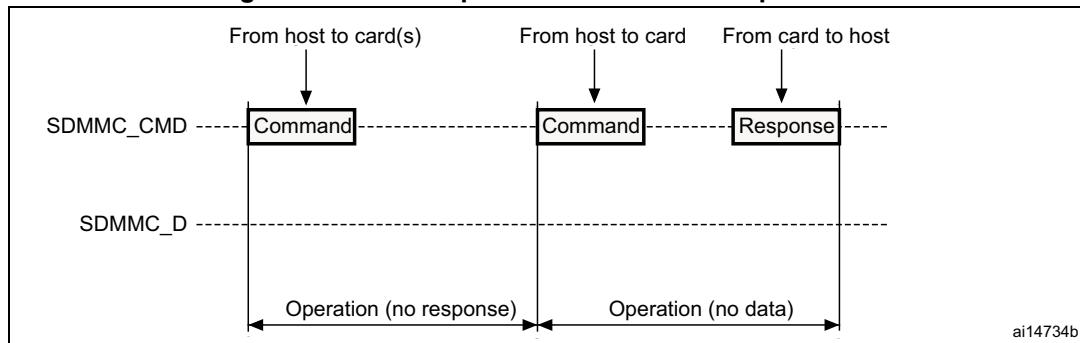
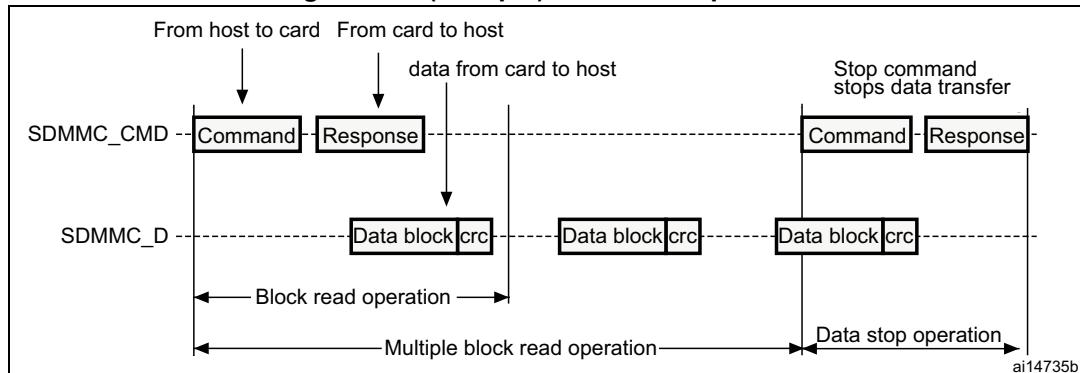
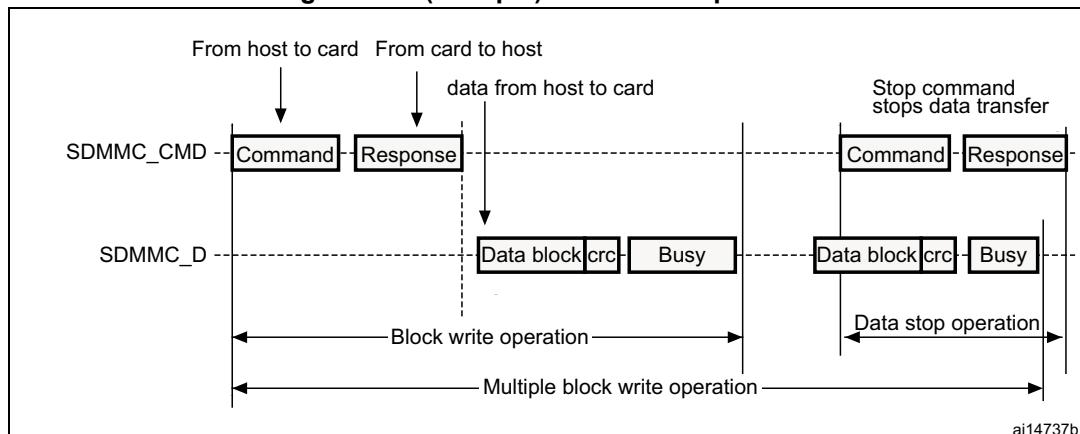
The current version of the SDMMC supports only one SD/SDIO/MMC4.2 card at any one time and a stack of MMC4.1 or previous.

30.2 SDMMC bus topology

Communication over the bus is based on command and data transfers.

The basic transaction on the MultiMediaCard/SD/SD I/O bus is the command/response transaction. These types of bus transaction transfer their information directly within the command or response structure. In addition, some operations have a data token.

Data transfers to/from SD/SDIO memory cards are done in data blocks. Data transfers to/from MMC are done data blocks or streams.

Figure 396. “No response” and “no data” operations**Figure 397. (Multiple) block read operation****Figure 398. (Multiple) block write operation**

Note: The SDMMC will not send any data as long as the Busy signal is asserted (SDMMC_D0 pulled low).

Figure 399. Sequential read operation

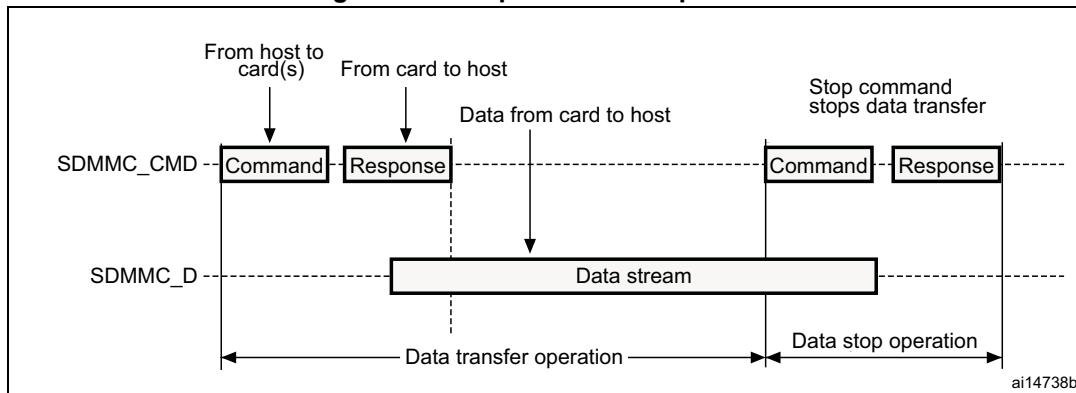
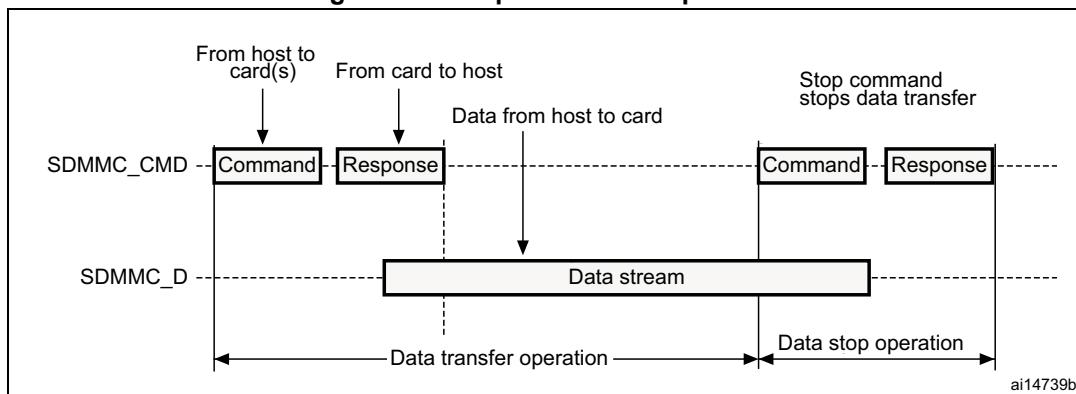


Figure 400. Sequential write operation

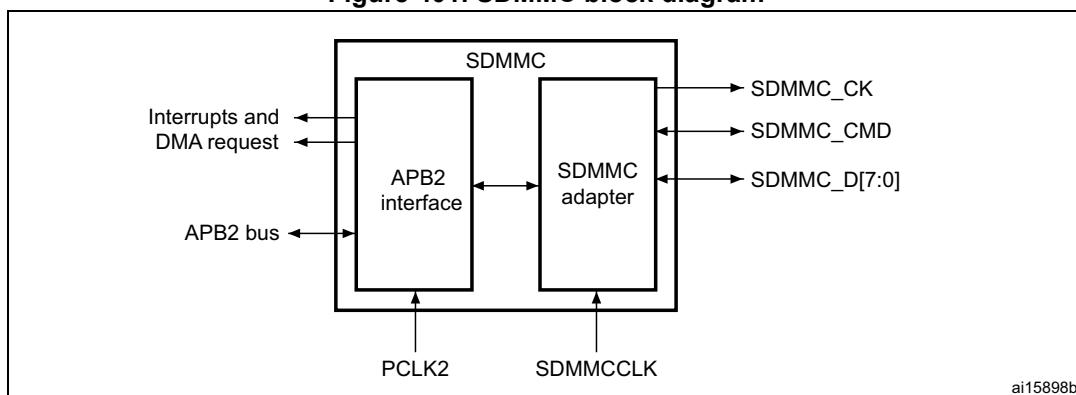


30.3 SDMMC functional description

The SDMMC consists of two parts:

- The SDMMC adapter block provides all functions specific to the MMC/SD/SD I/O card such as the clock generation unit, command and data transfer.
- The APB2 interface accesses the SDMMC adapter registers, and generates interrupt and DMA request signals.

Figure 401. SDMMC block diagram



By default SDMMC_D0 is used for data transfer. After initialization, the host can change the databus width.

If a MultiMediaCard is connected to the bus, SDMMC_D0, SDMMC_D[3:0] or SDMMC_D[7:0] can be used for data transfer. MMC V3.31 or previous, supports only 1 bit of data so only SDMMC_D0 can be used.

If an SD or SD I/O card is connected to the bus, data transfer can be configured by the host to use SDMMC_D0 or SDMMC_D[3:0]. All data lines are operating in push-pull mode.

SDMMC_CMD has two operational modes:

- Open-drain for initialization (only for MMCV3.31 or previous)
- Push-pull for command transfer (SD/SD I/O card MMC4.2 use push-pull drivers also for initialization)

SDMMC_CK is the clock to the card: one bit is transferred on both command and data lines with each clock cycle.

The SDMMC uses two clock signals:

- SDMMC adapter clock SDMMCCLK = 50 MHz)
- APB2 bus clock (PCLK2)

PCLK2 and SDMMC_CK clock frequencies must respect the following condition:

$$\text{Frequenc(PCLK2)} > ((3 \times \text{Width}) / 32) \times \text{Frequency(SDMMC_CK)}$$

The signals shown in [Table 184](#) are used on the MultiMediaCard/SD/SD I/O card bus.

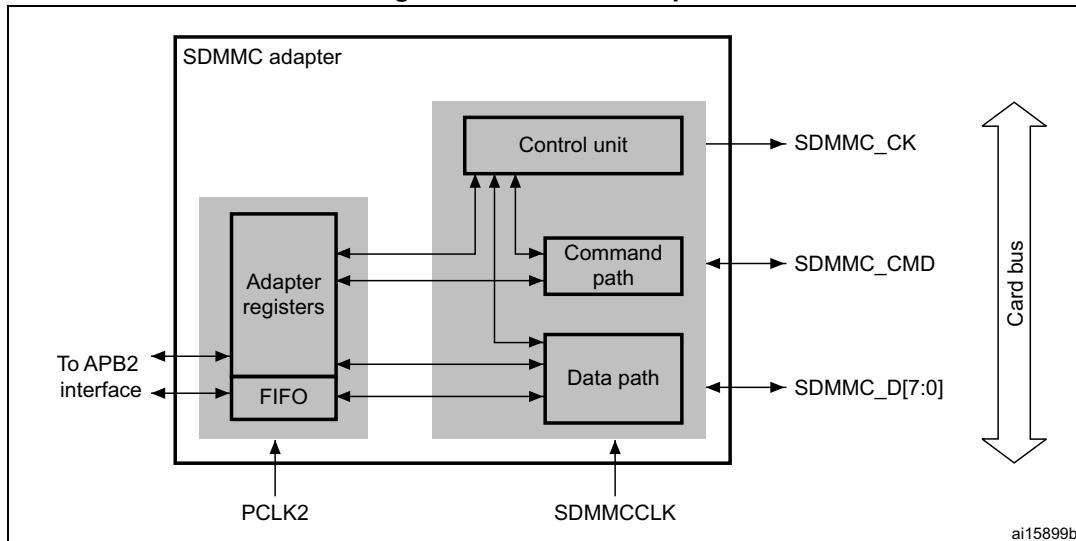
Table 184. SDMMC I/O definitions

Pin	Direction	Description
SDMMC_CK	Output	MultiMediaCard/SD/SDIO card clock. This pin is the clock from host to card.
SDMMC_CMD	Bidirectional	MultiMediaCard/SD/SDIO card command. This pin is the bidirectional command/response signal.
SDMMC_D[7:0]	Bidirectional	MultiMediaCard/SD/SDIO card data. These pins are the bidirectional databus.

30.3.1 SDMMC adapter

Figure 402 shows a simplified block diagram of an SDMMC adapter.

Figure 402. SDMMC adapter



ai15899b

The SDMMC adapter is a multimedia/secure digital memory card bus master that provides an interface to a multimedia card stack or to a secure digital memory card. It consists of five subunits:

- Adapter register block
- Control unit
- Command path
- Data path
- Data FIFO

Note:

The adapter registers and FIFO use the APB2 bus clock domain (PCLK2). The control unit, command path and data path use the SDMMC adapter clock domain (SDMMCCLK).

Adapter register block

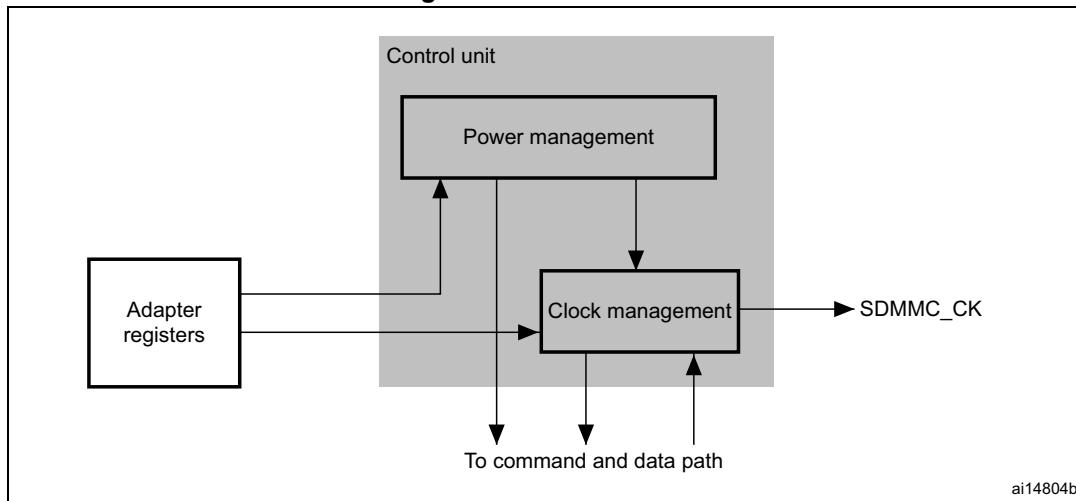
The adapter register block contains all system registers. This block also generates the signals that clear the static flags in the multimedia card. The clear signals are generated when 1 is written into the corresponding bit location in the SDMMC Clear register.

Control unit

The control unit contains the power management functions and the clock divider for the memory card clock.

There are three power phases:

- power-off
- power-up
- power-on

Figure 403. Control unit

ai14804b

The control unit is illustrated in [Figure 403](#). It consists of a power management subunit and a clock management subunit.

The power management subunit disables the card bus output signals during the power-off and power-up phases.

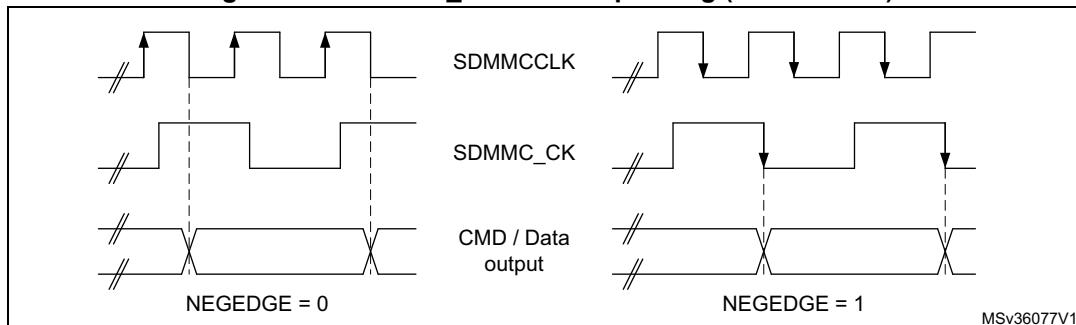
The clock management subunit generates and controls the SDMMC_CK signal. The SDMMC_CK output can use either the clock divide or the clock bypass mode. The clock output is inactive:

- after reset
- during the power-off or power-up phases
- if the power saving mode is enabled and the card bus is in the Idle state (eight clock periods after both the command and data path subunits enter the Idle phase)

The clock management subunit controls SDMMC_CK dephasing. When not in bypass mode the SDMMC command and data output are generated on the SDMMCCCLK falling edge succeeding the rising edge of SDMMC_CK. (SDMMC_CK rising edge occurs on SDMMCCCLK rising edge) when SDMMC_CLKCR[13] bit is reset (NEGEDGE = 0). When SDMMC_CLKCR[13] bit is set (NEGEDGE = 1) SDMMC command and data changed on the SDMMC_CK falling edge.

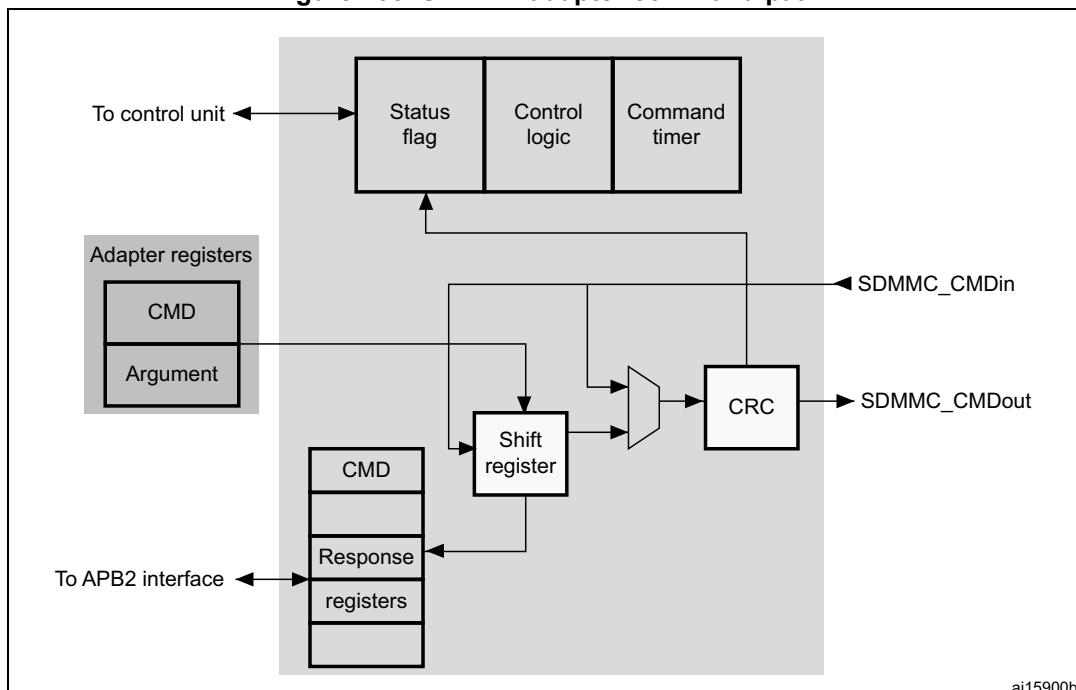
When SDMMC_CLKCR[10] is set (BYPASS = 1), SDMMC_CK rising edge occurs on SDMMCCCLK rising edge. The data and the command change on SDMMCCCLK falling edge whatever NEGEDGE value.

The data and command responses are latched using SDMMC_CK rising edge.

Figure 404. SDMMC_CK clock dephasing (BYPASS = 0)

Command path

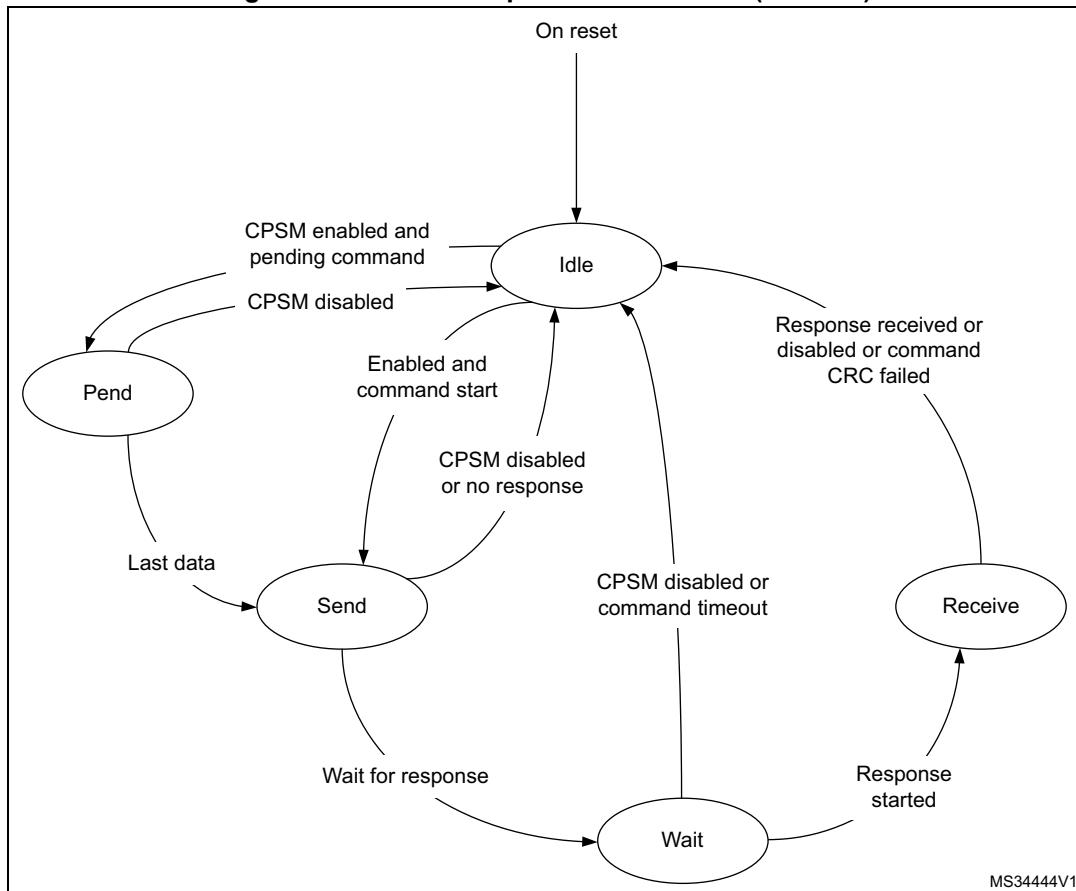
The command path unit sends commands to and receives responses from the cards.

Figure 405. SDMMC adapter command path

ai15900b

- Command path state machine (CPSM)
 - When the command register is written to and the enable bit is set, command transfer starts. When the command has been sent, the command path state machine (CPSM) sets the status flags and enters the Idle state if a response is not required. If a response is required, it waits for the response (see [Figure 406 on page 1071](#)). When the response is received, the received CRC code and the internally generated code are compared, and the appropriate status flags are set.

Figure 406. Command path state machine (SDMMC)



When the Wait state is entered, the command timer starts running. If the timeout is reached before the CPSM moves to the Receive state, the timeout flag is set and the Idle state is entered.

Note:

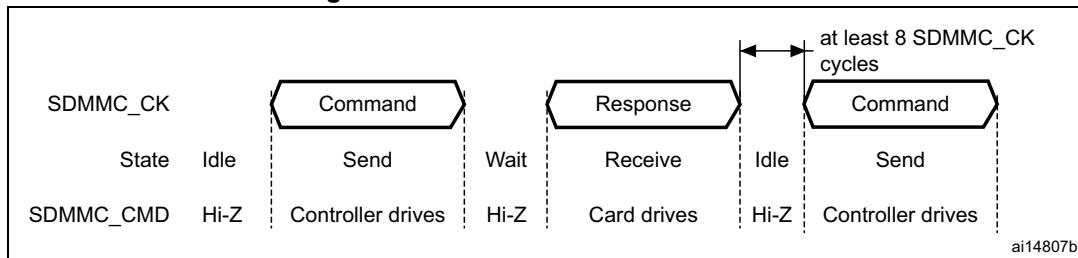
The command timeout has a fixed value of 64 SDMMC_CK clock periods.

If the interrupt bit is set in the command register, the timer is disabled and the CPSM waits for an interrupt request from one of the cards. If a pending bit is set in the command register, the CPSM enters the Pend state, and waits for a CmdPend signal from the data path subunit. When CmdPend is detected, the CPSM moves to the Send state. This enables the data counter to trigger the stop command transmission.

Note:

The CPSM remains in the Idle state for at least eight SDMMC_CK periods to meet the N_{CC} and N_{RC} timing constraints. N_{CC} is the minimum delay between two host commands, and N_{RC} is the minimum delay between the host command and the card response.

Figure 407. SDMMC command transfer



ai14807b

- Command format

- Command: a command is a token that starts an operation. Commands are sent from the host either to a single card (addressed command) or to all connected cards (broadcast command are available for MMC V3.31 or previous). Commands are transferred serially on the CMD line. All commands have a fixed length of 48 bits. The general format for a command token for MultiMediaCards, SD-Memory cards and SDIO-Cards is shown in [Table 185](#).

The command path operates in a half-duplex mode, so that commands and responses can either be sent or received. If the CPSM is not in the Send state, the SDMMC_CMD output is in the Hi-Z state, as shown in [Figure 407 on page 1072](#). Data on SDMMC_CMD are synchronous with the rising edge of SDMMC_CK. [Table 185](#) shows the command format.

Table 185. Command format

Bit position	Width	Value	Description
47	1	0	Start bit
46	1	1	Transmission bit
[45:40]	6	-	Command index
[39:8]	32	-	Argument
[7:1]	7	-	CRC7
0	1	1	End bit

- Response: a response is a token that is sent from an addressed card (or synchronously from all connected cards for MMC V3.31 or previous), to the host as an answer to a previously received command. Responses are transferred serially on the CMD line.

The SDMMC supports two response types. Both use CRC error checking:

- 48 bit short response
- 136 bit long response

Note: *If the response does not contain a CRC (CMD1 response), the device driver must ignore the CRC failed status.*

Table 186. Short response format

Bit position	Width	Value	Description
47	1	0	Start bit
46	1	0	Transmission bit
[45:40]	6	-	Command index
[39:8]	32	-	Argument
[7:1]	7	-	CRC7(or 1111111)
0	1	1	End bit

Table 187. Long response format

Bit position	Width	Value	Description
135	1	0	Start bit
134	1	0	Transmission bit
[133:128]	6	111111	Reserved
[127:1]	127	-	CID or CSD (including internal CRC7)
0	1	1	End bit

The command register contains the command index (six bits sent to a card) and the command type. These determine whether the command requires a response, and whether the response is 48 or 136 bits long (see [Section 30.8.4 on page 1108](#)). The command path implements the status flags shown in [Table 188](#):

Table 188. Command path status flags

Flag	Description
CMDREND	Set if response CRC is OK.
CCRCFAIL	Set if response CRC fails.
CMDSENT	Set when command (that does not require response) is sent
CTIMEOUT	Response timeout.
CMDACT	Command transfer in progress.

The CRC generator calculates the CRC checksum for all bits before the CRC code. This includes the start bit, transmitter bit, command index, and command argument (or card status). The CRC checksum is calculated for the first 120 bits of CID or CSD for the long response format. Note that the start bit, transmitter bit and the six reserved bits are not used in the CRC calculation.

The CRC checksum is a 7-bit value:

$$\text{CRC}[6:0] = \text{Remainder } [(M(x) * x^7) / G(x)]$$

$$G(x) = x^7 + x^3 + 1$$

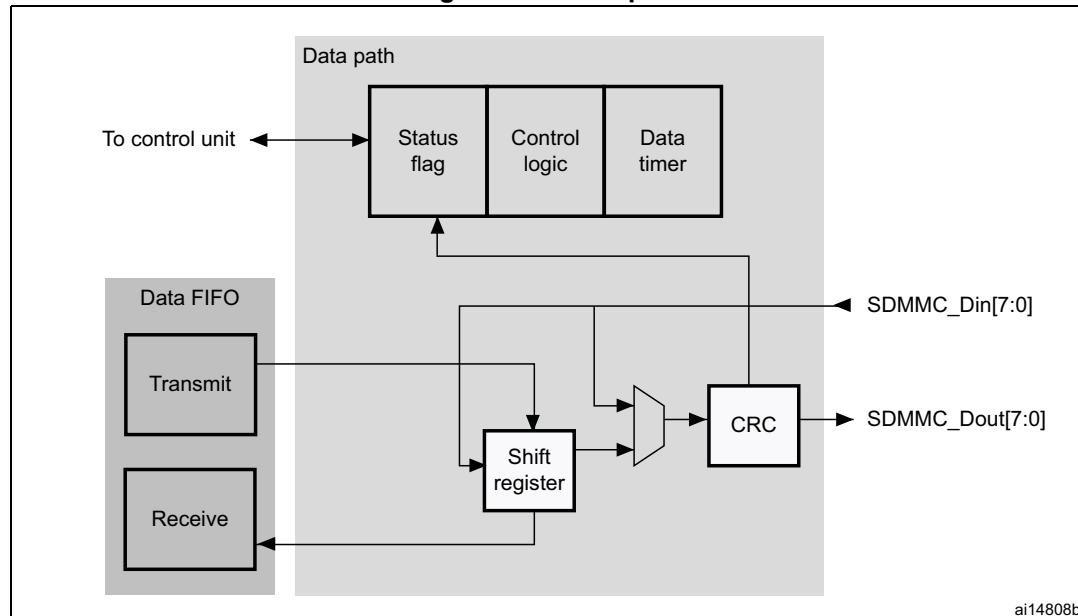
$$M(x) = (\text{start bit}) * x^{39} + \dots + (\text{last bit before CRC}) * x^0, \text{ or}$$

$$M(x) = (\text{start bit}) * x^{119} + \dots + (\text{last bit before CRC}) * x^0$$

Data path

The data path subunit transfers data to and from cards. [Figure 408](#) shows a block diagram of the data path.

Figure 408. Data path



The card databus width can be programmed using the clock control register. If the 4-bit wide bus mode is enabled, data is transferred at four bits per clock cycle over all four data signals (SDMMC_D[3:0]). If the 8-bit wide bus mode is enabled, data is transferred at eight bits per clock cycle over all eight data signals (SDMMC_D[7:0]). If the wide bus mode is not enabled, only one bit per clock cycle is transferred over SDMMC_D0.

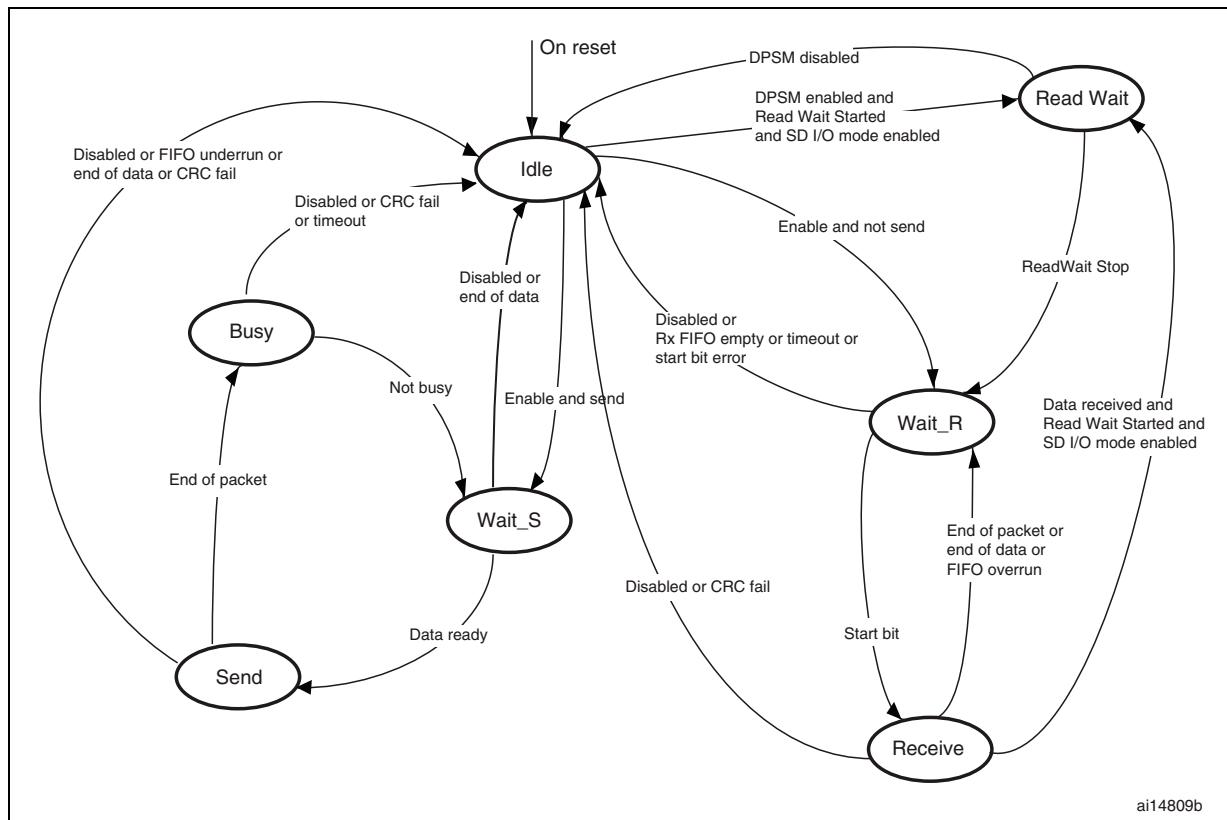
Depending on the transfer direction (send or receive), the data path state machine (DPSM) moves to the Wait_S or Wait_R state when it is enabled:

- Send: the DPSM moves to the Wait_S state. If there is data in the transmit FIFO, the DPSM moves to the Send state, and the data path subunit starts sending data to a card.
- Receive: the DPSM moves to the Wait_R state and waits for a start bit. When it receives a start bit, the DPSM moves to the Receive state, and the data path subunit starts receiving data from a card.

Data path state machine (DPSM)

The DPSM operates at SDMMC_CK frequency. Data on the card bus signals is synchronous to the rising edge of SDMMC_CK. The DPSM has six states, as shown in [Figure 409: Data path state machine \(DPSM\)](#).

Figure 409. Data path state machine (DPSM)



- **Idle:** the data path is inactive, and the SDMMC_D[7:0] outputs are in Hi-Z. When the data control register is written and the enable bit is set, the DPSM loads the data counter with a new value and, depending on the data direction bit, moves to either the Wait_S or the Wait_R state.
- **Wait_R:** if the data counter equals zero, the DPSM moves to the Idle state when the receive FIFO is empty. If the data counter is not zero, the DPSM waits for a start bit on SDMMC_D. The DPSM moves to the Receive state if it receives a start bit before a timeout, and loads the data block counter. If it reaches a timeout before it detects a start bit, it moves to the Idle state and sets the timeout status flag.
- **Receive:** serial data received from a card is packed in bytes and written to the data FIFO. Depending on the transfer mode bit in the data control register, the data transfer mode can be either block or stream:
 - In block mode, when the data block counter reaches zero, the DPSM waits until it receives the CRC code. If the received code matches the internally generated CRC code, the DPSM moves to the Wait_R state. If not, the CRC fail status flag is set and the DPSM moves to the Idle state.
 - In stream mode, the DPSM receives data while the data counter is not zero. When the counter is zero, the remaining data in the shift register is written to the data FIFO, and the DPSM moves to the Wait_R state.
- If a FIFO overrun error occurs, the DPSM sets the FIFO error flag and moves to the Idle state:
- **Wait_S:** the DPSM moves to the Idle state if the data counter is zero. If not, it waits until the data FIFO empty flag is deasserted, and moves to the Send state.

Note: The DPSM remains in the Wait_S state for at least two clock periods to meet the N_{WR} timing requirements, where N_{WR} is the number of clock cycles between the reception of the card response and the start of the data transfer from the host.

- Send: the DPSM starts sending data to a card. Depending on the transfer mode bit in the data control register, the data transfer mode can be either block or stream:
 - In block mode, when the data block counter reaches zero, the DPSM sends an internally generated CRC code and end bit, and moves to the Busy state.
 - In stream mode, the DPSM sends data to a card while the enable bit is high and the data counter is not zero. It then moves to the Idle state.
- If a FIFO underrun error occurs, the DPSM sets the FIFO error flag and moves to the Idle state.
- Busy: the DPSM waits for the CRC status flag:
 - If it does not receive a positive CRC status, it moves to the Idle state and sets the CRC fail status flag.
 - If it receives a positive CRC status, it moves to the Wait_S state if SDMMC_D0 is not low (the card is not busy).

If a timeout occurs while the DPSM is in the Busy state, it sets the data timeout flag and moves to the Idle state.

The data timer is enabled when the DPSM is in the Wait_R or Busy state, and generates the data timeout error:

- When transmitting data, the timeout occurs if the DPSM stays in the Busy state for longer than the programmed timeout period
- When receiving data, the timeout occurs if the end of the data is not true, and if the DPSM stays in the Wait_R state for longer than the programmed timeout period.
- **Data:** data can be transferred from the card to the host or vice versa. Data is transferred via the data lines. They are stored in a FIFO of 32 words, each word is 32 bits wide.

Table 189. Data token format

Description	Start bit	Data	CRC16	End bit
Block Data	0	-	yes	1
Stream Data	0	-	no	1

DPSM Flags

The status of the data path subunit transfer is reported by several status flags

Table 190. DPSM flags

Flag	Description
DBCKEND	Set to high when data block send/receive CRC check is passed. In SDIO multibyte transfer mode this flag is set at the end of the transfer (a multibyte transfer is considered as a single block transfer by the host).
DATAEND	Set to high when SDMMC_DCOUNT register decrements and reaches 0. DATAEND indicates the end of a transfer on SDMMC data line.
DTIMEOUT	Set to high when data timeout period is reached. When data timer reaches zero while DPSM is in Wait_R or Busy state, timeout is set. DTIMEOUT can be set after DATAEND if DPSM remains in busy state for longer than the programmed period.
DCRCFAIL	Set to high when data block send/receive CRC check fails.

Data FIFO

The data FIFO (first-in-first-out) subunit is a data buffer with a transmit and receive unit.

The FIFO contains a 32-bit wide, 32-word deep data buffer, and transmit and receive logic. Because the data FIFO operates in the APB2 clock domain (PCLK2), all signals from the subunits in the SDMMC clock domain (SDMMCCCLK) are resynchronized.

Depending on the TXACT and RXACT flags, the FIFO can be disabled, transmit enabled, or receive enabled. TXACT and RXACT are driven by the data path subunit and are mutually exclusive:

- The transmit FIFO refers to the transmit logic and data buffer when TXACT is asserted
- The receive FIFO refers to the receive logic and data buffer when RXACT is asserted
- Transmit FIFO:
Data can be written to the transmit FIFO through the APB2 interface when the SDMMC is enabled for transmission.
The transmit FIFO is accessible via 32 sequential addresses. The transmit FIFO contains a data output register that holds the data word pointed to by the read pointer. When the data path subunit has loaded its shift register, it increments the read pointer and drives new data out.
If the transmit FIFO is disabled, all status flags are deasserted. The data path subunit asserts TXACT when it transmits data.

Table 191. Transmit FIFO status flags

Flag	Description
TXFIFOF	Set to high when all 32 transmit FIFO words contain valid data.
TXFIFOE	Set to high when the transmit FIFO does not contain valid data.
TXFIFOHE	Set to high when 8 or more transmit FIFO words are empty. This flag can be used as a DMA request.
TXDAVL	Set to high when the transmit FIFO contains valid data. This flag is the inverse of the TXFIFOE flag.
TXUNDERR	Set to high when an underrun error occurs. This flag is cleared by writing to the SDMMC Clear register. <i>Note:</i> In case of TXUNDERR, and DMA is used to fill SDMMC FIFO, user software should disable DMA stream, and then write DMAEN bit in SDMMC_DCTRL with '0' (to disable DMA request generation).

- Receive FIFO

When the data path subunit receives a word of data, it drives the data on the write databus. The write pointer is incremented after the write operation completes. On the read side, the contents of the FIFO word pointed to by the current value of the read pointer is driven onto the read databus. If the receive FIFO is disabled, all status flags are deasserted, and the read and write pointers are reset. The data path subunit asserts RXACT when it receives data. [Table 192](#) lists the receive FIFO status flags. The receive FIFO is accessible via 32 sequential addresses.

Table 192. Receive FIFO status flags

Flag	Description
RXFIFOF	Set to high when all 32 receive FIFO words contain valid data
RXFIFOE	Set to high when the receive FIFO does not contain valid data.
RXFIFOHF	Set to high when 8 or more receive FIFO words contain valid data. This flag can be used as a DMA request.
RXDAVL	Set to high when the receive FIFO is not empty. This flag is the inverse of the RXFIFOE flag.
RXOVERR	Set to high when an overrun error occurs. This flag is cleared by writing to the SDMMC Clear register. <i>Note:</i> In case of RXOVERR, and DMA is used to read SDMMC FIFO, user software should disable DMA stream, and then write DMAEN bit in SDMMC_DCTRL with '0' (to disable DMA request generation).

30.3.2 SDMMC APB2 interface

The APB2 interface generates the interrupt and DMA requests, and accesses the SDMMC adapter registers and the data FIFO. It consists of a data path, register decoder, and interrupt/DMA logic.

SDMMC interrupts

The interrupt logic generates an interrupt request signal that is asserted when at least one of the selected status flags is high. A mask register is provided to allow selection of the conditions that will generate an interrupt. A status flag generates the interrupt request if a corresponding mask flag is set.

SDMMC/DMA interface

SDMMC APB interface controls all subunit to perform transfers between the host and card

Example of read procedure using DMA

Send CMD17 (READ_BLOCK) as follows:

- a) Program the SDMMC data length register (SDMMC data timer register should be already programmed before the card identification process)
- b) Program DMA channel (refer to [DMA configuration for SDMMC controller](#))
- c) Program the SDMMC data control register: DTEN with '1' (SDMMC card host enabled to send data); DTDIR with '1' (from card to controller); DTMODE with '0' (block data transfer); DMAEN with '1' (DMA enabled); DBLOCKSIZE with 0x9 (512 bytes). Other fields are don't care.
- d) Program the SDMMC argument register with the address location of the card from where data is to be transferred
- e) Program the SDMMC command register: CmdIndex with 17(READ_BLOCK); WaitResp with '1' (SDMMC card host waits for a response); CPSMEN with '1' (SDMMC card host enabled to send a command). Other fields are at their reset value.
- f) Wait for SDMMC_STA[6] = CMDREND interrupt, (CMDREND is set if there is no error on command path).
- g) Wait for SDMMC_STA[10] = DBCKEND, (DBCKEND is set in case of no errors until the CRC check is passed)
- h) Wait until the FIFO is empty, when FIFO is empty the SDMMC_STA[5] = RXOVERRR value has to be checked to guarantee that read succeeded

Note:

When FIFO overrun error occurs with last 1-4 bytes, it may happen that RXOVERRR flag is set 2 APB clock cycles after DATAEND flag is set. To guarantee success of read operation RXOVERRR must be checked after FIFO is empty.

Example of write procedure using DMA

Send CMD24 (WRITE_BLOCK) as follows:

- a) Program the SDMMC data length register (SDMMC data timer register should be already programmed before the card identification process)
- b) Program DMA channel (refer to [DMA configuration for SDMMC controller](#))
- c) Program the SDMMC argument register with the address location of the card from where data is to be transferred
- d) Program the SDMMC command register: CmdIndex with 24(WRITE_BLOCK); WaitResp with '1' (SDMMC card host waits for a response); CPSMEN with '1' (SDMMC card host enabled to send a command). Other fields are at their reset value.
- e) Wait for SDMMC_STA[6] = CMDREND interrupt, then Program the SDMMC data control register: DTEN with '1' (SDMMC card host enabled to send data); DTDIR with '0' (from controller to card); DTMODE with '0' (block data transfer); DMAEN with '1' (DMA enabled); DBLOCKSIZE with 0x9 (512 bytes). Other fields are don't care.
- f) Wait for SDMMC_STA[10] = DBCKEND, (DBCKEND is set in case of no errors)

DMA configuration for SDMMC controller

- a) Enable DMA2 controller and clear any pending interrupts.
- b) Program the DMA2_Stream3 (or DMA2_Stream6) Channel4 source address register with the memory location base address and DMA2_Stream3 (or DMA2_Stream6) Channel4 destination address register with the SDMMC_FIFO register address.
- c) Program DMA2_Stream3 (or DMA2_Stream6) Channel4 control register (memory increment, not peripheral increment, peripheral and source width is word size).
- d) Program DMA2_Stream3 (or DMA2_Stream6) Channel4 to select the peripheral as flow controller (set PFCTRL bit in DMA_S3CR (or DMA_S6CR) configuration register).
- e) Configure the incremental burst transfer to 4 beats (at least from peripheral side) in DMA2_Stream3 (or DMA2_Stream6) Channel4.
- f) Enable DMA2_Stream3 (or DMA2_Stream6) Channel4

Note: SDMMC host allows only to use the DMA in peripheral flow controller mode. DMA stream used to serve SDMMC must be configured in peripheral flow controller mode

SDMMC generates only DMA burst requests to DMA controller. DMA must be configured in incremental burst mode on peripheral side.

30.4 Card functional description

30.4.1 Card identification mode

While in card identification mode the host resets all cards, validates the operation voltage range, identifies cards and sets a relative card address (RCA) for each card on the bus. All data communications in the card identification mode use the command line (CMD) only.

30.4.2 Card reset

The GO_IDLE_STATE command (CMD0) is the software reset command and it puts the MultiMediaCard and SD memory in the Idle state. The IO_RW_DIRECT command (CMD52) resets the SD I/O card. After power-up or CMD0, all cards output bus drivers are in the high-impedance state and the cards are initialized with a default relative card address (RCA=0x0001) and with a default driver stage register setting (lowest speed, highest driving current capability).

30.4.3 Operating voltage range validation

All cards can communicate with the SDMMC card host using any operating voltage within the specification range. The supported minimum and maximum V_{DD} values are defined in the operation conditions register (OCR) on the card.

Cards that store the card identification number (CID) and card specific data (CSD) in the payload memory are able to communicate this information only under data-transfer V_{DD} conditions. When the SDMMC card host module and the card have incompatible V_{DD} ranges, the card is not able to complete the identification cycle and cannot send CSD data. For this purpose, the special commands, SEND_OP_COND (CMD1), SD_APP_OP_COND (ACMD41 for SD Memory), and IO_SEND_OP_COND (CMD5 for SD I/O), are designed to provide a mechanism to identify and reject cards that do not match the V_{DD} range desired by the SDMMC card host. The SDMMC card host sends the required V_{DD} voltage window as the operand of these commands. Cards that cannot perform data transfer in the specified range disconnect from the bus and go to the inactive state.

By using these commands without including the voltage range as the operand, the SDMMC card host can query each card and determine the common voltage range before placing out-of-range cards in the inactive state. This query is used when the SDMMC card host is able to select a common voltage range or when the user requires notification that cards are not usable.

30.4.4 Card identification process

The card identification process differs for MultiMediaCards and SD cards. For MultiMediaCard cards, the identification process starts at clock rate F_{od} . The SDMMC_CMD line output drivers are open-drain and allow parallel card operation during this process. The registration process is accomplished as follows:

1. The bus is activated.
2. The SDMMC card host broadcasts SEND_OP_COND (CMD1) to receive operation conditions.
3. The response is the wired AND operation of the operation condition registers from all cards.
4. Incompatible cards are placed in the inactive state.
5. The SDMMC card host broadcasts ALL_SEND_CID (CMD2) to all active cards.
6. The active cards simultaneously send their CID numbers serially. Cards with outgoing CID bits that do not match the bits on the command line stop transmitting and must wait for the next identification cycle. One card successfully transmits a full CID to the SDMMC card host and enters the Identification state.
7. The SDMMC card host issues SET_RELATIVE_ADDR (CMD3) to that card. This new address is called the relative card address (RCA); it is shorter than the CID and

addresses the card. The assigned card changes to the Standby state, it does not react to further identification cycles, and its output switches from open-drain to push-pull.

8. The SDMMC card host repeats steps 5 through 7 until it receives a timeout condition.

For the SD card, the identification process starts at clock rate F_{od} , and the SDMMC_CMD line output drivers are push-pull drivers instead of open-drain. The registration process is accomplished as follows:

1. The bus is activated.
2. The SDMMC card host broadcasts SD_APP_OP_COND (ACMD41).
3. The cards respond with the contents of their operation condition registers.
4. The incompatible cards are placed in the inactive state.
5. The SDMMC card host broadcasts ALL_SEND_CID (CMD2) to all active cards.
6. The cards send back their unique card identification numbers (CIDs) and enter the Identification state.
7. The SDMMC card host issues SET_RELATIVE_ADDR (CMD3) to an active card with an address. This new address is called the relative card address (RCA); it is shorter than the CID and addresses the card. The assigned card changes to the Standby state. The SDMMC card host can reissue this command to change the RCA. The RCA of the card is the last assigned value.
8. The SDMMC card host repeats steps 5 through 7 with all active cards.

For the SD I/O card, the registration process is accomplished as follows:

1. The bus is activated.
2. The SDMMC card host sends IO_SEND_OP_COND (CMD5).
3. The cards respond with the contents of their operation condition registers.
4. The incompatible cards are set to the inactive state.
5. The SDMMC card host issues SET_RELATIVE_ADDR (CMD3) to an active card with an address. This new address is called the relative card address (RCA); it is shorter than the CID and addresses the card. The assigned card changes to the Standby state. The SDMMC card host can reissue this command to change the RCA. The RCA of the card is the last assigned value.

30.4.5 Block write

During block write (CMD24 - 27) one or more blocks of data are transferred from the host to the card with a CRC appended to the end of each block by the host. A card supporting block write is always able to accept a block of data defined by WRITE_BL_LEN. If the CRC fails, the card indicates the failure on the SDMMC_D line and the transferred data are discarded and not written, and all further transmitted blocks (in multiple block write mode) are ignored.

If the host uses partial blocks whose accumulated length is not block aligned and, block misalignment is not allowed (CSD parameter WRITE_BLK_MISALIGN is not set), the card will detect the block misalignment error before the beginning of the first misaligned block. (ADDRESS_ERROR error bit is set in the status register). The write operation will also be aborted if the host tries to write over a write-protected area. In this case, however, the card will set the WP_VIOLATION bit.

Programming of the CID and CSD registers does not require a previous block length setting. The transferred data is also CRC protected. If a part of the CSD or CID register is stored in ROM, then this unchangeable part must match the corresponding part of the receive buffer. If this match fails, then the card reports an error and does not change any register contents.

Some cards may require long and unpredictable times to write a block of data. After receiving a block of data and completing the CRC check, the card begins writing and holds the SDMMC_D line low if its write buffer is full and unable to accept new data from a new WRITE_BLOCK command. The host may poll the status of the card with a SEND_STATUS command (CMD13) at any time, and the card will respond with its status. The READY_FOR_DATA status bit indicates whether the card can accept new data or whether the write process is still in progress. The host may deselect the card by issuing CMD7 (to select a different card), which will place the card in the Disconnect state and release the SDMMC_D line(s) without interrupting the write operation. When reselecting the card, it will reactivate busy indication by pulling SDMMC_D to low if programming is still in progress and the write buffer is unavailable.

30.4.6 Block read

In Block read mode the basic unit of data transfer is a block whose maximum size is defined in the CSD (READ_BL_LEN). If READ_BL_PARTIAL is set, smaller blocks whose start and end addresses are entirely contained within one physical block (as defined by READ_BL_LEN) may also be transmitted. A CRC is appended to the end of each block, ensuring data transfer integrity. CMD17 (READ_SINGLE_BLOCK) initiates a block read and after completing the transfer, the card returns to the Transfer state.

CMD18 (READ_MULTIPLE_BLOCK) starts a transfer of several consecutive blocks.

The host can abort reading at any time, within a multiple block operation, regardless of its type. Transaction abort is done by sending the stop transmission command.

If the card detects an error (for example, out of range, address misalignment or internal error) during a multiple block read operation (both types) it stops the data transmission and remains in the data state. The host must then abort the operation by sending the stop transmission command. The read error is reported in the response to the stop transmission command.

If the host sends a stop transmission command after the card transmits the last block of a multiple block operation with a predefined number of blocks, it is responded to as an illegal command, since the card is no longer in the data state. If the host uses partial blocks whose accumulated length is not block-aligned and block misalignment is not allowed, the card detects a block misalignment error condition at the beginning of the first misaligned block (ADDRESS_ERROR error bit is set in the status register).

30.4.7 Stream access, stream write and stream read (MultiMediaCard only)

In stream mode, data is transferred in bytes and no CRC is appended at the end of each block.

Stream write (MultiMediaCard only)

WRITE_DAT_UNTIL_STOP (CMD20) starts the data transfer from the SDMMC card host to the card, beginning at the specified address and continuing until the SDMMC card host issues a stop command. When partial blocks are allowed (CSD parameter WRITE_BL_PARTIAL is set), the data stream can start and stop at any address within the card address space, otherwise it can only start and stop at block boundaries. Because the amount of data to be transferred is not determined in advance, a CRC cannot be used. When the end of the memory range is reached while sending data and no stop command is sent by the SDMMC card host, any additional transferred data are discarded.

The maximum clock frequency for a stream write operation is given by the following equation fields of the card-specific data register:

$$\text{Maximumspeed} = \text{MIN}(\text{TRANSPEED}, \frac{(8 \times 2^{\text{writeblen}})(-\text{NSAC})}{\text{TAAC} \times \text{R2WFACTOR}})$$

- Maximumspeed = maximum write frequency
- TRANSPEED = maximum data transfer rate
- writeblen = maximum write data block length
- NSAC = data read access time 2 in CLK cycles
- TAAC = data read access time 1
- R2WFACTOR = write speed factor

If the host attempts to use a higher frequency, the card may not be able to process the data and stop programming, set the OVERRUN error bit in the status register, and while ignoring all further data transfer, wait (in the receive data state) for a stop command. The write operation is also aborted if the host tries to write over a write-protected area. In this case, however, the card sets the WP_VIOLATION bit.

Stream read (MultiMediaCard only)

READ_DAT_UNTIL_STOP (CMD11) controls a stream-oriented data transfer.

This command instructs the card to send its data, starting at a specified address, until the SDMMC card host sends STOP_TRANSMISSION (CMD12). The stop command has an execution delay due to the serial command transmission and the data transfer stops after the end bit of the stop command. When the end of the memory range is reached while sending data and no stop command is sent by the SDMMC card host, any subsequent data sent are considered undefined.

The maximum clock frequency for a stream read operation is given by the following equation and uses fields of the card specific data register.

$$\text{Maximumspeed} = \text{MIN}(\text{TRANSPEED}, \frac{(8 \times 2^{\text{readblen}})(-\text{NSAC})}{\text{TAAC} \times \text{R2WFACTOR}})$$

- Maximumspeed = maximum read frequency
- TRANSPEED = maximum data transfer rate
- readblen = maximum read data block length
- writeblen = maximum write data block length
- NSAC = data read access time 2 in CLK cycles
- TAAC = data read access time 1
- R2WFACTOR = write speed factor

If the host attempts to use a higher frequency, the card is not able to sustain data transfer. If this happens, the card sets the UNDERRUN error bit in the status register, aborts the transmission and waits in the data state for a stop command.

30.4.8 Erase: group erase and sector erase

The erasable unit of the MultiMediaCard is the erase group. The erase group is measured in write blocks, which are the basic writable units of the card. The size of the erase group is a card-specific parameter and defined in the CSD.

The host can erase a contiguous range of Erase Groups. Starting the erase process is a three-step sequence.

First the host defines the start address of the range using the ERASE_GROUP_START (CMD35) command, next it defines the last address of the range using the ERASE_GROUP_END (CMD36) command and, finally, it starts the erase process by issuing the ERASE (CMD38) command. The address field in the erase commands is an Erase Group address in byte units. The card ignores all LSBs below the Erase Group size, effectively rounding the address down to the Erase Group boundary.

If an erase command is received out of sequence, the card sets the ERASE_SEQ_ERROR bit in the status register and resets the whole sequence.

If an out-of-sequence (neither of the erase commands, except SEND_STATUS) command received, the card sets the ERASE_RESET status bit in the status register, resets the erase sequence and executes the last command.

If the erase range includes write protected blocks, they are left intact and only nonprotected blocks are erased. The WP_ERASE_SKIP status bit in the status register is set.

The card indicates that an erase is in progress by holding SDMMC_D low. The actual erase time may be quite long, and the host may issue CMD7 to deselect the card.

30.4.9 Wide bus selection or deselection

Wide bus (4-bit bus width) operation mode is selected or deselected using SET_BUS_WIDTH (ACMD6). The default bus width after power-up or GO_IDLE_STATE (CMD0) is 1 bit. SET_BUS_WIDTH (ACMD6) is only valid in a transfer state, which means that the bus width can be changed only after a card is selected by SELECT/DESELECT_CARD (CMD7).

30.4.10 Protection management

Three write protection methods for the cards are supported in the SDMMC card host module:

1. internal card write protection (card responsibility)
2. mechanical write protection switch (SDMMC card host module responsibility only)
3. password-protected card lock operation

Internal card write protection

Card data can be protected against write and erase. By setting the permanent or temporary write-protect bits in the CSD, the entire card can be permanently write-protected by the manufacturer or content provider. For cards that support write protection of groups of sectors by setting the WP_GRP_ENABLE bit in the CSD, portions of the data can be protected, and the write protection can be changed by the application. The write protection is in units of WP_GRP_SIZE sectors as specified in the CSD. The SET_WRITE_PROT and CLR_WRITE_PROT commands control the protection of the addressed group. The SEND_WRITE_PROT command is similar to a single block read command. The card sends

a data block containing 32 write protection bits (representing 32 write protect groups starting at the specified address) followed by 16 CRC bits. The address field in the write protect commands is a group address in byte units.

The card ignores all LSBs below the group size.

Mechanical write protect switch

A mechanical sliding tab on the side of the card allows the user to set or clear the write protection on a card. When the sliding tab is positioned with the window open, the card is write-protected, and when the window is closed, the card contents can be changed. A matched switch on the socket side indicates to the SDMMC card host module that the card is write-protected. The SDMMC card host module is responsible for protecting the card. The position of the write protect switch is unknown to the internal circuitry of the card.

Password protect

The password protection feature enables the SDMMC card host module to lock and unlock a card with a password. The password is stored in the 128-bit PWD register and its size is set in the 8-bit PWD_LEN register. These registers are nonvolatile so that a power cycle does not erase them. Locked cards respond to and execute certain commands. This means that the SDMMC card host module is allowed to reset, initialize, select, and query for status, however it is not allowed to access data on the card. When the password is set (as indicated by a nonzero value of PWD_LEN), the card is locked automatically after power-up. As with the CSD and CID register write commands, the lock/unlock commands are available in the transfer state only. In this state, the command does not include an address argument and the card must be selected before using it. The card lock/unlock commands have the structure and bus transaction types of a regular single-block write command. The transferred data block includes all of the required information for the command (the password setting mode, the PWD itself, and card lock/unlock). The command data block size is defined by the SDMMC card host module before it sends the card lock/unlock command, and has the structure shown in [Table 206](#).

The bit settings are as follows:

- ERASE: setting it forces an erase operation. All other bits must be zero, and only the command byte is sent
- LOCK_UNLOCK: setting it locks the card. LOCK_UNLOCK can be set simultaneously with SET_PWD, however not with CLR_PWD
- CLR_PWD: setting it clears the password data
- SET_PWD: setting it saves the password data to memory
- PWD_LEN: it defines the length of the password in bytes
- PWD: the password (new or currently used, depending on the command)

The following sections list the command sequences to set/reset a password, lock/unlock the card, and force an erase.

Setting the password

1. Select a card (SELECT/DESELECT_CARD, CMD7), if none is already selected.
2. Define the block length (SET_BLOCKLEN, CMD16) to send, given by the 8-bit card lock/unlock mode, the 8-bit PWD_LEN, and the number of bytes of the new password.

- When a password replacement is done, the block size must take into account that both the old and the new passwords are sent with the command.
3. Send LOCK/UNLOCK (CMD42) with the appropriate data block size on the data line including the 16-bit CRC. The data block indicates the mode (SET_PWD = 1), the length (PWD_LEN), and the password (PWD) itself. When a password replacement is done, the length value (PWD_LEN) includes the length of both passwords, the old and the new one, and the PWD field includes the old password (currently used) followed by the new password.
 4. When the password is matched, the new password and its size are saved into the PWD and PWD_LEN fields, respectively. When the old password sent does not correspond (in size and/or content) to the expected password, the LOCK_UNLOCK_FAILED error bit is set in the card status register, and the password is not changed.

The password length field (PWD_LEN) indicates whether a password is currently set. When this field is nonzero, there is a password set and the card locks itself after power-up. It is possible to lock the card immediately in the current power session by setting the LOCK_UNLOCK bit (while setting the password) or sending an additional command for card locking.

Resetting the password

1. Select a card (SELECT/DESELECT_CARD, CMD7), if none is already selected.
2. Define the block length (SET_BLOCKLEN, CMD16) to send, given by the 8-bit card lock/unlock mode, the 8-bit PWD_LEN, and the number of bytes in the currently used password.
3. Send LOCK/UNLOCK (CMD42) with the appropriate data block size on the data line including the 16-bit CRC. The data block indicates the mode (CLR_PWD = 1), the length (PWD_LEN) and the password (PWD) itself. The LOCK_UNLOCK bit is ignored.
4. When the password is matched, the PWD field is cleared and PWD_LEN is set to 0. When the password sent does not correspond (in size and/or content) to the expected password, the LOCK_UNLOCK_FAILED error bit is set in the card status register, and the password is not changed.

Locking a card

1. Select a card (SELECT/DESELECT_CARD, CMD7), if none is already selected.
2. Define the block length (SET_BLOCKLEN, CMD16) to send, given by the 8-bit card lock/unlock mode (byte 0 in [Table 206](#)), the 8-bit PWD_LEN, and the number of bytes of the current password.
3. Send LOCK/UNLOCK (CMD42) with the appropriate data block size on the data line including the 16-bit CRC. The data block indicates the mode (LOCK_UNLOCK = 1), the length (PWD_LEN), and the password (PWD) itself.
4. When the password is matched, the card is locked and the CARD_IS_LOCKED status bit is set in the card status register. When the password sent does not correspond (in size and/or content) to the expected password, the LOCK_UNLOCK_FAILED error bit is set in the card status register, and the lock fails.

It is possible to set the password and to lock the card in the same sequence. In this case, the SDMMC card host module performs all the required steps for setting the password (see [Setting the password on page 1086](#)), however it is necessary to set the LOCK_UNLOCK bit in Step 3 when the new password command is sent.

When the password is previously set (PWD_LEN is not 0), the card is locked automatically after power on reset. An attempt to lock a locked card or to lock a card that does not have a password fails and the LOCK_UNLOCK_FAILED error bit is set in the card status register.

Unlocking the card

1. Select a card (SELECT/DESELECT_CARD, CMD7), if none is already selected.
2. Define the block length (SET_BLOCKLEN, CMD16) to send, given by the 8-bit cardlock/unlock mode (byte 0 in [Table 206](#)), the 8-bit PWD_LEN, and the number of bytes of the current password.
3. Send LOCK/UNLOCK (CMD42) with the appropriate data block size on the data line including the 16-bit CRC. The data block indicates the mode (LOCK_UNLOCK = 0), the length (PWD_LEN), and the password (PWD) itself.
4. When the password is matched, the card is unlocked and the CARD_IS_LOCKED status bit is cleared in the card status register. When the password sent is not correct in size and/or content and does not correspond to the expected password, the LOCK_UNLOCK_FAILED error bit is set in the card status register, and the card remains locked.

The unlocking function is only valid for the current power session. When the PWD field is not clear, the card is locked automatically on the next power-up.

An attempt to unlock an unlocked card fails and the LOCK_UNLOCK_FAILED error bit is set in the card status register.

Forcing erase

If the user has forgotten the password (PWD content), it is possible to access the card after clearing all the data on the card. This forced erase operation erases all card data and all password data.

1. Select a card (SELECT/DESELECT_CARD, CMD7), if none is already selected.
2. Set the block length (SET_BLOCKLEN, CMD16) to 1 byte. Only the 8-bit card lock/unlock byte (byte 0 in [Table 206](#)) is sent.
3. Send LOCK/UNLOCK (CMD42) with the appropriate data byte on the data line including the 16-bit CRC. The data block indicates the mode (ERASE = 1). All other bits must be zero.
4. When the ERASE bit is the only bit set in the data field, all card contents are erased, including the PWD and PWD_LEN fields, and the card is no longer locked. When any other bits are set, the LOCK_UNLOCK_FAILED error bit is set in the card status register and the card retains all of its data, and remains locked.

An attempt to use a force erase on an unlocked card fails and the LOCK_UNLOCK_FAILED error bit is set in the card status register.

30.4.11 Card status register

The response format R1 contains a 32-bit field named card status. This field is intended to transmit the card status information (which may be stored in a local status register) to the host. If not specified otherwise, the status entries are always related to the previously issued command.

Table 193 defines the different entries of the status. The type and clear condition fields in the table are abbreviated as follows:

Type:

- E: error bit
- S: status bit
- R: detected and set for the actual command response
- X: detected and set during command execution. The SDMMC card host must poll the card by issuing the status command to read these bits.

Clear condition:

- A: according to the card current state
- B: always related to the previous command. Reception of a valid command clears it (with a delay of one command)
- C: clear by read

Table 193. Card status

Bits	Identifier	Type	Value	Description	Clear condition
31	ADDRESS_OUT_OF_RANGE	E R X	'0'= no error '1'= error	The command address argument was out of the allowed range for this card. A multiple block or stream read/write operation is (although started in a valid address) attempting to read or write beyond the card capacity.	C
30	ADDRESS_MISALIGN	-	'0'= no error '1'= error	The commands address argument (in accordance with the currently set block length) positions the first data block misaligned to the card physical blocks. A multiple block read/write operation (although started with a valid address/block-length combination) is attempting to read or write a data block which is not aligned with the physical blocks of the card.	C
29	BLOCK_LEN_ERROR	-	'0'= no error '1'= error	Either the argument of a SET_BLOCKLEN command exceeds the maximum value allowed for the card, or the previously defined block length is illegal for the current command (e.g. the host issues a write command, the current block length is smaller than the maximum allowed value for the card and it is not allowed to write partial blocks)	C

Table 193. Card status (continued)

Bits	Identifier	Type	Value	Description	Clear condition
28	ERASE_SEQ_ERROR	-	'0'= no error '1'= error	An error in the sequence of erase commands occurred.	C
27	ERASE_PARAM	E X	'0'= no error '1'= error	An invalid selection of erase groups for erase occurred.	C
26	WP_VIOLATION	E X	'0'= no error '1'= error	Attempt to program a write-protected block.	C
25	CARD_IS_LOCKED	S R	'0' = card unlocked '1' = card locked	When set, signals that the card is locked by the host	A
24	LOCK_UNLOCK_FAILED	E X	'0'= no error '1'= error	Set when a sequence or password error has been detected in lock/unlock card command	C
23	COM_CRC_ERROR	E R	'0'= no error '1'= error	The CRC check of the previous command failed.	B
22	ILLEGAL_COMMAND	E R	'0'= no error '1'= error	Command not legal for the card state	B
21	CARD_ECC FAILED	E X	'0'= success '1'= failure	Card internal ECC was applied but failed to correct the data.	C
20	CC_ERROR	E R	'0'= no error '1'= error	(Undefined by the standard) A card error occurred, which is not related to the host command.	C
19	ERROR	E X	'0'= no error '1'= error	(Undefined by the standard) A generic card error related to the (and detected during) execution of the last host command (e.g. read or write failures).	C
18	Reserved				
17	Reserved				
16	CID/CSD_OVERWRITE	E X	'0'= no error '1'= error	Can be either of the following errors: – The CID register has already been written and cannot be overwritten – The read-only section of the CSD does not match the card contents – An attempt to reverse the copy (set as original) or permanent WP (unprotected) bits was made	C
15	WP_ERASE_SKIP	E X	'0'= not protected '1'= protected	Set when only partial address space was erased due to existing write	C
14	CARD_ECC_DISABLED	S X	'0'= enabled '1'= disabled	The command has been executed without using the internal ECC.	A

Table 193. Card status (continued)

Bits	Identifier	Type	Value	Description	Clear condition
13	ERASE_RESET	-	'0'= cleared '1'= set	An erase sequence was cleared before executing because an out of erase sequence command was received (commands other than CMD35, CMD36, CMD38 or CMD13)	C
12:9	CURRENT_STATE	S R	0 = Idle 1 = Ready 2 = Ident 3 = Stby 4 = Tran 5 = Data 6 = Rcv 7 = Prg 8 = Dis 9 = Btst 10-15 = reserved	The state of the card when receiving the command. If the command execution causes a state change, it will be visible to the host in the response on the next command. The four bits are interpreted as a binary number between 0 and 15.	B
8	READY_FOR_DATA	S R	'0'= not ready '1' = ready	Corresponds to buffer empty signalling on the bus	-
7	SWITCH_ERROR	E X	'0'= no error '1'= switch error	If set, the card did not switch to the expected mode as requested by the SWITCH command	B
6	Reserved				
5	APP_CMD	S R	'0' = Disabled '1' = Enabled	The card will expect ACMD, or an indication that the command has been interpreted as ACMD	C
4	Reserved for SD I/O Card				
3	AKE_SEQ_ERROR	E R	'0'= no error '1'= error	Error in the sequence of the authentication process	C
2	Reserved for application specific commands				
1	Reserved for manufacturer test mode				
0					

30.4.12 SD status register

The SD status contains status bits that are related to the SD memory card proprietary features and may be used for future application-specific usage. The size of the SD Status is one data block of 512 bits. The contents of this register are transmitted to the SDMMC card host if ACMD13 is sent (CMD55 followed with CMD13). ACMD13 can be sent to a card in transfer state only (card is selected).

Table 194 defines the different entries of the SD status register. The type and clear condition fields in the table are abbreviated as follows:

Type:

- E: error bit
- S: status bit
- R: detected and set for the actual command response
- X: detected and set during command execution. The SDMMC card Host must poll the card by issuing the status command to read these bits

Clear condition:

- A: according to the card current state
- B: always related to the previous command. Reception of a valid command clears it (with a delay of one command)
- C: clear by read

Table 194. SD status

Bits	Identifier	Type	Value	Description	Clear condition
511: 510	DAT_BUS_WIDTH	S R	'00'= 1 (default) '01'= reserved '10'= 4 bit width '11'= reserved	Shows the currently defined databus width that was defined by SET_BUS_WIDTH command	A
509	SECURED_MODE	S R	'0'= Not in the mode '1'= In Secured Mode	Card is in Secured Mode of operation (refer to the "SD Security Specification").	A
508: 496	Reserved				
495: 480	SD_CARD_TYPE	S R	'00xxh'= SD Memory Cards as defined in Physical Spec Ver1.01-2.00 ('x'= don't care). The following cards are currently defined: '0000'= Regular SD RD/WR Card. '0001'= SD ROM Card	In the future, the 8 LSBs will be used to define different variations of an SD memory card (each bit will define different SD types). The 8 MSBs will be used to define SD Cards that do not comply with current SD physical layer specification.	A
479: 448	SIZE_OF_PROTECTED_AREA	S R	Size of protected area (See below)	(See below)	A
447: 440	SPEED_CLASS	S R	Speed Class of the card (See below)	(See below)	A

Table 194. SD status (continued)

Bits	Identifier	Type	Value	Description	Clear condition
439:432	PERFORMANCE_MOVE	S R	Performance of move indicated by 1 [MB/s] step. (See below)	(See below)	A
431:428	AU_SIZE	S R	Size of AU (See below)	(See below)	A
427:424	Reserved				
423:408	ERASE_SIZE	S R	Number of AUs to be erased at a time	(See below)	A
407:402	ERASE_TIMEOUT	S R	Timeout value for erasing areas specified by UNIT_OF_ERASE_AU	(See below)	A
401:400	ERASE_OFFSET	S R	Fixed offset value added to erase time.	(See below)	A
399:312	Reserved				
311:0	Reserved for Manufacturer				

SIZE_OF_PROTECTED_AREA

Setting this field differs between standard- and high-capacity cards. In the case of a standard-capacity card, the capacity of protected area is calculated as follows:

$$\text{Protected area} = \text{SIZE_OF_PROTECTED_AREA_} * \text{MULT} * \text{BLOCK_LEN}.$$

SIZE_OF_PROTECTED_AREA is specified by the unit in MULT*BLOCK_LEN.

In the case of a high-capacity card, the capacity of protected area is specified in this field:

$$\text{Protected area} = \text{SIZE_OF_PROTECTED_AREA}$$

SIZE_OF_PROTECTED_AREA is specified by the unit in bytes.

SPEED_CLASS

This 8-bit field indicates the speed class and the value can be calculated by $P_W/2$ (where P_W is the write performance).

Table 195. Speed class code field

SPEED_CLASS	Value definition
00h	Class 0
01h	Class 2
02h	Class 4
03h	Class 6
04h – FFh	Reserved

PERFORMANCE_MOVE

This 8-bit field indicates Pm (performance move) and the value can be set by 1 [MB/sec] steps. If the card does not move used RUs (recording units), Pm should be considered as infinity. Setting the field to FFh means infinity.

Table 196. Performance move field

PERFORMANCE_MOVE	Value definition
00h	Not defined
01h	1 [MB/sec]
02h	02h 2 [MB/sec]
-----	-----
FEh	254 [MB/sec]
FFh	Infinity

AU_SIZE

This 4-bit field indicates the AU size and the value can be selected in the power of 2 base from 16 KB.

Table 197. AU_SIZE field

AU_SIZE	Value definition
00h	Not defined
01h	16 KB
02h	32 KB
03h	64 KB
04h	128 KB
05h	256 KB
06h	512 KB
07h	1 MB
08h	2 MB
09h	4 MB
Ah – Fh	Reserved

The maximum AU size, which depends on the card capacity, is defined in [Table 198](#). The card can be set to any AU size between RU size and maximum AU size.

Table 198. Maximum AU size

Capacity	16 MB-64 MB	128 MB-256 MB	512 MB	1 GB-32 GB
Maximum AU Size	512 KB	1 MB	2 MB	4 MB

ERASE_SIZE

This 16-bit field indicates N_{ERASE} . When N_{ERASE} numbers of AUs are erased, the timeout value is specified by ERASE_TIMEOUT (Refer to [ERASE_TIMEOUT](#)). The host should determine the proper number of AUs to be erased in one operation so that the host can show the progress of the erase operation. If this field is set to 0, the erase timeout calculation is not supported.

Table 199. Erase size field

ERASE_SIZE	Value definition
0000h	Erase timeout calculation is not supported.
0001h	1 AU
0002h	2 AU
0003h	3 AU
-----	-----
FFFFh	65535 AU

ERASE_TIMEOUT

This 6-bit field indicates T_{ERASE} and the value indicates the erase timeout from offset when multiple AUs are being erased as specified by ERASE_SIZE. The range of ERASE_TIMEOUT can be defined as up to 63 seconds and the card manufacturer can choose any combination of ERASE_SIZE and ERASE_TIMEOUT depending on the implementation. Determining ERASE_TIMEOUT determines the ERASE_SIZE.

Table 200. Erase timeout field

ERASE_TIMEOUT	Value definition
00	Erase timeout calculation is not supported.
01	1 [sec]
02	2 [sec]
03	3 [sec]
-----	-----
63	63 [sec]

ERASE_OFFSET

This 2-bit field indicates T_{OFFSET} and one of four values can be selected. This field is meaningless if the ERASE_SIZE and ERASE_TIMEOUT fields are set to 0.

Table 201. Erase offset field

ERASE_OFFSET	Value definition
0h	0 [sec]
1h	1 [sec]

Table 201. Erase offset field (continued)

ERASE_OFFSET	Value definition
2h	2 [sec]
3h	3 [sec]

30.4.13 SD I/O mode

SD I/O interrupts

To allow the SD I/O card to interrupt the MultiMediaCard/SD module, an interrupt function is available on a pin on the SD interface. Pin 8, used as SDMMC_D1 when operating in the 4-bit SD mode, signals the cards interrupt to the MultiMediaCard/SD module. The use of the interrupt is optional for each card or function within a card. The SD I/O interrupt is level-sensitive, which means that the interrupt line must be held active (low) until it is either recognized and acted upon by the MultiMediaCard/SD module or deasserted due to the end of the interrupt period. After the MultiMediaCard/SD module has serviced the interrupt, the interrupt status bit is cleared via an I/O write to the appropriate bit in the SD I/O card's internal registers. The interrupt output of all SD I/O cards is active low and the application must provide pull-up resistors externally on all data lines (SDMMC_D[3:0]). The MultiMediaCard/SD module samples the level of pin 8 (SDMMC_D/IRQ) into the interrupt detector only during the interrupt period. At all other times, the MultiMediaCard/SD module ignores this value.

The interrupt period is applicable for both memory and I/O operations. The definition of the interrupt period for operations with single blocks is different from the definition for multiple-block data transfers.

SD I/O suspend and resume

Within a multifunction SD I/O or a card with both I/O and memory functions, there are multiple devices (I/O and memory) that share access to the MMC/SD bus. To share access to the MMC/SD module among multiple devices, SD I/O and combo cards optionally implement the concept of suspend/resume. When a card supports suspend/resume, the MMC/SD module can temporarily halt a data transfer operation to one function or memory (suspend) to free the bus for a higher-priority transfer to a different function or memory. After this higher-priority transfer is complete, the original transfer is resumed (restarted) where it left off. Support of suspend/resume is optional on a per-card basis. To perform the suspend/resume operation on the MMC/SD bus, the MMC/SD module performs the following steps:

1. Determines the function currently using the SDMMC_D [3:0] line(s)
2. Requests the lower-priority or slower transaction to suspend
3. Waits for the transaction suspension to complete
4. Begins the higher-priority transaction
5. Waits for the completion of the higher priority transaction
6. Restores the suspended transaction

SD I/O ReadWait

The optional ReadWait (RW) operation is defined only for the SD 1-bit and 4-bit modes. The ReadWait operation allows the MMC/SD module to signal a card that it is reading multiple

registers (IO_RW_EXTENDED, CMD53) to temporarily stall the data transfer while allowing the MMC/SD module to send commands to any function within the SD I/O device. To determine when a card supports the ReadWait protocol, the MMC/SD module must test capability bits in the internal card registers. The timing for ReadWait is based on the interrupt period.

30.4.14 Commands and responses

Application-specific and general commands

The SDMMC card host module system is designed to provide a standard interface for a variety of applications types. In this environment, there is a need for specific customer/application features. To implement these features, two types of generic commands are defined in the standard: application-specific commands (ACMD) and general commands (GEN_CMD).

When the card receives the APP_CMD (CMD55) command, the card expects the next command to be an application-specific command. ACMDs have the same structure as regular MultiMediaCard commands and can have the same CMD number. The card recognizes it as ACMD because it appears after APP_CMD (CMD55). When the command immediately following the APP_CMD (CMD55) is not a defined application-specific command, the standard command is used. For example, when the card has a definition for SD_STATUS (ACMD13), and receives CMD13 immediately following APP_CMD (CMD55), this is interpreted as SD_STATUS (ACMD13). However, when the card receives CMD7 immediately following APP_CMD (CMD55) and the card does not have a definition for ACMD7, this is interpreted as the standard (SELECT/DESELECT_CARD) CMD7.

To use one of the manufacturer-specific ACMDs the SD card Host must perform the following steps:

1. Send APP_CMD (CMD55)
The card responds to the MultiMediaCard/SD module, indicating that the APP_CMD bit is set and an ACMD is now expected.
2. Send the required ACMD
The card responds to the MultiMediaCard/SD module, indicating that the APP_CMD bit is set and that the accepted command is interpreted as an ACMD. When a nonACMD is sent, it is handled by the card as a normal MultiMediaCard command and the APP_CMD bit in the card status register stays clear.

When an invalid command is sent (neither ACMD nor CMD) it is handled as a standard MultiMediaCard illegal command error.

The bus transaction for a GEN_CMD is the same as the single-block read or write commands (WRITE_BLOCK, CMD24 or READ_SINGLE_BLOCK,CMD17). In this case, the argument denotes the direction of the data transfer rather than the address, and the data block has vendor-specific format and meaning.

The card must be selected (in transfer state) before sending GEN_CMD (CMD56). The data block size is defined by SET_BLOCKLEN (CMD16). The response to GEN_CMD (CMD56) is in R1b format.

Command types

Both application-specific and general commands are divided into the four following types:

- **broadcast command (BC)**: sent to all cards; no responses returned.
- **broadcast command with response (BCR)**: sent to all cards; responses received from all cards simultaneously.
- **addressed (point-to-point) command (AC)**: sent to the card that is selected; does not include a data transfer on the SDMMC_D line(s).
- **addressed (point-to-point) data transfer command (ADTC)**: sent to the card that is selected; includes a data transfer on the SDMMC_D line(s).

Command formats

See [Table 185 on page 1072](#) for command formats.

Commands for the MultiMediaCard/SD module

Table 202. Block-oriented write commands

CMD index	Type	Argument	Response format	Abbreviation	Description
CMD23	ac	[31:16] set to 0 [15:0] number of blocks	R1	SET_BLOCK_COUNT	Defines the number of blocks which are going to be transferred in the multiple-block read or write command that follows.
CMD24	adtc	[31:0] data address	R1	WRITE_BLOCK	Writes a block of the size selected by the SET_BLOCKLEN command.
CMD25	adtc	[31:0] data address	R1	WRITE_MULTIPLE_BLOCK	Continuously writes blocks of data until a STOP_TRANSMISSION follows or the requested number of blocks has been received.
CMD26	adtc	[31:0] stuff bits	R1	PROGRAM_CID	Programming of the card identification register. This command must be issued only once per card. The card contains hardware to prevent this operation after the first programming. Normally this command is reserved for manufacturer.
CMD27	adtc	[31:0] stuff bits	R1	PROGRAM_CSD	Programming of the programmable bits of the CSD.

Table 203. Block-oriented write protection commands

CMD index	Type	Argument	Response format	Abbreviation	Description
CMD28	ac	[31:0] data address	R1b	SET_WRITE_PROT	If the card has write protection features, this command sets the write protection bit of the addressed group. The properties of write protection are coded in the card-specific data (WP_GRP_SIZE).
CMD29	ac	[31:0] data address	R1b	CLR_WRITE_PROT	If the card provides write protection features, this command clears the write protection bit of the addressed group.
CMD30	adtc	[31:0] write protect data address	R1	SEND_WRITE_PROT	If the card provides write protection features, this command asks the card to send the status of the write protection bits.
CMD31	Reserved				

Table 204. Erase commands

CMD index	Type	Argument	Response format	Abbreviation	Description
CMD32 ... CMD34		Reserved. These command indexes cannot be used in order to maintain backward compatibility with older versions of the MultiMediaCard.			
CMD35	ac	[31:0] data address	R1	ERASE_GROUP_START	Sets the address of the first erase group within a range to be selected for erase.
CMD36	ac	[31:0] data address	R1	ERASE_GROUP_END	Sets the address of the last erase group within a continuous range to be selected for erase.
CMD37		Reserved. This command index cannot be used in order to maintain backward compatibility with older versions of the MultiMediaCards			
CMD38	ac	[31:0] stuff bits	R1	ERASE	Erases all previously selected write blocks.

Table 205. I/O mode commands

CMD index	Type	Argument	Response format	Abbreviation	Description
CMD39	ac	[31:16] RCA [15:15] register write flag [14:8] register address [7:0] register data	R4	FAST_IO	Used to write and read 8-bit (register) data fields. The command addresses a card and a register and provides the data for writing if the write flag is set. The R4 response contains data read from the addressed register. This command accesses application-dependent registers that are not defined in the MultiMediaCard standard.

Table 205. I/O mode commands (continued)

CMD index	Type	Argument	Response format	Abbreviation	Description
CMD40	bcr	[31:0] stuff bits	R5	GO_IRQ_STATE	Places the system in the interrupt mode.
CMD41	Reserved				

Table 206. Lock card

CMD index	Type	Argument	Response format	Abbreviation	Description
CMD42	adtc	[31:0] stuff bits	R1b	LOCK_UNLOCK	Sets/resets the password or locks/unlocks the card. The size of the data block is set by the SET_BLOCK_LEN command.
CMD43 ... CMD54	Reserved				

Table 207. Application-specific commands

CMD index	Type	Argument	Response format	Abbreviation	Description
CMD55	ac	[31:16] RCA [15:0] stuff bits	R1	APP_CMD	Indicates to the card that the next command bits is an application specific command rather than a standard command
CMD56	adtc	[31:1] stuff bits [0]: RD/WR	-	-	Used either to transfer a data block to the card or to get a data block from the card for general purpose/application-specific commands. The size of the data block shall be set by the SET_BLOCK_LEN command.
CMD57 ... CMD59	Reserved.				
CMD60 ... CMD63	Reserved for manufacturer.				

30.5 Response formats

All responses are sent via the SDMMC command line SDMMC_CMD. The response transmission always starts with the left bit of the bit string corresponding to the response code word. The code length depends on the response type.

A response always starts with a start bit (always 0), followed by the bit indicating the direction of transmission (card = 0). A value denoted by x in the tables below indicates a variable entry. All responses, except for the R3 response type, are protected by a CRC. Every command code word is terminated by the end bit (always 1).

There are five types of responses. Their formats are defined as follows:

30.5.1 R1 (normal response command)

Code length = 48 bits. The 45:40 bits indicate the index of the command to be responded to, this value being interpreted as a binary-coded number (between 0 and 63). The status of the card is coded in 32 bits.

Table 208. R1 response

Bit position	Width (bits)	Value	Description
47	1	0	Start bit
46	1	0	Transmission bit
[45:40]	6	X	Command index
[39:8]	32	X	Card status
[7:1]	7	X	CRC7
0	1	1	End bit

30.5.2 R1b

It is identical to R1 with an optional busy signal transmitted on the data line. The card may become busy after receiving these commands based on its state prior to the command reception.

30.5.3 R2 (CID, CSD register)

Code length = 136 bits. The contents of the CID register are sent as a response to the CMD2 and CMD10 commands. The contents of the CSD register are sent as a response to CMD9. Only the bits [127...1] of the CID and CSD are transferred, the reserved bit [0] of these registers is replaced by the end bit of the response. The card indicates that an erase is in progress by holding SDMMC_D0 low. The actual erase time may be quite long, and the host may issue CMD7 to deselect the card.

Table 209. R2 response

Bit position	Width (bits)	Value	Description
135	1	0	Start bit
134	1	0	Transmission bit
[133:128]	6	'111111'	Command index
[127:1]	127	X	Card status
0	1	1	End bit

30.5.4 R3 (OCR register)

Code length: 48 bits. The contents of the OCR register are sent as a response to CMD1. The level coding is as follows: restricted voltage windows = low, card busy = low.

Table 210. R3 response

Bit position	Width (bits)	Value	Description
47	1	0	Start bit
46	1	0	Transmission bit
[45:40]	6	'111111'	Reserved
[39:8]	32	X	OCR register
[7:1]	7	'1111111'	Reserved
0	1	1	End bit

30.5.5 R4 (Fast I/O)

Code length: 48 bits. The argument field contains the RCA of the addressed card, the register address to be read out or written to, and its content.

Table 211. R4 response

Bit position	Width (bits)	Value	Description
47	1	0	Start bit
46	1	0	Transmission bit
[45:40]	6	'100111'	CMD39
[39:8] Argument field	[31:16]	16	RCA
	[15:8]	8	register address
	[7:0]	8	read register contents
[7:1]	7	X	CRC7
0	1	1	End bit

30.5.6 R4b

For SD I/O only: an SDIO card receiving the CMD5 will respond with a unique SDIO response R4. The format is:

Table 212. R4b response

Bit position	Width (bits)	Value	Description
47	1	0	Start bit
46	1	0	Transmission bit
[45:40]	6	X	Reserved

Table 212. R4b response (continued)

Bit position	Width (bits)	Value	Description
[39:8] Argument field	39	16	X Card is ready
	[38:36]	3	X Number of I/O functions
	35	1	X Present memory
	[34:32]	3	X Stuff bits
	[31:8]	24	X I/O ORC
[7:1]	7	X	Reserved
0	1	1	End bit

Once an SD I/O card has received a CMD5, the I/O portion of that card is enabled to respond normally to all further commands. This I/O enable of the function within the I/O card will remain set until a reset, power cycle or CMD52 with write to I/O reset is received by the card. Note that an SD memory-only card may respond to a CMD5. The proper response for a memory-only card would be *Present memory* = 1 and *Number of I/O functions* = 0. A memory-only card built to meet the SD Memory Card specification version 1.0 would detect the CMD5 as an illegal command and not respond. The I/O aware host will send CMD5. If the card responds with response R4, the host determines the card's configuration based on the data contained within the R4 response.

30.5.7 R5 (interrupt request)

Only for MultiMediaCard. Code length: 48 bits. If the response is generated by the host, the RCA field in the argument will be 0x0.

Table 213. R5 response

Bit position	Width (bits)	Value	Description
47	1	0	Start bit
46	1	0	Transmission bit
[45:40]	6	'101000'	CMD40
[39:8] Argument field	[31:16]	16	X RCA [31:16] of winning card or of the host
	[15:0]	16	X Not defined. May be used for IRQ data
[7:1]	7	X	CRC7
0	1	1	End bit

30.5.8 R6

Only for SD I/O. The normal response to CMD3 by a memory device. It is shown in [Table 214](#).

Table 214. R6 response

Bit position	Width (bits)	Value	Description
47	1	0	Start bit
46	1	0	Transmission bit
[45:40]	6	'101000'	CMD40
[39:8] Argument field	[31:16]	16	RCA [31:16] of winning card or of the host
	[15:0]	16	Not defined. May be used for IRQ data
[7:1]	7	X	CRC7
0	1	1	End bit

The card [23:8] status bits are changed when CMD3 is sent to an I/O-only card. In this case, the 16 bits of response are the SD I/O-only values:

- Bit [15] COM_CRC_ERROR
- Bit [14] ILLEGAL_COMMAND
- Bit [13] ERROR
- Bits [12:0] Reserved

30.6 SDIO I/O card-specific operations

The following features are SD I/O-specific operations:

- SDIO read wait operation by SDMMC_D2 signalling
- SDIO read wait operation by stopping the clock
- SDIO suspend/resume operation (write and read suspend)
- SDIO interrupts

The SDMMC supports these operations only if the SDMMC_DCTRL[11] bit is set, except for read suspend that does not need specific hardware implementation.

30.6.1 SDIO I/O read wait operation by SDMMC_D2 signalling

It is possible to start the readwait interval before the first block is received: when the data path is enabled (SDMMC_DCTRL[0] bit set), the SDIO-specific operation is enabled (SDMMC_DCTRL[11] bit set), read wait starts (SDMMC_DCTRL[10] =0 and SDMMC_DCTRL[8] =1) and data direction is from card to SDMMC (SDMMC_DCTRL[1] = 1), the DPSM directly moves from Idle to Readwait. In Readwait the DPSM drives SDMMC_D2 to 0 after 2 SDMMC_CK clock cycles. In this state, when you set the RWSTOP bit (SDMMC_DCTRL[9]), the DPSM remains in Wait for two more SDMMC_CK clock cycles to drive SDMMC_D2 to 1 for one clock cycle (in accordance with SDIO specification). The DPSM then starts waiting again until it receives data from the card. The DPSM will not start a readwait interval while receiving a block even if read wait start is set: the readwait interval will start after the CRC is received. The RWSTOP bit has to be cleared to start a new read wait operation. During the readwait interval, the SDMMC can detect SDIO interrupts on SDMMC_D1.

30.6.2 SDIO read wait operation by stopping SDMMC_CK

If the SDIO card does not support the previous read wait method, the SDMMC can perform a read wait by stopping SDMMC_CK (SDMMC_DCTRL is set just like in the method presented in [Section 30.6.1](#), but SDMMC_DCTRL[10] =1): DPSM stops the clock two SDMMC_CK cycles after the end bit of the current received block and starts the clock again after the read wait start bit is set.

As SDMMC_CK is stopped, any command can be issued to the card. During a read/wait interval, the SDMMC can detect SDIO interrupts on SDMMC_D1.

30.6.3 SDIO suspend/resume operation

While sending data to the card, the SDMMC can suspend the write operation. the SDMMC_CMD[11] bit is set and indicates to the CPSM that the current command is a suspend command. The CPSM analyzes the response and when the ACK is received from the card (suspend accepted), it acknowledges the DPSM that goes Idle after receiving the CRC token of the current block.

The hardware does not save the number of the remaining block to be sent to complete the suspended operation (resume).

The write operation can be suspended by software, just by disabling the DPSM (SDMMC_DCTRL[0] =0) when the ACK of the suspend command is received from the card. The DPSM enters then the Idle state.

To suspend a read: the DPSM waits in the Wait_r state as the function to be suspended sends a complete packet just before stopping the data transaction. The application continues reading RxFIFO until the FIFO is empty, and the DPSM goes Idle automatically.

30.6.4 SDIO interrupts

SDIO interrupts are detected on the SDMMC_D1 line once the SDMMC_DCTRL[11] bit is set.

When SDIO interrupt is detected, SDMMC_STA[22] (SDIOIT) bit is set. This static bit can be cleared with clear bit SDMMC_ICR[22] (SDIOITC). An interrupt can be generated when SDIOIT status bit is set. Separated interrupt enable SDMMC_MASK[22] bit (SDIOITE) is available to enable and disable interrupt request.

When SD card interrupt occurs (SDMMC_STA[22] bit set), host software follows below steps to handle it.

1. Disable SDIOIT interrupt signaling by clearing SDIOITE bit (SDMMC_MASK[22] = '0'),
2. Serve card interrupt request, and clear the source of interrupt on the SD card,
3. Clear SDIOIT bit by writing '1' to SDIOITC bit (SDMMC_ICR[22] = '1'),
4. Enable SDIOIT interrupt signaling by writing '1' to SDIOITE bit (SDMMC_MASK[22] = '1').

Steps 2 to 4 can be executed out of the SDIO interrupt service routine.

30.7 HW flow control

The HW flow control functionality is used to avoid FIFO underrun (TX mode) and overrun (RX mode) errors.

The behavior is to stop SDMMC_CK and freeze SDMMC state machines. The data transfer is stalled while the FIFO is unable to transmit or receive data. Only state machines clocked by SDMMCCLK are frozen, the APB2 interface is still alive. The FIFO can thus be filled or emptied even if flow control is activated.

To enable HW flow control, the SDMMC_CLKCR[14] register bit must be set to 1. After reset Flow Control is disabled.

30.8 SDMMC registers

The device communicates to the system via 32-bit-wide control registers accessible via APB2.

30.8.1 SDMMC power control register (SDMMC_POWER)

Address offset: 0x00

Reset value: 0x0000 0000

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

Bits 31:2 Reserved, must be kept at reset value.

[1:0] **PWRCTRL:** Power supply control bits.

These bits are used to define the current functional state of the card clock:

00: Power-off: the clock to card is stopped.

01: Reserved

10: Reserved power-up

11: Power-on: the card is clocked.

Note: At least seven PCLK2 clock periods are needed between two write accesses to this register.

Note: After a data write, data cannot be written to this register for three SDMMCCLK clock periods plus two PCLK2 clock periods.

30.8.2 SDMMC clock control register (SDMMC_CLKCR)

Address offset: 0x04

Reset value: 0x0000 0000

The SDMMC_CLKCR register controls the SDMMC_CK output clock.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	HWFC_EN	NEGEDGE	WIDBUS	BYPASS	PWRSAV	CLKEN	CLKDIV								
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bit 14 **HWFC_EN:** HW Flow Control enable

0b: HW Flow Control is disabled

1b: HW Flow Control is enabled

When HW Flow Control is enabled, the meaning of the TXFIFOE and RXFIFOF interrupt signals, see SDMMC Status register definition in [Section 30.8.11](#).

Bit 13 **NEGEDGE:** SDMMC_CK dephasing selection bit

0b: Command and Data changed on the SDMMCCLK falling edge succeeding the rising edge of SDMMC_CK. (SDMMC_CK rising edge occurs on SDMMCCLK rising edge).

1b: Command and Data changed on the SDMMC_CK falling edge.

When BYPASS is active, the data and the command change on SDMMCCLK falling edge whatever NEGEDGE value.

Bits 12:11 **WIDBUS:** Wide bus mode enable bit

00: Default bus mode: SDMMC_D0 used

01: 4-wide bus mode: SDMMC_D[3:0] used

10: 8-wide bus mode: SDMMC_D[7:0] used

Bit 10 **BYPASS:** Clock divider bypass enable bit

0: Disable bypass: SDMMCCLK is divided according to the CLKDIV value before driving the SDMMC_CK output signal.

1: Enable bypass: SDMMCCLK directly drives the SDMMC_CK output signal.

Bit 9 **PWRSAV:** Power saving configuration bit

For power saving, the SDMMC_CK clock output can be disabled when the bus is idle by setting PWRSAV:

0: SDMMC_CK clock is always enabled

1: SDMMC_CK is only enabled when the bus is active

Bit 8 **CLKEN:** Clock enable bit

0: SDMMC_CK is disabled

1: SDMMC_CK is enabled

Bits 7:0 **CLKDIV:** Clock divide factor

This field defines the divide factor between the input clock (SDMMCCLK) and the output clock (SDMMC_CK): SDMMC_CK frequency = SDMMCCLK / [CLKDIV + 2].

- Note:
- 1 While the SD/SDIO card or MultiMediaCard is in identification mode, the SDMMC_CK frequency must be less than 400 kHz.
 - 2 The clock frequency can be changed to the maximum card bus frequency when relative card addresses are assigned to all cards.
 - 3 After a data write, data cannot be written to this register for three SDMMCCLK clock periods plus two PCLK2 clock periods. SDMMC_CK can also be stopped during the read wait interval for SD I/O cards: in this case the SDMMC_CLKCR register does not control SDMMC_CK.

30.8.3 SDMMC argument register (SDMMC_ARG)

Address offset: 0x08

Reset value: 0x0000 0000

The SDMMC_ARG register contains a 32-bit command argument, which is sent to a card as part of a command message.

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

Bits 31:0 **CMDARG:** Command argument

Command argument sent to a card as part of a command message. If a command contains an argument, it must be loaded into this register before writing a command to the command register.

30.8.4 SDMMC command register (SDMMC_CMD)

Address offset: 0x0C

Reset value: 0x0000 0000

The SDMMC_CMD register contains the command index and command type bits. The command index is sent to a card as part of a command message. The command type bits control the command path state machine (CPSM).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	SDIO Suspend	CPSM EN	WAIT PEND	WAIT INT	WAITRESP		CMDINDEX					
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:12 Reserved, must be kept at reset value.

Bit 11 **SDIOSuspend:** SD I/O suspend command

If this bit is set, the command to be sent is a suspend command (to be used only with SDIO card).

Bit 10 **CPSMEN:** Command path state machine (CPSM) Enable bit

If this bit is set, the CPSM is enabled.

Bit 9 **WAITPEND:** CPSM Waits for ends of data transfer (CmdPend internal signal).

If this bit is set, the CPSM waits for the end of data transfer before it starts sending a command. This feature is available only with Stream data transfer mode SDMMC_DCTRL[2] = 1.

Bit 8 **WAITINT**: CPSM waits for interrupt request

If this bit is set, the CPSM disables command timeout and waits for an interrupt request.

Bits 7:6 **WAITRESP**: Wait for response bits

They are used to configure whether the CPSM is to wait for a response, and if yes, which kind of response.

00: No response, expect CMDSENT flag

01: Short response, expect CMDREND or CCRCFAIL flag

10: No response, expect CMDSENT flag

11: Long response, expect CMDREND or CCRCFAIL flag

Bits 5:0 **CMDINDEX**: Command index

The command index is sent to the card as part of a command message.

- Note:**
- 1 *After a data write, data cannot be written to this register for three SDMMCCLK clock periods plus two PCLK2 clock periods.*
 - 2 *MultiMediaCards can send two kinds of response: short responses, 48 bits long, or long responses, 136 bits long. SD card and SD I/O card can send only short responses, the argument can vary according to the type of response: the software will distinguish the type of response according to the sent command.*

30.8.5 SDMMC command response register (SDMMC_RESPCMD)

Address offset: 0x10

Reset value: 0x0000 0000

The SDMMC_RESPCMD register contains the command index field of the last command response received. If the command response transmission does not contain the command index field (long or OCR response), the RESPCMD field is unknown, although it must contain 111111b (the value of the reserved field from the response).

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

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

Bits 5:0 **RESPCMD**: Response command index

Read-only bit field. Contains the command index of the last command response received.

30.8.6 SDMMC response 1..4 register (SDMMC_RESPx)

Address offset: (0x10 + (4 × x)); x = 1..4

Reset value: 0x0000 0000

The SDMMC_RESP1/2/3/4 registers contain the status of a card, which is part of the received response.

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

Bits 31:0 **CARDSTATUSx**: see [Table 215](#).

The Card Status size is 32 or 127 bits, depending on the response type.

Table 215. Response type and SDMMC_RESPx registers

Register	Short response	Long response
SDMMC_RESP1	Card Status[31:0]	Card Status [127:96]
SDMMC_RESP2	Unused	Card Status [95:64]
SDMMC_RESP3	Unused	Card Status [63:32]
SDMMC_RESP4	Unused	Card Status [31:1]0b

The most significant bit of the card status is received first. The SDMMC_RESP4 register LSB is always 0b.

30.8.7 SDMMC data timer register (SDMMC_DTIMER)

Address offset: 0x24

Reset value: 0x0000 0000

The SDMMC_DTIMER register contains the data timeout period, in card bus clock periods.

A counter loads the value from the SDMMC_DTIMER register, and starts decrementing when the data path state machine (DPSM) enters the Wait_R or Busy state. If the timer reaches 0 while the DPSM is in either of these states, the timeout status flag is set.

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

Bits 31:0 **DATETIME**: Data timeout period

Data timeout period expressed in card bus clock periods.

Note: A data transfer must be written to the data timer register and the data length register before being written to the data control register.

30.8.8 SDMMC data length register (SDMMC_DLEN)

Address offset: 0x28

Reset value: 0x0000 0000

The SDMMC_DLEN register contains the number of data bytes to be transferred. The value is loaded into the data counter when data transfer starts.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16					
Res.	Res.	Res.	Res.	Res.	Res.	Res.	DATALENGTH[24:16]													
							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					
DATALENGTH[15:0]																				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw					

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

Bits 24:0 **DATALENGTH**: Data length value

Number of data bytes to be transferred.

Note: For a block data transfer, the value in the data length register must be a multiple of the block size (see SDMMC_DCTRL). Before being written to the data control register a timeout must be written to the data timer register and the data length register.

In case of IO_RW_EXTENDED (CMD53):

- If the Stream or SDIO multibyte data transfer is selected the value in the data length register must be between 1 and 512.

- If the Block data transfer is selected the value in the data length register must be between 1*Data block size and 512*Data block size.

30.8.9 SDMMC data control register (SDMMC_DCTRL)

Address offset: 0x2C

Reset value: 0x0000 0000

The SDMMC_DCTRL register control the data path state machine (DPSM).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	SDIO EN	RW MOD	RW STOP	RW START	DBLOCKSIZE				DMA EN	DT MODE	DTDIF	DTEN
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:12 Reserved, must be kept at reset value.

Bit 11 **SDIOEN:** SD I/O enable functions

If this bit is set, the DPSM performs an SD I/O-card-specific operation.

Bit 10 **RWMOD:** Read wait mode

- 0: Read Wait control stopping SDMMC_D2
- 1: Read Wait control using SDMMC_CK

Bit 9 **RWSTOP:** Read wait stop

- 0: Read wait in progress if RWSTART bit is set
- 1: Enable for read wait stop if RWSTART bit is set

Bit 8 **RWSTART:** Read wait start

If this bit is set, read wait operation starts.

Bits 7:4 **DBLOCKSIZE:** Data block size

Define the data block length when the block data transfer mode is selected:

- 0000: (0 decimal) lock length = 2^0 = 1 byte
- 0001: (1 decimal) lock length = 2^1 = 2 bytes
- 0010: (2 decimal) lock length = 2^2 = 4 bytes
- 0011: (3 decimal) lock length = 2^3 = 8 bytes
- 0100: (4 decimal) lock length = 2^4 = 16 bytes
- 0101: (5 decimal) lock length = 2^5 = 32 bytes
- 0110: (6 decimal) lock length = 2^6 = 64 bytes
- 0111: (7 decimal) lock length = 2^7 = 128 bytes
- 1000: (8 decimal) lock length = 2^8 = 256 bytes
- 1001: (9 decimal) lock length = 2^9 = 512 bytes
- 1010: (10 decimal) lock length = 2^{10} = 1024 bytes
- 1011: (11 decimal) lock length = 2^{11} = 2048 bytes
- 1100: (12 decimal) lock length = 2^{12} = 4096 bytes
- 1101: (13 decimal) lock length = 2^{13} = 8192 bytes
- 1110: (14 decimal) lock length = 2^{14} = 16384 bytes
- 1111: (15 decimal) reserved

Bit 3 **DMAEN:** DMA enable bit

- 0: DMA disabled.
- 1: DMA enabled.

Bit 2 **DTMODE:** Data transfer mode selection 1: Stream or SDIO multibyte data transfer.

- 0: Block data transfer
- 1: Stream or SDIO multibyte data transfer

Bit 1 **DTDIR:** Data transfer direction selection

- 0: From controller to card.
- 1: From card to controller.

[0] **DTEN:** Data transfer enabled bit

Data transfer starts if 1b is written to the DTEN bit. Depending on the direction bit, DTDIR, the DPSM moves to the Wait_S, Wait_R state or Readwait if RW Start is set immediately at the beginning of the transfer. It is not necessary to clear the enable bit after the end of a data transfer but the SDMMC_DCTRL must be updated to enable a new data transfer

Note: After a data write, data cannot be written to this register for three SDMMCCLK clock periods plus two PCLK2 clock periods.

The meaning of the DTMODE bit changes according to the value of the SDIOEN bit. When SDIOEN=0 and DTMODE=1, the MultiMediaCard stream mode is enabled, and when SDIOEN=1 and DTMODE=1, the peripheral enables an SDIO multibyte transfer.

30.8.10 SDMMC data counter register (SDMMC_DCOUNT)

Address offset: 0x30

Reset value: 0x0000 0000

The SDMMC_DCOUNT register loads the value from the data length register (see SDMMC_DLEN) when the DPSM moves from the Idle state to the Wait_R or Wait_S state. As data is transferred, the counter decrements the value until it reaches 0. The DPSM then moves to the Idle state and the data status end flag, DATAEND, is set.

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

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

Bits 24:0 **DATACOUNT:** Data count value

When this bit is read, the number of remaining data bytes to be transferred is returned. Write has no effect.

Note: This register should be read only when the data transfer is complete.

30.8.11 SDMMC status register (SDMMC_STA)

Address offset: 0x34

Reset value: 0x0000 0000

The SDMMC_STA register is a read-only register. It contains two types of flag:

- Static flags (bits [23:22,10:0]): these bits remain asserted until they are cleared by writing to the SDMMC Interrupt Clear register (see SDMMC_ICR)
- Dynamic flags (bits [21:11]): these bits change state depending on the state of the underlying logic (for example, FIFO full and empty flags are asserted and deasserted as data while written to the FIFO)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SDIOIT	RXD AVL	TXD AVL	RX FIFOE	TX FIFOE	RX FIFOF	TX FIFOF
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RX FIFO HF															
r	r	r	r	r	r		r	r	r	r	r	r	r	r	r

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

Bit 22 **SDIOIT:** SDIO interrupt received

Bit 21 **RXD AVL:** Data available in receive FIFO

- Bit 20 **TXDAVL:** Data available in transmit FIFO
- Bit 19 **RXFIFOE:** Receive FIFO empty
- Bit 18 **TXFIFOE:** Transmit FIFO empty
When HW Flow Control is enabled, TXFIFOE signals becomes activated when the FIFO contains 2 words.
- Bit 17 **RXFIFOF:** Receive FIFO full
When HW Flow Control is enabled, RXFIFOF signals becomes activated 2 words before the FIFO is full.
- Bit 16 **TXFIFOF:** Transmit FIFO full
- Bit 15 **RXFIFOHF:** Receive FIFO half full: there are at least 8 words in the FIFO
- Bit 14 **TXFIFOHE:** Transmit FIFO half empty: at least 8 words can be written into the FIFO
- Bit 13 **RXACT:** Data receive in progress
- Bit 12 **TXACT:** Data transmit in progress
- Bit 11 **CMDACT:** Command transfer in progress
- Bit 10 **DBCKEND:** Data block sent/received (CRC check passed)
- Bit 9 Reserved, must be kept at reset value.
- Bit 8 **DATAEND:** Data end (data counter, SDIDCOUNT, is zero)
- Bit 7 **CMDSENT:** Command sent (no response required)
- Bit 6 **CMDREND:** Command response received (CRC check passed)
- Bit 5 **RXOVERR:** Received FIFO overrun error
Note: If DMA is used to read SDMMC FIFO (DMAEN bit is set in SDMMC_DCTRL register), user software should disable DMA stream, and then write with '0' (to disable DMA request generation).
- Bit 4 **TXUNDERR:** Transmit FIFO underrun error
Note: If DMA is used to fill SDMMC FIFO (DMAEN bit is set in SDMMC_DCTRL register), user software should disable DMA stream, and then write DMAEN with '0' (to disable DMA request generation).
- Bit 3 **DTIMEOUT:** Data timeout
- Bit 2 **CTIMEOUT:** Command response timeout
The Command TimeOut period has a fixed value of 64 SDMMC_CK clock periods.
- Bit 1 **DCRCFAIL:** Data block sent/received (CRC check failed)
- Bit 0 **CCRCFAIL:** Command response received (CRC check failed)

30.8.12 SDMMC interrupt clear register (SDMMC_ICR)

Address offset: 0x38

Reset value: 0x0000 0000

The SDMMC_ICR register is a write-only register. Writing a bit with 1b clears the corresponding bit in the SDMMC_STA Status register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SDIO ITC	Res.	Res.	Res.	Res.	Res.	Res.
									rw						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	DBCK ENDC	Res.	DATA ENDC	CMD SENTC	CMD REND C	RX OVERR C	TX UNDERR C	DTIME OUTC	CTIME OUTC	DCRC FAILC	CCRC FAILC
					rw		rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bit 22 **SDIOITC:** SDIOIT flag clear bit

Set by software to clear the SDIOIT flag.

- 0: SDIOIT not cleared
- 1: SDIOIT cleared

Bits 21:11 Reserved, must be kept at reset value.

Bit 10 **DBCKENDC:** DBCKEND flag clear bit

Set by software to clear the DBCKEND flag.

- 0: DBCKEND not cleared
- 1: DBCKEND cleared

Bit 9 Reserved, must be kept at reset value.

Bit 8 **DATAENDC:** DATAEND flag clear bit

Set by software to clear the DATAEND flag.

- 0: DATAEND not cleared
- 1: DATAEND cleared

Bit 7 **CMDSENTC:** CMDSENT flag clear bit

Set by software to clear the CMDSENT flag.

- 0: CMDSENT not cleared
- 1: CMDSENT cleared

Bit 6 **CMDRENDC:** CMDREND flag clear bit

Set by software to clear the CMDREND flag.

- 0: CMDREND not cleared
- 1: CMDREND cleared

Bit 5 **RXOVERRC:** RXOVERR flag clear bit

Set by software to clear the RXOVERR flag.

- 0: RXOVERR not cleared
- 1: RXOVERR cleared

Bit 4 **TXUNDERRC:** TXUNDERR flag clear bit

Set by software to clear TXUNDERR flag.

- 0: TXUNDERR not cleared
- 1: TXUNDERR cleared

Bit 3 **DTIMEOUTC:** DTIMEOUT flag clear bit

Set by software to clear the DTIMEOUT flag.

- 0: DTIMEOUT not cleared
- 1: DTIMEOUT cleared

Bit 2 **CTIMEOUTC:** CTIMEOUT flag clear bit

Set by software to clear the CTIMEOUT flag.

0: CTIMEOUT not cleared

1: CTIMEOUT cleared

Bit 1 **DCRCFAILC:** DCRCFAIL flag clear bit

Set by software to clear the DCRCFAIL flag.

0: DCRCFAIL not cleared

1: DCRCFAIL cleared

Bit 0 **CCRCFAILC:** CCRCFAIL flag clear bit

Set by software to clear the CCRCFAIL flag.

0: CCRCFAIL not cleared

1: CCRCFAIL cleared

30.8.13 SDMMC mask register (SDMMC_MASK)

Address offset: 0x3C

Reset value: 0x0000 0000

The interrupt mask register determines which status flags generate an interrupt request by setting the corresponding bit to 1b.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SDIO ITIE	RXD AVLIE	TXD AVLIE	RX FIFO EIE	TX FIFO EIE	RX FIFO FIE	TX FIFO FIE
									rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RX FIFO HFIE	TX FIFO HEIE	RX ACTIE	TX ACTIE	CMD ACTIE	DBCK ENDIE	Res.	DATA ENDIE	CMD SENT IE	CMD REND IE	RX OVERR IE	TX UNDERR IE	DTIME OUTIE	CTIME OUTIE	DCRC FAILIE	CCRC FAILIE
rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bit 22 **SDIOITIE:** SDIO mode interrupt received interrupt enable

Set and cleared by software to enable/disable the interrupt generated when receiving the SDIO mode interrupt.

0: SDIO Mode Interrupt Received interrupt disabled

1: SDIO Mode Interrupt Received interrupt enabled

Bit 21 **RXDAVLIE:** Data available in Rx FIFO interrupt enable

Set and cleared by software to enable/disable the interrupt generated by the presence of data available in Rx FIFO.

0: Data available in Rx FIFO interrupt disabled

1: Data available in Rx FIFO interrupt enabled

Bit 20 **TXDAVLIE:** Data available in Tx FIFO interrupt enable

Set and cleared by software to enable/disable the interrupt generated by the presence of data available in Tx FIFO.

0: Data available in Tx FIFO interrupt disabled

1: Data available in Tx FIFO interrupt enabled

- Bit 19 **RXFIFOEIE:** Rx FIFO empty interrupt enable
Set and cleared by software to enable/disable interrupt caused by Rx FIFO empty.
0: Rx FIFO empty interrupt disabled
1: Rx FIFO empty interrupt enabled
- Bit 18 **TXFIFOEIE:** Tx FIFO empty interrupt enable
Set and cleared by software to enable/disable interrupt caused by Tx FIFO empty.
0: Tx FIFO empty interrupt disabled
1: Tx FIFO empty interrupt enabled
- Bit 17 **RXFIFOFIE:** Rx FIFO full interrupt enable
Set and cleared by software to enable/disable interrupt caused by Rx FIFO full.
0: Rx FIFO full interrupt disabled
1: Rx FIFO full interrupt enabled
- Bit 16 **TXFIFOFIE:** Tx FIFO full interrupt enable
Set and cleared by software to enable/disable interrupt caused by Tx FIFO full.
0: Tx FIFO full interrupt disabled
1: Tx FIFO full interrupt enabled
- Bit 15 **RXFIFOHFIE:** Rx FIFO half full interrupt enable
Set and cleared by software to enable/disable interrupt caused by Rx FIFO half full.
0: Rx FIFO half full interrupt disabled
1: Rx FIFO half full interrupt enabled
- Bit 14 **TXFIFOHEIE:** Tx FIFO half empty interrupt enable
Set and cleared by software to enable/disable interrupt caused by Tx FIFO half empty.
0: Tx FIFO half empty interrupt disabled
1: Tx FIFO half empty interrupt enabled
- Bit 13 **RXACTIE:** Data receive acting interrupt enable
Set and cleared by software to enable/disable interrupt caused by data being received (data receive acting).
0: Data receive acting interrupt disabled
1: Data receive acting interrupt enabled
- Bit 12 **TXACTIE:** Data transmit acting interrupt enable
Set and cleared by software to enable/disable interrupt caused by data being transferred (data transmit acting).
0: Data transmit acting interrupt disabled
1: Data transmit acting interrupt enabled
- Bit 11 **CMDACTIE:** Command acting interrupt enable
Set and cleared by software to enable/disable interrupt caused by a command being transferred (command acting).
0: Command acting interrupt disabled
1: Command acting interrupt enabled
- Bit 10 **DBCKENDIE:** Data block end interrupt enable
Set and cleared by software to enable/disable interrupt caused by data block end.
0: Data block end interrupt disabled
1: Data block end interrupt enabled
- Bit 9 Reserved, must be kept at reset value.

- Bit 8 **DATAENDIE:** Data end interrupt enable
Set and cleared by software to enable/disable interrupt caused by data end.
0: Data end interrupt disabled
1: Data end interrupt enabled
- Bit 7 **CMDSENTIE:** Command sent interrupt enable
Set and cleared by software to enable/disable interrupt caused by sending command.
0: Command sent interrupt disabled
1: Command sent interrupt enabled
- Bit 6 **CMDRENDIE:** Command response received interrupt enable
Set and cleared by software to enable/disable interrupt caused by receiving command response.
0: Command response received interrupt disabled
1: Command Response Received interrupt enabled
- Bit 5 **RXOVERRIE:** Rx FIFO overrun error interrupt enable
Set and cleared by software to enable/disable interrupt caused by Rx FIFO overrun error.
0: Rx FIFO overrun error interrupt disabled
1: Rx FIFO overrun error interrupt enabled
- Bit 4 **TXUNDERRIE:** Tx FIFO underrun error interrupt enable
Set and cleared by software to enable/disable interrupt caused by Tx FIFO underrun error.
0: Tx FIFO underrun error interrupt disabled
1: Tx FIFO underrun error interrupt enabled
- Bit 3 **DTIMEOUTIE:** Data timeout interrupt enable
Set and cleared by software to enable/disable interrupt caused by data timeout.
0: Data timeout interrupt disabled
1: Data timeout interrupt enabled
- Bit 2 **CTIMEOUTIE:** Command timeout interrupt enable
Set and cleared by software to enable/disable interrupt caused by command timeout.
0: Command timeout interrupt disabled
1: Command timeout interrupt enabled
- Bit 1 **DCRCFAILIE:** Data CRC fail interrupt enable
Set and cleared by software to enable/disable interrupt caused by data CRC failure.
0: Data CRC fail interrupt disabled
1: Data CRC fail interrupt enabled
- Bit 0 **CCRCFAILIE:** Command CRC fail interrupt enable
Set and cleared by software to enable/disable interrupt caused by command CRC failure.
0: Command CRC fail interrupt disabled
1: Command CRC fail interrupt enabled

30.8.14 SDMMC FIFO counter register (SDMMC_FIFOCNT)

Address offset: 0x48

Reset value: 0x0000 0000

The SDMMC_FIFOCNT register contains the remaining number of words to be written to or read from the FIFO. The FIFO counter loads the value from the data length register (see SDMMC_DLEN) when the data transfer enable bit, DTEN, is set in the data control register (SDMMC_DCTRL register) and the DPSM is at the Idle state. If the data length is not word-aligned (multiple of 4), the remaining 1 to 3 bytes are regarded as a word.

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

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

Bits 23:0 **FIFOCOUNT**: Remaining number of words to be written to or read from the FIFO.

30.8.15 SDMMC data FIFO register (SDMMC_FIFO)

Address offset: 0x80

Reset value: 0x0000 0000

The receive and transmit FIFOs can be read or written as 32-bit wide registers. The FIFOs contain 32 entries on 32 sequential addresses. This allows the CPU to use its load and store multiple operands to read from/write to the FIFO.

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

bits 31:0 **FIFOData**: Receive and transmit FIFO data

The FIFO data occupies 32 entries of 32-bit words, from address:
SDMMC base + 0x080 to SDMMC base + 0xFC.

30.8.16 SDMMC register map

The following table summarizes the SDMMC registers.

Table 216. SDMMC register map

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	SDMMC_POWER	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	
		Reset value																															
0x04	SDMMC_CLKCR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	
		Reset value																															
0x08	SDMMC_ARG	CMDARG																															
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x0C	SDMMC_CMD	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	
		Reset value																															
0x10	SDMMC_RESPCMD	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	
		Reset value																															
0x14	SDMMC_RESP1	CARDSTATUS1																															
		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	
0x18	SDMMC_RESP2	CARDSTATUS2																															
		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	
0x1C	SDMMC_RESP3	CARDSTATUS3																															
		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	
0x20	SDMMC_RESP4	CARDSTATUS4																															
		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	
0x24	SDMMC_DTIME	DATATIME																															
		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	
0x28	SDMMC_DLEN	DATALENGTH																															
		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	
0x2C	SDMMC_DCTRL	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	
		Reset value																															

Table 216. SDMMC register map (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			
0x30	SDMMC_DCOUNT	Res.																																		
	Reset value																																			
0x34	SDMMC_STA	Res.																																		
	Reset value																																			
0x38	SDMMC_ICR	Res.																																		
	Reset value																																			
0x3C	SDMMC_MASK	Res.																																		
	Reset value																																			
0x48	SDMMC_FIFOCNT	Res.																																		
	Reset value																																			
0x80	SDMMC_FIFO																																			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

Refer to [Section 1.5.2: Memory map and register boundary addresses](#) for the register boundary addresses.

31 Controller area network (bxCAN)

31.1 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.

31.2 bxCAN main features

- Supports CAN protocol version 2.0 A, B Active
- Bit rates up to 1 Mbit/s
- Supports the Time Triggered Communication option

Transmission

- Three transmit mailboxes
- Configurable transmit priority
- Time Stamp on SOF transmission

Reception

- Two receive FIFOs with three stages
- Scalable filter banks:
 - 14 filter banks for single CAN
- Identifier list feature
- Configurable FIFO overrun
- Time Stamp on SOF reception

Time-triggered communication option

- Disable automatic retransmission mode
- 16-bit free running timer
- Time Stamp sent in last two data bytes

Management

- Maskable interrupts
- Software-efficient mailbox mapping at a unique address space

31.3 bxCAN general description

In today CAN applications, the number of nodes in a network is increasing and often several networks are linked together via gateways. Typically the number of messages in the system (to be handled by each node) has significantly increased. In addition to the application messages, Network Management and Diagnostic messages have been introduced.

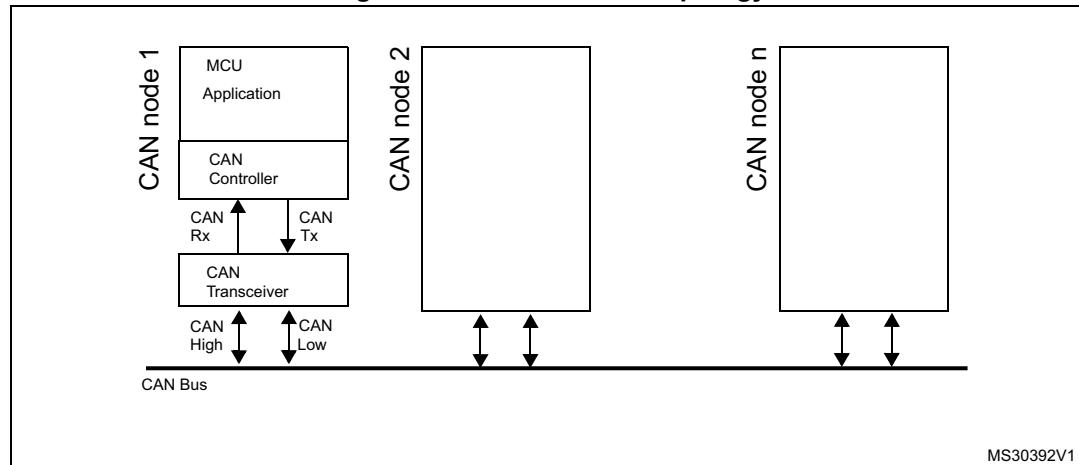
- An enhanced filtering mechanism is required to handle each type of message.

Furthermore, application tasks require more CPU time, therefore real-time constraints caused by message reception have to be reduced.

- A receive FIFO scheme allows the CPU to be dedicated to application tasks for a long time period without losing messages.

The standard HLP (Higher Layer Protocol) based on standard CAN drivers requires an efficient interface to the CAN controller.

Figure 410. CAN network topology



31.3.1 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.

31.3.2 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

31.3.3 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.

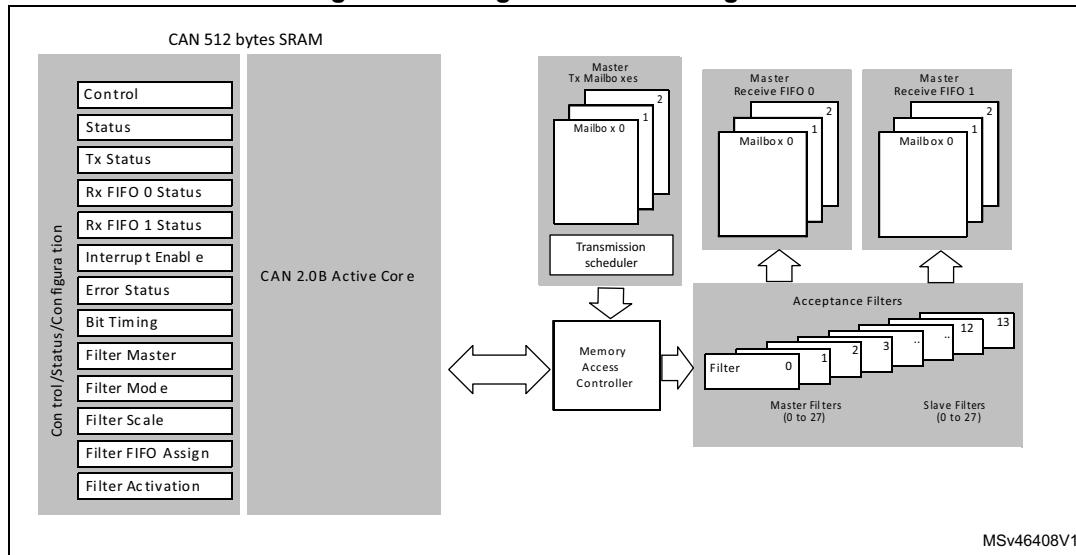
31.3.4 Acceptance filters

The bxCAN provides up to 14 scalable/configurable identifier filter banks in single CAN configuration, for selecting the incoming messages, that the software needs and discarding the others.

Receive FIFO

Two receive FIFOs are used by hardware to store the incoming messages. Three complete messages can be stored in each FIFO. The FIFOs are managed completely by hardware.

Figure 411. Single-CAN block diagram



MSv46408V1

31.4 bxCAN operating modes

bxCAN has three main operating modes: **initialization**, **normal** and **Sleep**. After a hardware reset, bxCAN is in Sleep mode to reduce power consumption and an internal pull-up is active on CANTX. The software requests bxCAN to enter **initialization** or **Sleep** mode by setting the INRQ or SLEEP bits in the CAN_MCR register. Once the mode has been entered, bxCAN confirms it by setting the INAK or SLAK bits in the CAN_MSR register and the internal pull-up is disabled. When neither INAK nor SLAK are set, bxCAN is in **normal** mode. Before entering **normal** mode bxCAN always has to **synchronize** on the CAN bus. To synchronize, bxCAN waits until the CAN bus is idle, this means 11 consecutive recessive bits have been monitored on CANRX.

31.4.1 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).

31.4.2 Normal mode

Once the initialization is complete, the software must request the hardware to enter Normal mode to be able to synchronize on the CAN bus and start reception and transmission.

The request to enter Normal mode is issued by clearing the INRQ bit in the CAN_MCR register. The bxCAN enters Normal mode and is ready to take part in bus activities when it has synchronized with the data transfer on the CAN bus. This is done by waiting for the occurrence of a sequence of 11 consecutive recessive bits (Bus Idle state). The switch to Normal mode is confirmed by the hardware by clearing the INAK bit in the CAN_MSR register.

The initialization of the filter values is independent from Initialization Mode but must be done while the filter is not active (corresponding FACTx bit cleared). The filter scale and mode configuration must be configured before entering Normal Mode.

31.4.3 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.

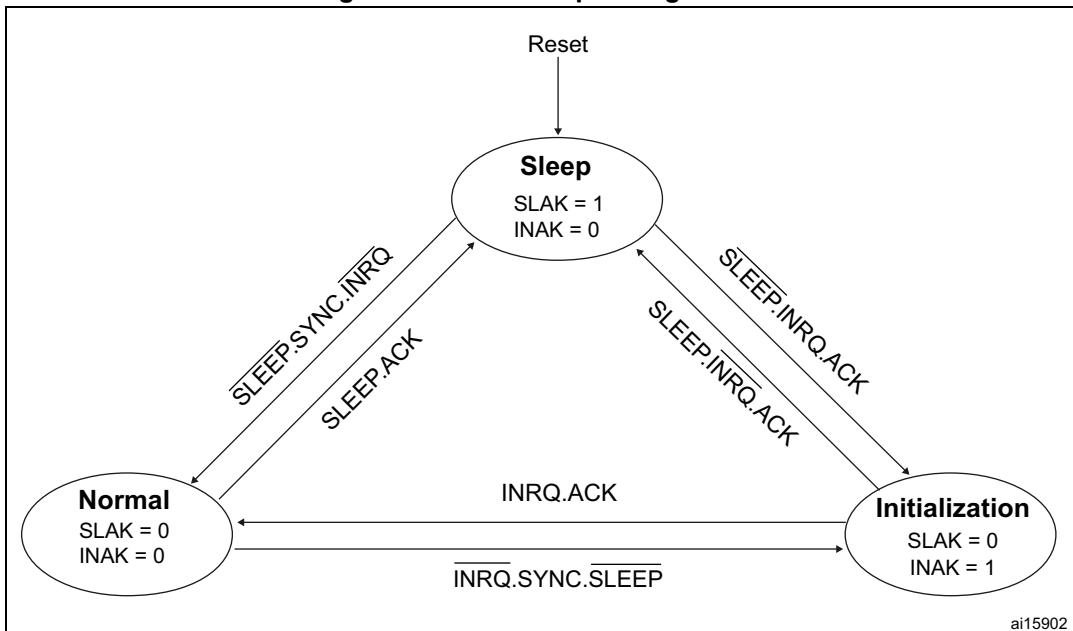
bxCAN can be woken up (exit Sleep mode) either by software clearing the SLEEP bit or on detection of CAN bus activity.

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 412: bxCAN operating modes](#). The Sleep mode is exited once the SLAK bit has been cleared by hardware.

Figure 412. bxCAN operating modes



1. ACK = The wait state during which hardware confirms a request by setting the INAK or SLAK bits in the CAN_MSR register
2. SYNC = The state during which bxCAN waits until the CAN bus is idle, meaning 11 consecutive recessive bits have been monitored on CANRX

31.5 Test mode

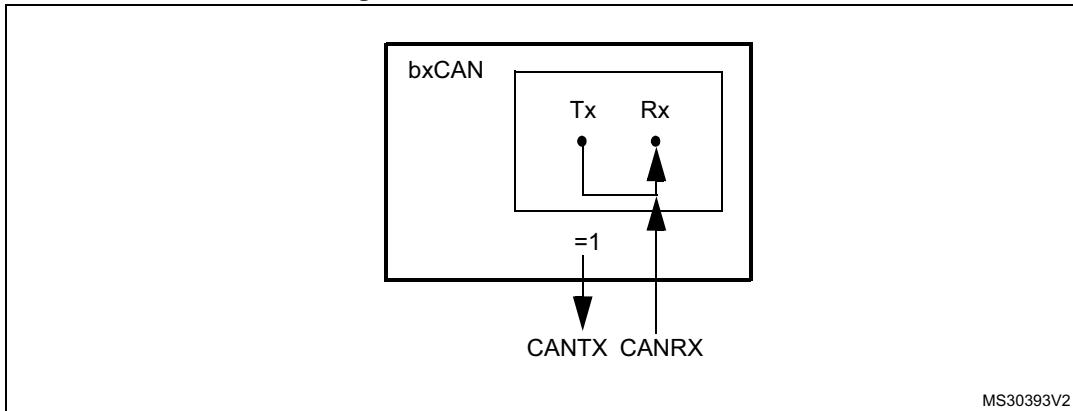
Test mode can be selected by the SILM and LBKM bits in the CAN_BTR register. These bits must be configured while bxCAN is in Initialization mode. Once test mode has been selected, the INRQ bit in the CAN_MCR register must be reset to enter Normal mode.

31.5.1 Silent mode

The bxCAN can be put in Silent mode by setting the SILM bit in the CAN_BTR register.

In Silent mode, the bxCAN is able to receive valid data frames and valid remote frames, but 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).

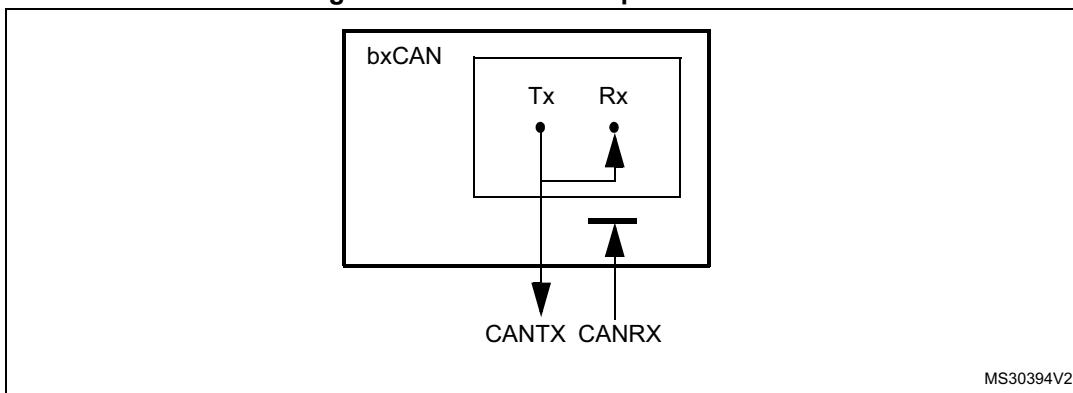
Figure 413. bxCAN in silent mode



31.5.2 Loop back mode

The bxCAN can be set in Loop Back Mode by setting the LBKM bit in the CAN_BTR register. In Loop Back Mode, the bxCAN treats its own transmitted messages as received messages and stores them (if they pass acceptance filtering) in a Receive mailbox.

Figure 414. bxCAN in loop back mode

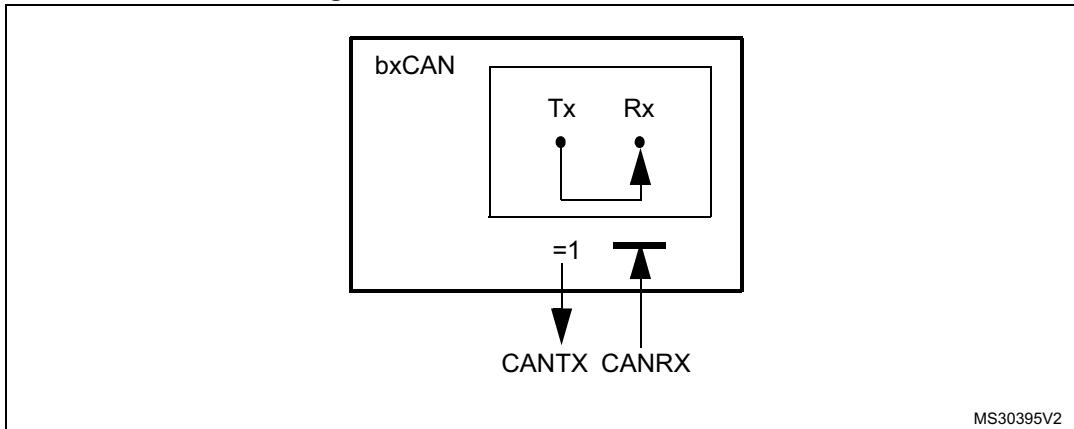


This mode is provided for self-test functions. To be independent of external events, the CAN Core ignores acknowledge errors (no dominant bit sampled in the acknowledge slot of a data / remote frame) in Loop Back Mode. In this mode, the bxCAN performs an internal feedback from its Tx output to its Rx input. The actual value of the CANRX input pin is disregarded by the bxCAN. The transmitted messages can be monitored on the CANTX pin.

31.5.3 Loop back combined with silent mode

It is also possible to combine Loop Back mode and Silent mode by setting the LBKM and 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 415. bxCAN in combined mode



31.6 Behavior in debug mode

When the microcontroller enters the debug mode (Cortex®-M7 core halted), the bxCAN continues to work normally or stops, depending on:

- the DBG_CAN1_STOP bit in the DBG module for the single mode.
- the DBF bit in CAN_MCR. For more details, refer to [Section 31.9.2: CAN control and status registers](#).

31.7 bxCAN functional description

31.7.1 Transmission handling

In order to transmit a message, the application must select one **empty** transmit mailbox, set up the identifier, the data length code (DLC) and the data before requesting the transmission by setting the corresponding TXRQ bit in the CAN_TlxR register. Once the mailbox has left **empty** state, the software no longer has write access to the mailbox registers. Immediately after the TXRQ bit has been set, the mailbox enters **pending** state and waits to become the highest priority mailbox, see *Transmit Priority*. As soon as the mailbox has the highest priority it 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.

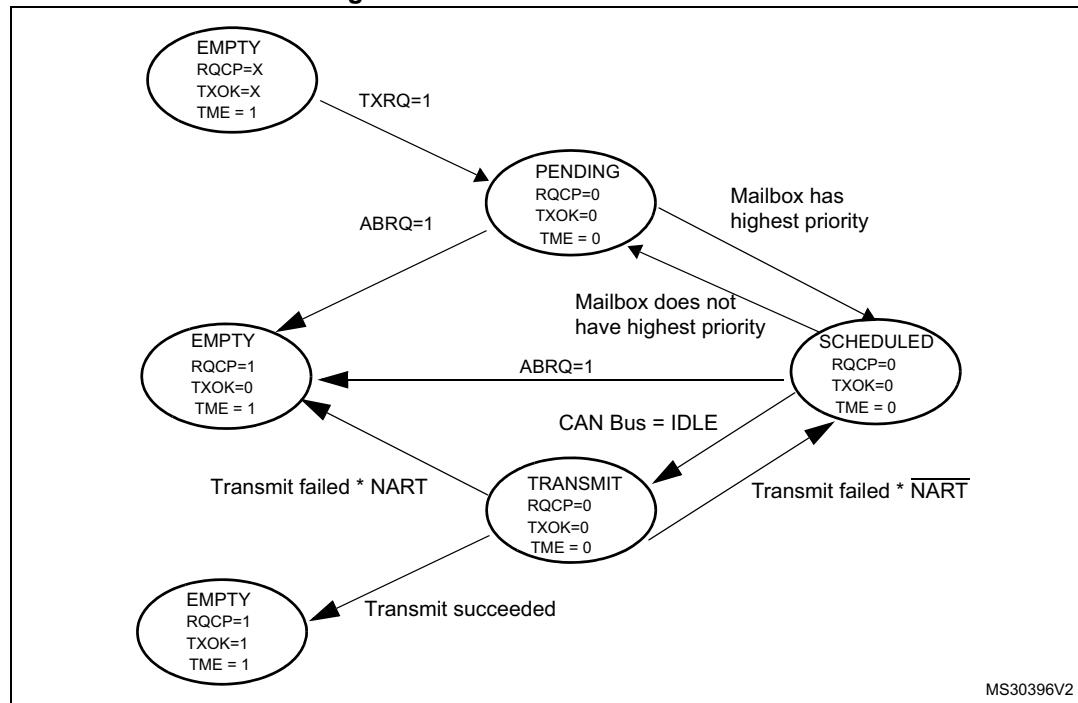
Non automatic retransmission mode

This mode has been implemented in order to fulfill the requirement of the Time Triggered Communication option of the CAN standard. To configure the hardware in this mode the NART bit in the CAN_MCR register must be set.

In this mode, each transmission is started only once. If the first attempt fails, due to an arbitration loss or an error, the hardware 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 416. Transmit mailbox states



31.7.2 Time triggered communication mode

In this mode, the internal counter of the CAN hardware is activated and used to generate the Time Stamp value stored in the CAN_RDTxR/CAN_TDTxR registers, respectively (for Rx and Tx mailboxes). The internal counter is incremented each CAN bit time (refer to [Section 31.7.7: Bit timing](#)). The internal counter is captured on the sample point of the Start Of Frame bit in both reception and transmission.

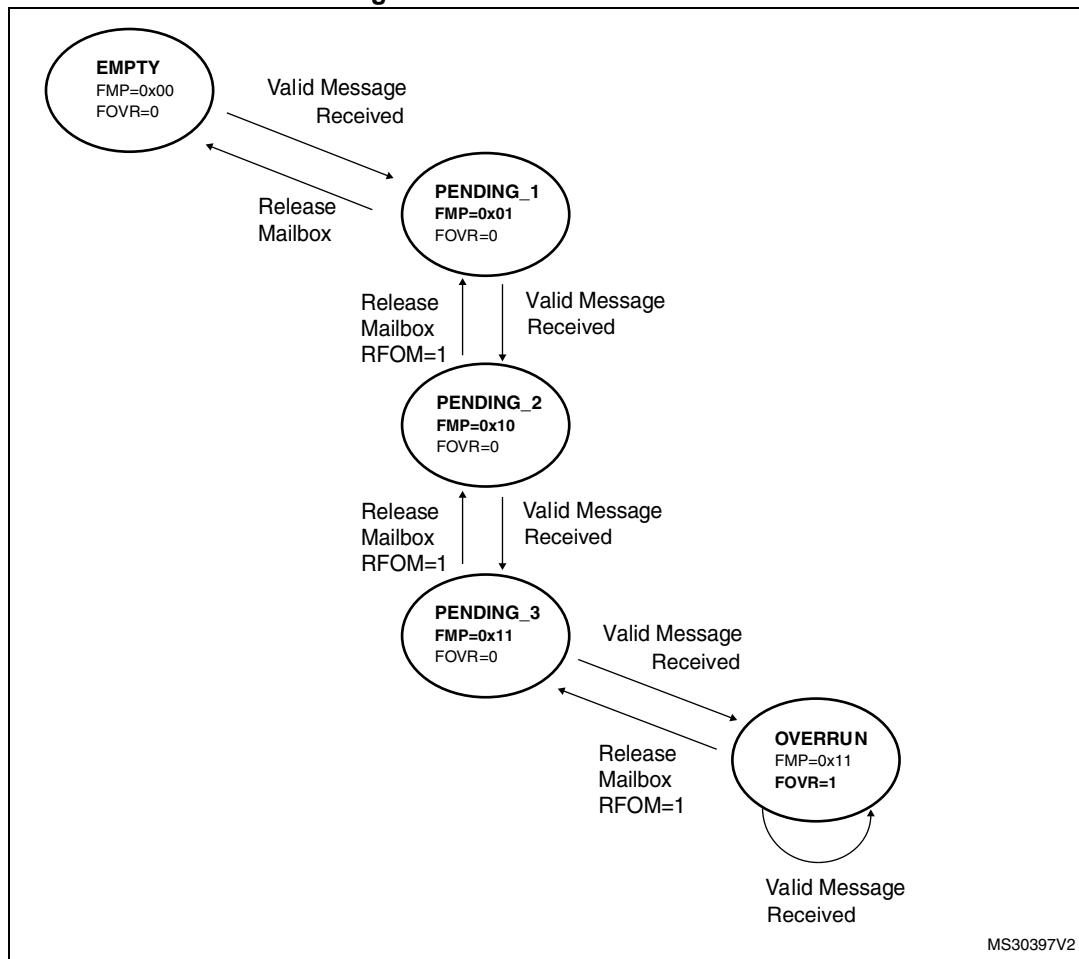
31.7.3 Reception handling

For the reception of CAN messages, three mailboxes organized as a FIFO are provided. In order to save CPU load, simplify the software and guarantee data consistency, the FIFO is managed completely by hardware. The application accesses the messages stored in the FIFO through the FIFO output mailbox.

Valid message

A received message is considered as valid **when** it has been received correctly according to the CAN protocol (no error until the last but one bit of the EOF field) **and** It passed through the identifier filtering successfully, see [Section 31.7.4: Identifier filtering](#).

Figure 417. Receive FIFO states



FIFO management

Starting from the **empty** state, the first valid message received is stored in the FIFO which becomes **pending_1**. The hardware signals the event setting the FMP[1:0] bits in the CAN_RFR register to the value 01b. The message is available in the FIFO output mailbox. The software reads out the mailbox content and releases it by setting the RFOM bit in the CAN_RFR register. The FIFO becomes **empty** again. If a new valid message has been received in the meantime, the FIFO stays in **pending_1** state and the new message is available in the output mailbox.

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 31.7.5: 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 CAN_RFR register. Which message is lost depends on the configuration of the FIFO:

- If the FIFO lock function is disabled (RFLM bit in the CAN_MCR register cleared) the last message stored in the FIFO will be overwritten by the new incoming message. 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 CAN_RFR register is set and an interrupt is generated if the FFIE bit in the CAN_IER register is set.

On overrun condition, the FOVR bit is set and an interrupt is generated if the FOVIE bit in the CAN_IER register is set.

31.7.4 Identifier filtering

In the CAN protocol the identifier of a message is not associated with the address of a node but related to the content of the message. Consequently a transmitter broadcasts its message to all receivers. On message reception a receiver node decides - depending on the identifier value - whether the software needs the message or not. If the message is needed, it is copied into the SRAM. If not, the message must be discarded without intervention by the software.

To fulfill this requirement the bxCAN Controller provides 14 configurable and scalable filter banks (13-0) to the application, in order to receive only the messages the software needs.

This hardware filtering saves CPU resources which would be otherwise needed to perform filtering by software. Each filter bank x consists of two 32-bit registers, CAN_FxR0 and CAN_FxR1.

Scalable width

To optimize and adapt the filters to the application needs, each filter bank can be scaled independently. Depending on the filter scale a filter bank provides:

- One 32-bit filter for the STDID[10:0], EXTID[17:0], IDE and RTR bits.
- Two 16-bit filters for the STDID[10:0], RTR, IDE and EXTID[17:15] bits.

Refer to [Figure 418](#).

Furthermore, the filters can be configured in mask mode or in identifier list mode.

Mask mode

In **mask** mode the identifier registers are associated with mask registers specifying which bits of the identifier are handled as “must match” or as “don’t care”.

Identifier list mode

In **identifier list** mode, the mask registers are used as identifier registers. Thus instead of defining an identifier and a mask, two identifiers are specified, doubling the number of single identifiers. All bits of the incoming identifier must match the bits specified in the filter registers.

Filter bank scale and mode configuration

The filter banks are configured by means of the corresponding CAN_FMR register. To configure a filter bank it must be deactivated by clearing the FACT bit in the CAN_FAR register. The filter scale is configured by means of the corresponding FSCx bit in the CAN_FS1R register, refer to [Figure 418](#). The **identifier list** or **identifier mask** mode for the corresponding Mask/Identifier registers is configured by means of the FBMx bits in the CAN_FMR register.

To filter a group of identifiers, configure the Mask/Identifier registers in mask mode.

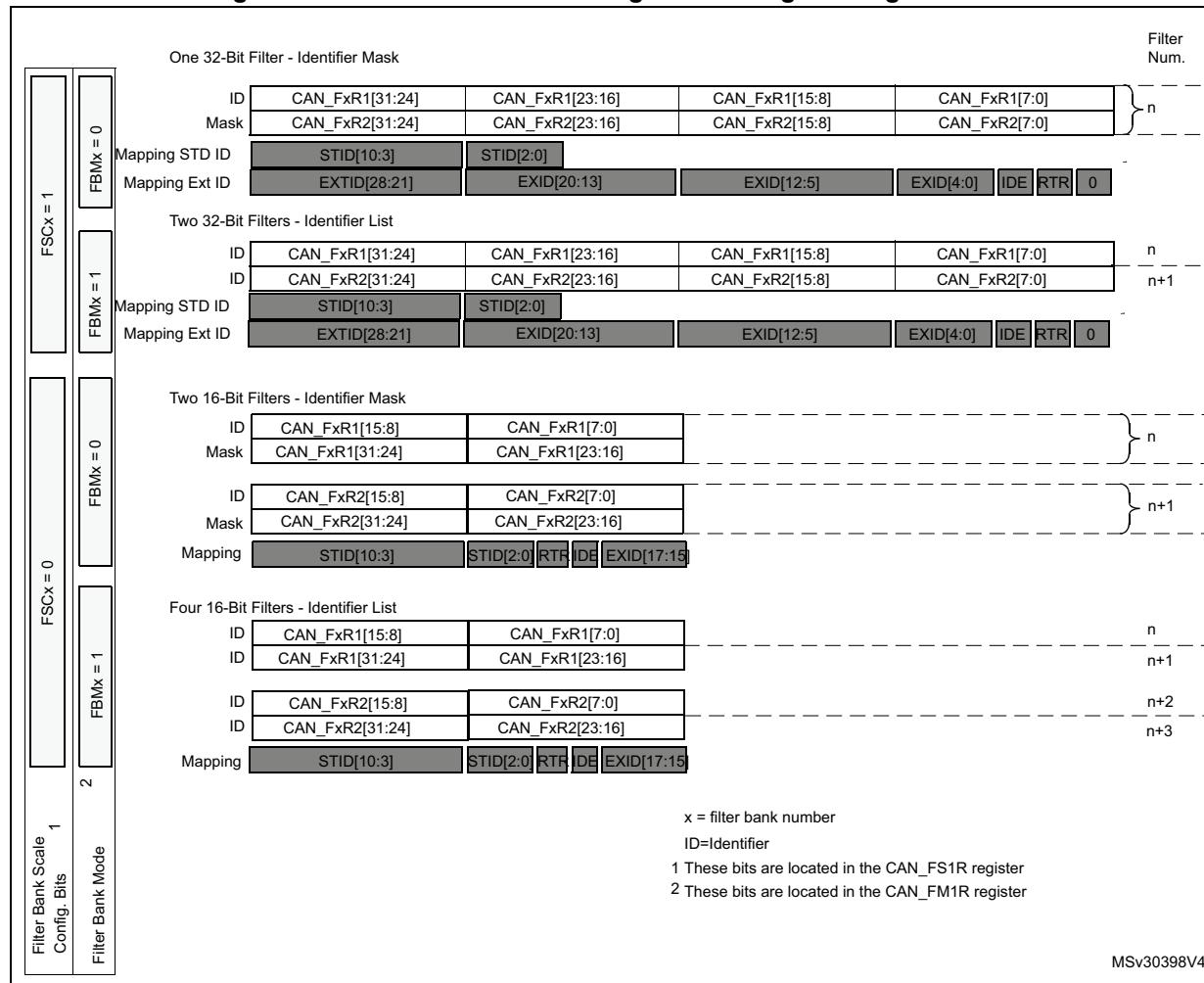
To select single identifiers, configure the Mask/Identifier registers in identifier list mode.

Filters not used by the application should be left deactivated.

Each filter within a filter bank is numbered (called the *Filter Number*) from 0 to a maximum dependent on the mode and the scale of each of the filter banks.

Concerning the filter configuration, refer to [Figure 418](#).

Figure 418. 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 419](#) for an example.

Figure 419. Example of filter numbering

Filter Bank	FIFO0	Filter Num.	Filter Bank	FIFO1	Filter Num.
0	ID List (32-bit)	0 1	2	ID Mask (16-bit)	0 1
1	ID Mask (32-bit)	2	4	ID List (32-bit)	2 3
3	ID List (16-bit)	3 4 5 6	7	Deactivated ID Mask (16-bit)	4 5
5	Deactivated ID List (32-bit)	7 8	8	ID Mask (16-bit)	6 7
6	ID Mask (16-bit)	9 10	10	Deactivated ID List (16-bit)	8 9 10 11
9	ID List (32-bit)	11 12	11	ID List (32-bit)	12 13
13	ID Mask (32-bit)	13	12	ID Mask (32-bit)	14

ID=Identifier

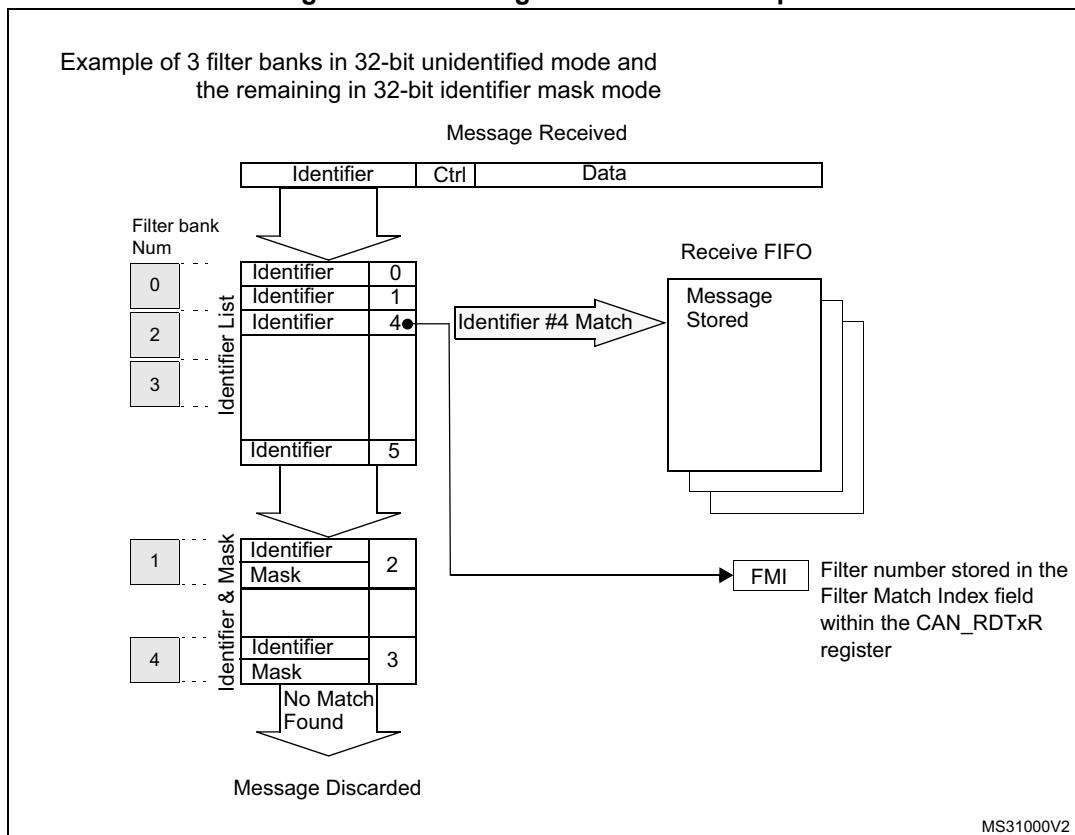
MS30399V2

Filter priority rules

Depending on the filter combination it may occur that an identifier passes successfully through several filters. In this case the filter match value stored in the receive mailbox is chosen according to the following priority rules:

- A 32-bit filter takes priority over a 16-bit filter.
- For filters of equal scale, priority is given to the Identifier List mode over the Identifier Mask mode
- For filters of equal scale and mode, priority is given by the filter number (the lower the number, the higher the priority).

Figure 420. 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.

31.7.5 Message storage

The interface between the software and the hardware for the CAN messages is implemented by means of mailboxes. A mailbox contains all information related to a message; identifier, data, control, status and time stamp information.

Transmit mailbox

The software sets up the message to be transmitted in an empty transmit mailbox. The status of the transmission is indicated by hardware in the CAN_TSR register.

Table 217. Transmit mailbox mapping

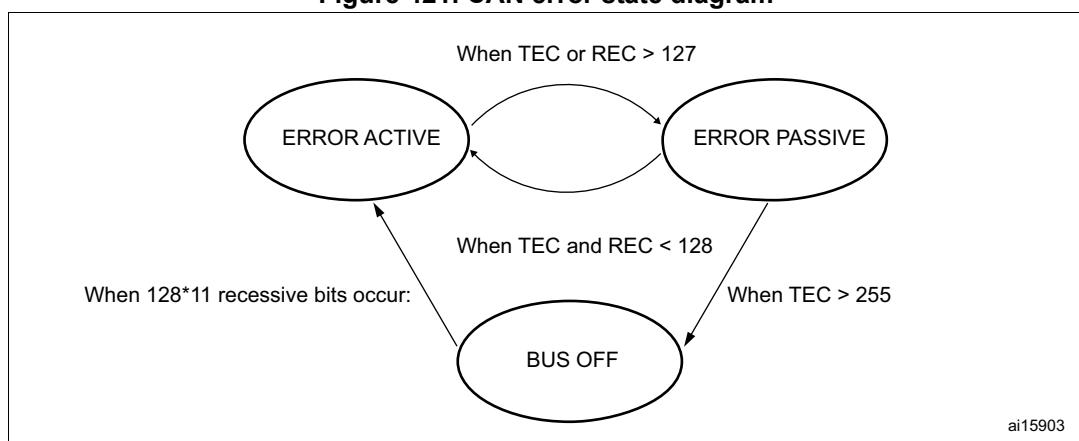
Offset to transmit mailbox base address	Register name
0	CAN_TIxR
4	CAN_TDTxR
8	CAN_TDLxR
12	CAN_TDhxR

Receive mailbox

When a message has been received, it is available to the software in the FIFO output mailbox. Once the software has handled the message (e.g. read it) the software must release the FIFO output mailbox by means of the RFOM bit in the CAN_RFR register to make the next incoming message available. The filter match index is stored in the MFMI field of the CAN_RDTxR register. The 16-bit time stamp value is stored in the TIME[15:0] field of CAN_RDTxR.

Table 218. 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 421. CAN error state diagram

31.7.6 Error management

The error management as described in the CAN protocol is handled entirely by hardware using a Transmit Error Counter (TEC value, in CAN_ESR register) and a Receive Error Counter (REC value, in the CAN_ESR register), which get incremented or decremented according to the error condition. For detailed information about TEC and REC management, refer to the CAN standard.

Both of them may be read by software to determine the stability of the network. Furthermore, the CAN hardware provides detailed information on the current error status in CAN_ESR register. By means of the CAN_IER register (ERRIE bit, etc.), the software can configure the interrupt generation on error detection in a very flexible way.

Bus-Off recovery

The Bus-Off state is reached when TEC is greater than 255, this state is indicated by BOFF bit in CAN_ESR register. In Bus-Off state, the bxCAN is no longer able to transmit and receive messages.

Depending on the ABOM bit in the CAN_MCR register bxCAN 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.*

31.7.7 Bit timing

The bit timing logic monitors the serial bus-line and performs sampling and adjustment of the sample point by synchronizing on the start-bit edge and resynchronizing on the following edges.

Its operation may be explained simply by splitting nominal bit time into three segments as follows:

- **Synchronization segment (SYNC_SEG):** a bit change is expected to occur within this time segment. It has a fixed length of one time quantum ($1 \times t_q$).
- **Bit segment 1 (BS1):** defines the location of the sample point. It includes the PROP_SEG and PHASE_SEG1 of the CAN standard. Its duration is programmable between 1 and 16 time quanta but may be automatically lengthened to compensate for positive phase drifts due to differences in the frequency of the various nodes of the network.
- **Bit segment 2 (BS2):** defines the location of the transmit point. It represents the PHASE_SEG2 of the CAN standard. Its duration is programmable between 1 and 8 time quanta but may also be automatically shortened to compensate for negative phase drifts.

The resynchronization Jump Width (SJW) defines an upper bound to the amount of lengthening or shortening of the bit segments. It is programmable between 1 and 4 time quanta.

A valid edge is defined as the first transition in a bit time from dominant to recessive bus level provided the controller itself does not send a recessive bit.

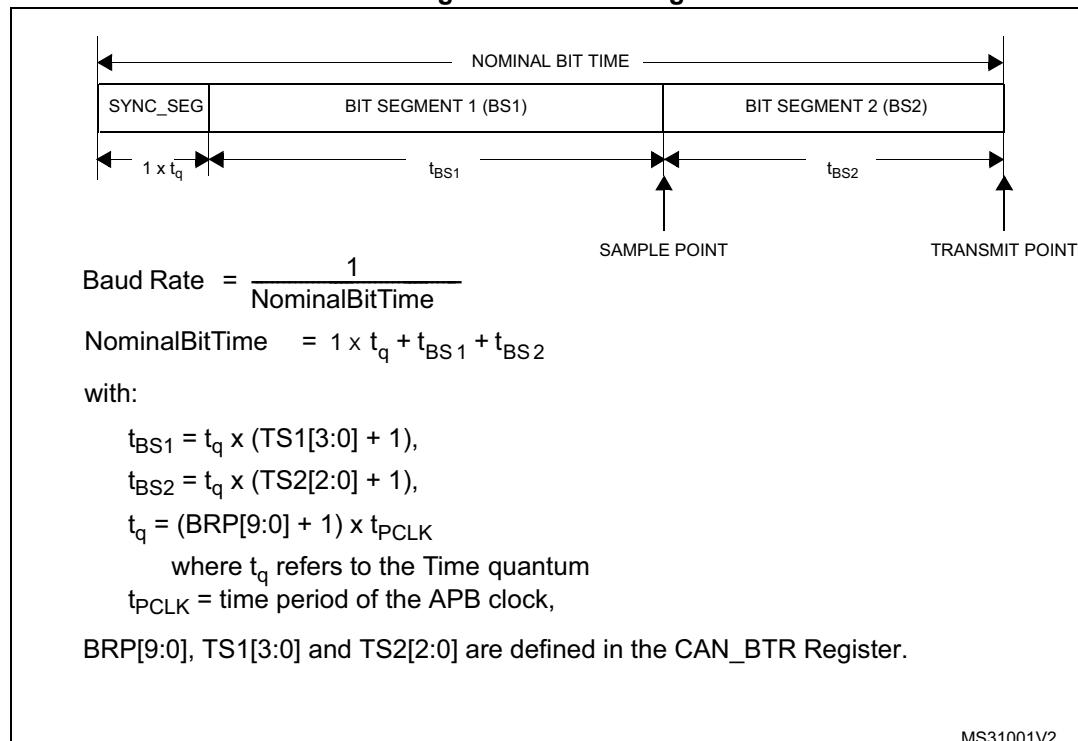
If a valid edge is detected in BS1 instead of SYNC_SEG, BS1 is extended by up to SJW so that the sample point is delayed.

Conversely, if a valid edge is detected in BS2 instead of SYNC_SEG, BS2 is shortened by up to SJW so that the transmit point is moved earlier.

As a safeguard against programming errors, the configuration of the Bit Timing Register (CAN_BTR) is only possible while the device is in Standby mode.

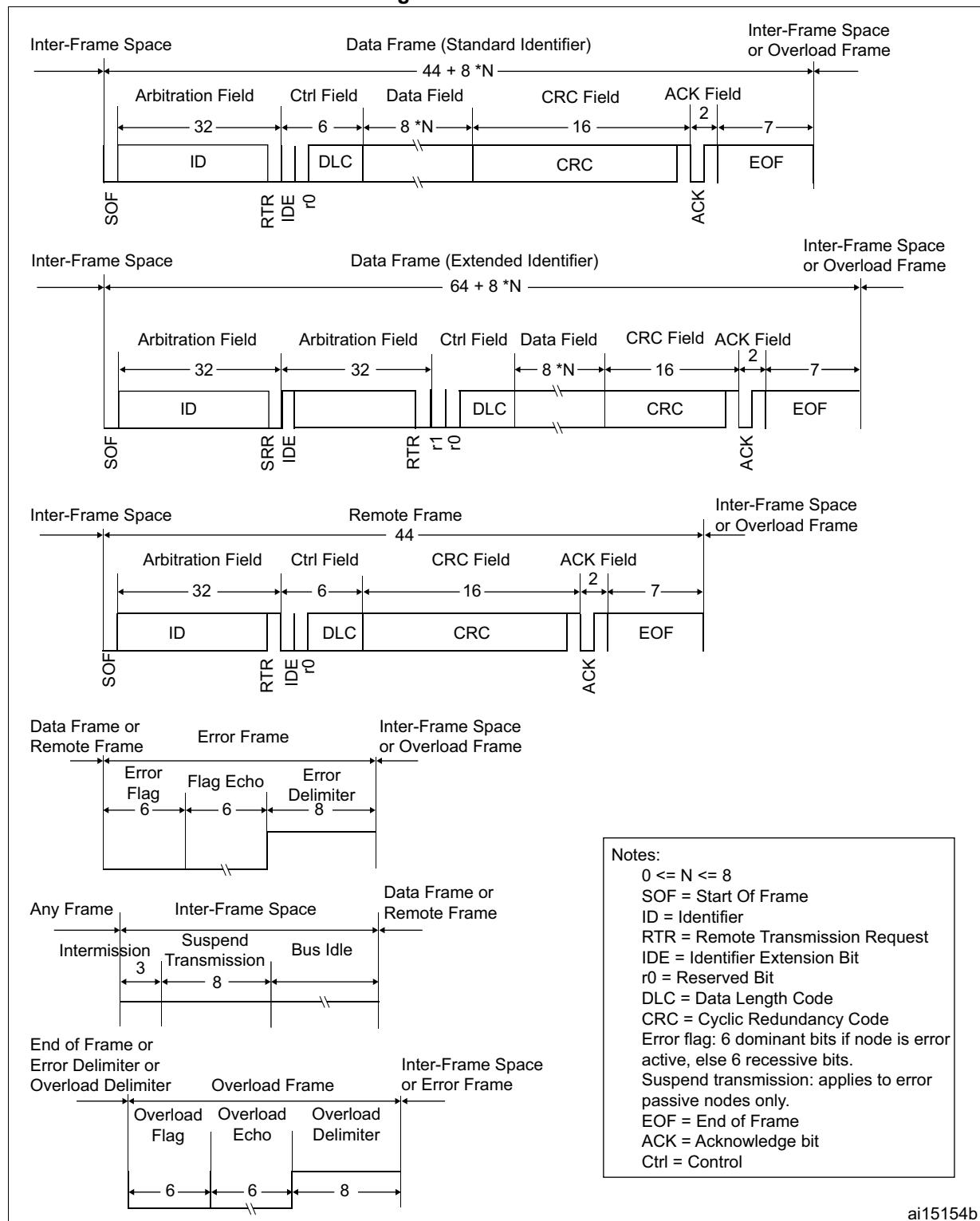
Note: *For a detailed description of the CAN bit timing and resynchronization mechanism, refer to the ISO 11898 standard.*

Figure 422. Bit timing



MS31001V2

Figure 423. CAN frames

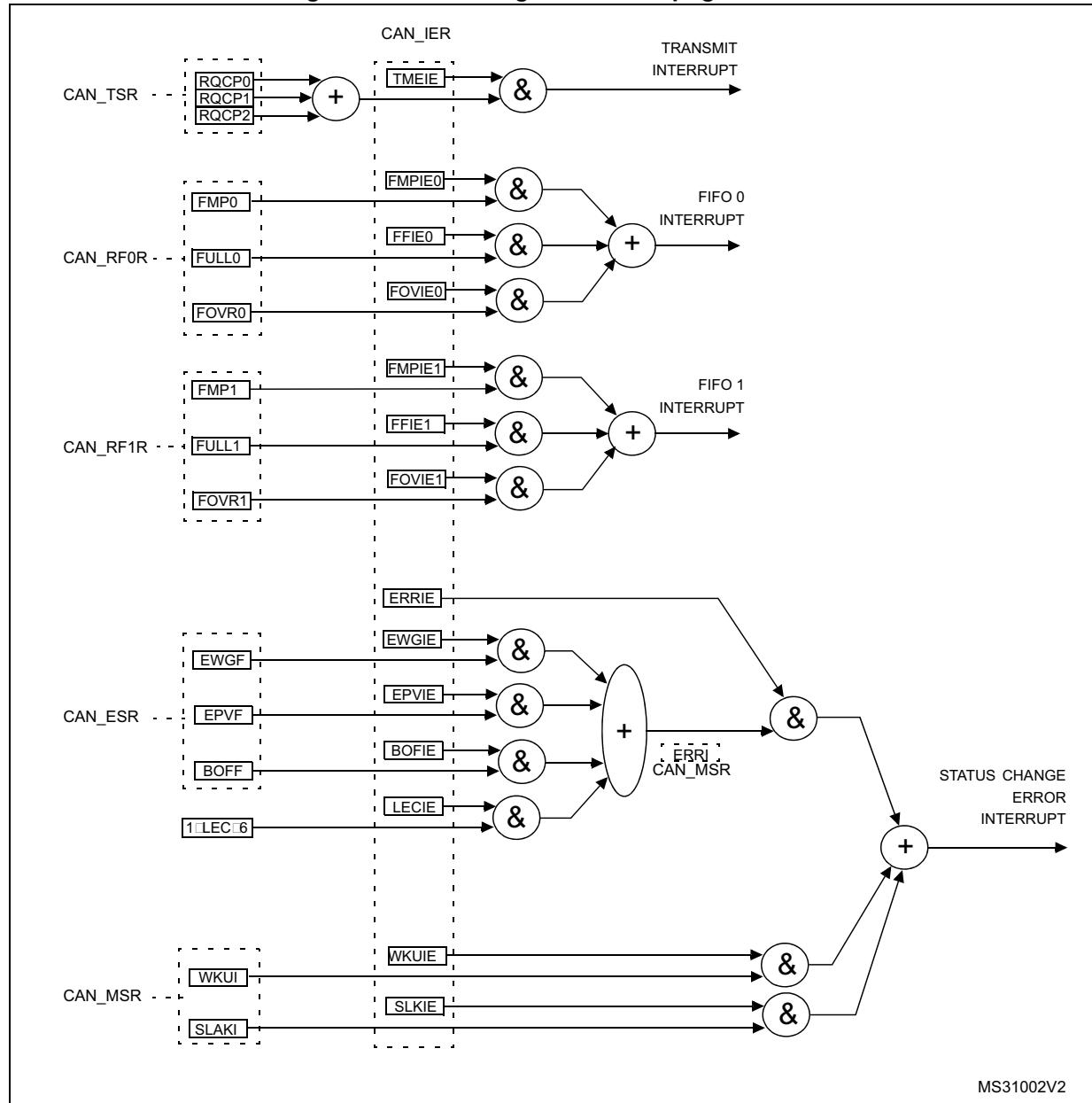


ai15154b

31.8 bxCAN interrupts

Four interrupt vectors are dedicated to bxCAN. Each interrupt source can be independently enabled or disabled by means of the CAN Interrupt Enable Register (CAN_IER).

Figure 424. Event flags and interrupt generation



- The **transmit interrupt** can be generated by the following events:
 - Transmit mailbox 0 becomes empty, RQCP0 bit in the CAN_TSR register set.
 - Transmit mailbox 1 becomes empty, RQCP1 bit in the CAN_TSR register set.
 - Transmit mailbox 2 becomes empty, RQCP2 bit in the CAN_TSR register set.
- The **FIFO 0 interrupt** can be generated by the following events:
 - Reception of a new message, FMP0 bits in the CAN_RF0R register are not '00'.
 - FIFO0 full condition, FULL0 bit in the CAN_RF0R register set.
 - FIFO0 overrun condition, FOVR0 bit in the CAN_RF0R register set.
- The **FIFO 1 interrupt** can be generated by the following events:
 - Reception of a new message, FMP1 bits in the CAN_RF1R register are not '00'.
 - FIFO1 full condition, FULL1 bit in the CAN_RF1R register set.
 - FIFO1 overrun condition, FOVR1 bit in the CAN_RF1R register set.
- The **error and status change interrupt** can be generated by the following events:
 - Error condition, for more details on error conditions refer to the CAN Error Status register (CAN_ESR).
 - Wakeup condition, SOF monitored on the CAN Rx signal.
 - Entry into Sleep mode.

31.9 CAN registers

The peripheral registers have to be accessed by words (32 bits).

31.9.1 Register access protection

Erroneous access to certain configuration registers can cause the hardware to temporarily disturb the whole CAN network. Therefore the CAN_BTR register can be modified by software only while the CAN hardware is in initialization mode.

Although the transmission of incorrect data 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 416: Transmit mailbox states](#).

The filter values can be modified either deactivating the associated filter banks or by setting the FINIT bit. Moreover, the modification of the filter configuration (scale, mode and FIFO assignment) in CAN_FMxR, CAN_FSxR and CAN_FFAR registers can only be done when the filter initialization mode is set (FINIT=1) in the CAN_FMR register.

31.9.2 CAN control and status registers

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

CAN master control register (CAN_MCR)

Address offset: 0x00

Reset value: 0x0001 0002

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DBF
															rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESET	Res.	TTCM	ABOM	AWUM	NART	RFLM	TXFP	SLEEP	INRQ						
rs								rw	rw						

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

Bit 16 **DBF:** Debug freeze

- 0: CAN working during debug
- 1: CAN reception/transmission frozen during debug. Reception FIFOs can still be accessed/controlled normally.

Bit 15 **RESET:** bxCAN software master reset

- 0: Normal operation.
- 1: Force a master reset of the bxCAN -> Sleep mode activated after reset (FMP bits and CAN_MCR register are initialized to the reset values). This bit is automatically reset to 0.

Bits 14:8 Reserved, must be kept at reset value.

Bit 7 **TTCM:** Time triggered communication mode

- 0: Time Triggered Communication mode disabled.
- 1: Time Triggered Communication mode enabled

Note: For more information on Time Triggered Communication mode, refer to Section 31.7.2: Time triggered communication mode.

Bit 6 **ABOM:** Automatic bus-off management

- This bit controls the behavior of the CAN hardware on leaving the Bus-Off state.
- 0: The Bus-Off state is left on software request, once 128 occurrences of 11 recessive bits have been monitored and the software has first set and cleared the INRQ bit of the CAN_MCR register.
- 1: The Bus-Off state is left automatically by hardware once 128 occurrences of 11 recessive bits have been monitored.

For detailed information on the Bus-Off state refer to [Section 31.7.6: 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
Res.	Res.	Res.	Res.	Res.	Res.										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	RX	SAMP	RXM	TXM	Res.	Res.	SLAKI	WKUI	ERRI	SLAK	INAK	
				r	r	r	r			rc_w1	rc_w1	rc_w1	r	r	

Bits 31:12 Reserved, must be kept at reset value.

Bit 11 **RX**: CAN Rx signal

Monitors the actual value of the **CAN_RX** Pin.

Bit 10 **SAMP**: Last sample point

The value of RX on the last sample point (current received bit value).

Bit 9 **RXM**: Receive mode

The CAN hardware is currently receiver.

Bit 8 **TXM**: Transmit mode

The CAN hardware is currently transmitter.

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

Bit 4 **SLAKI**: Sleep acknowledge interrupt

When SLKIE=1, this bit is set by hardware to signal that the bxCAN has entered Sleep Mode. When set, this bit generates a status change interrupt if the SLKIE bit in the CAN_IER register is set.

This bit is cleared by software or by hardware, when SLAK is cleared.

Note: When SLKIE=0, no polling on SLAKI is possible. In this case the SLAK bit can be polled.

Bit 3 **WKUI**: 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. Refer to the AWUM bit of the CAN_MCR register description for detailed information for clearing SLEEP bit

Bit 0 **INAK**: Initialization acknowledge

This bit is set by hardware and indicates to the software that the CAN hardware is now in initialization mode. This bit acknowledges the initialization request from the software (set INRQ bit in CAN_MCR register).

This bit is cleared by hardware when the CAN hardware has left the initialization mode (to be synchronized on the CAN bus). To be synchronized the hardware has to monitor a sequence of 11 consecutive recessive bits on the CAN RX signal.

CAN transmit status register (CAN_TSR)

Address offset: 0x08

Reset value: 0x1C00 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LOW2	LOW1	LOW0	TME2	TME1	TME0	CODE[1:0]		ABRQ2	Res.	Res.	Res.	TERR2	ALST2	TXOK2	RQCP2
r	r	r	r	r	r	r	r	rs				rc_w1	rc_w1	rc_w1	rc_w1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABRQ1	Res.	Res.	Res.	TERR1	ALST1	TXOK1	RQCP1	ABRQ0	Res.	Res.	Res.	TERR0	ALST0	TXOK0	RQCP0
rs				rc_w1	rc_w1	rc_w1	rc_w1	rs				rc_w1	rc_w1	rc_w1	rc_w1

- Bit 31 **LOW2**: Lowest priority flag for mailbox 2
 This bit is set by hardware when more than one mailbox are pending for transmission and mailbox 2 has the lowest priority.
- Bit 30 **LOW1**: Lowest priority flag for mailbox 1
 This bit is set by hardware when more than one mailbox are pending for transmission and mailbox 1 has the lowest priority.
- Bit 29 **LOW0**: Lowest priority flag for mailbox 0
 This bit is set by hardware when more than one mailbox are pending for transmission and mailbox 0 has the lowest priority.
Note: The LOW[2:0] bits are set to 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, must be kept at reset value.
- Bit 19 **TERR2**: Transmission error of mailbox 2
 This bit is set when the previous TX failed due to an error.
- Bit 18 **ALST2**: Arbitration lost for mailbox 2
 This bit is set when the previous TX failed due to an arbitration lost.
- Bit 17 **TXOK2**: Transmission OK of mailbox 2
 The hardware updates this bit after each transmission attempt.
 0: The previous transmission failed
 1: The previous transmission was successful
 This bit is set by hardware when the transmission request on mailbox 2 has been completed successfully. Refer to [Figure 416](#).
- Bit 16 **RQCP2**: Request completed mailbox2
 Set by hardware when the last request (transmit or abort) has been performed.
 Cleared by software writing a “1” or by hardware on transmission request (TXRQ2 set in CAN_TMRD2R register).
 Clearing this bit clears all the status bits (TXOK2, ALST2 and TERR2) for Mailbox 2.
- Bit 15 **ABRQ1**: Abort request for mailbox 1
 Set by software to abort the transmission request for the corresponding mailbox.
 Cleared by hardware when the mailbox becomes empty.
 Setting this bit has no effect when the mailbox is not pending for transmission.
- Bits 14:12 Reserved, must be kept at reset value.

Bit 11 **TERR1**: Transmission error of mailbox1

This bit is set when the previous TX failed due to an error.

Bit 10 **ALST1**: Arbitration lost for mailbox1

This bit is set when the previous TX failed due to an arbitration lost.

Bit 9 **TXOK1**: Transmission OK of mailbox1

The hardware updates this bit after each transmission attempt.

0: The previous transmission failed

1: The previous transmission was successful

This bit is set by hardware when the transmission request on mailbox 1 has been completed successfully. Refer to [Figure 416](#)

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_TI1R register).

Clearing this bit clears all the status bits (TXOK1, ALST1 and TERR1) for Mailbox 1.

Bit 7 **ABRQ0**: Abort request for mailbox0

Set by software to abort the transmission request for the corresponding mailbox.

Cleared by hardware when the mailbox becomes empty.

Setting this bit has no effect when the mailbox is not pending for transmission.

Bits 6:4 Reserved, must be kept at reset value.

Bit 3 **TERR0**: Transmission error of mailbox0

This bit is set when the previous TX failed due to an error.

Bit 2 **ALST0**: Arbitration lost for mailbox0

This bit is set when the previous TX failed due to an arbitration lost.

Bit 1 **TXOK0**: Transmission OK of mailbox0

The hardware updates this bit after each transmission attempt.

0: The previous transmission failed

1: The previous transmission was successful

This bit is set by hardware when the transmission request on mailbox 1 has been completed successfully. Refer to [Figure 416](#)

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: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	RFOM0	FOVR0	FULL0	Res.	FMP0[1:0]										
										rs	rc_w1	rc_w1		r	r

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

Bit 5 RFOM0: Release FIFO 0 output mailbox

Set by software to release the output mailbox of the FIFO. The output mailbox can only be released when at least one message is pending in the FIFO. Setting this bit when the FIFO is empty has no effect. If at least two messages are pending in the FIFO, the software has to release the output mailbox to access the next message.

Cleared by hardware when the output mailbox has been released.

Bit 4 FOVR0: FIFO 0 overrun

This bit is set by hardware when a new message has been received and passed the filter while the FIFO was full.

This bit is cleared by software.

Bit 3 FULL0: FIFO 0 full

Set by hardware when three messages are stored in the FIFO.

This bit is cleared by software.

Bit 2 Reserved, must be kept at reset value.

Bits 1:0 FMP0[1:0]: FIFO 0 message pending

These bits indicate how many messages are pending in the receive FIFO.

FMP is increased each time the hardware stores a new message in to the FIFO. FMP is decreased each time the software releases the output mailbox by setting the RFOM0 bit.

CAN receive FIFO 1 register (CAN_RF1R)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	RFOM1	FOVR1	FULL1	Res.	FMP1[1:0]										
										rs	rc_w1	rc_w1		r	r

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

Bit 5 RFOM1: Release FIFO 1 output mailbox

Set by software to release the output mailbox of the FIFO. The output mailbox can only be released when at least one message is pending in the FIFO. Setting this bit when the FIFO is empty has no effect. If at least two messages are pending in the FIFO, the software has to release the output mailbox to access the next message.

Cleared by hardware when the output mailbox has been released.

Bit 4 FOVR1: FIFO 1 overrun

This bit is set by hardware when a new message has been received and passed the filter while the FIFO was full.

This bit is cleared by software.

Bit 3 **FULL1**: FIFO 1 full

Set by hardware when three messages are stored in the FIFO.
This bit is cleared by software.

Bit 2 Reserved, must be kept at reset value.

Bits 1:0 **FMP1[1:0]**: FIFO 1 message pending

These bits indicate how many messages are pending in the receive FIFO1.
FMP1 is increased each time the hardware stores a new message in to the FIFO1. FMP is decreased each time the software releases the output mailbox by setting the RFOM1 bit.

CAN interrupt enable register (CAN_IER)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SLKIE	WKUIE
														rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERRIE	Res.	Res.	Res.	LEC IE	BOF IE	EPV IE	EWG IE	Res.	FOV IE1	FF IE1	FMP IE1	FOV IE0	FF IE0	FMP IE0	TME IE
rw				rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

Bits 31:18 Reserved, must be kept at reset value.

Bit 17 **SLKIE**: Sleep interrupt enable

0: No interrupt when SLAKI bit is set.
1: Interrupt generated when SLAKI bit is set.

Bit 16 **WKUIE**: 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, must be kept at reset value.

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, must be kept at reset value.
- Bit 6 **FOVIE1**: FIFO overrun interrupt enable
 0: No interrupt when FOVR is set.
 1: Interrupt generation when FOVR is set.
- Bit 5 **FFIE1**: FIFO full interrupt enable
 0: No interrupt when FULL bit is set.
 1: Interrupt generated when FULL bit is set.
- Bit 4 **FMPIE1**: FIFO message pending interrupt enable
 0: No interrupt generated when state of FMP[1:0] bits are not 00b.
 1: Interrupt generated when state of FMP[1:0] bits are not 00b.
- Bit 3 **FOVIE0**: FIFO overrun interrupt enable
 0: No interrupt when FOVR bit is set.
 1: Interrupt generated when FOVR bit is set.
- Bit 2 **FFIE0**: FIFO full interrupt enable
 0: No interrupt when FULL bit is set.
 1: Interrupt generated when FULL bit is set.
- Bit 1 **FMPIE0**: FIFO message pending interrupt enable
 0: No interrupt generated when state of FMP[1:0] bits are not 00b.
 1: Interrupt generated when state of FMP[1:0] bits are not 00b.
- Bit 0 **TMEIE**: Transmit mailbox empty interrupt enable
 0: No interrupt when RQCPx bit is set.
 1: Interrupt generated when RQCPx bit is set.

Note: Refer to [Section 31.8: bxCAN interrupts](#).

CAN error status register (CAN_ESR)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
REC[7:0]								TEC[7:0]								
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	LEC[2:0]				Res.	BOFF	EPVF	EWGF
									rw	rw	rw		r	r	r	

Bits 31:24 **REC[7:0]**: Receive error counter

The implementing part of the fault confinement mechanism of the CAN protocol. In case of an error during reception, this counter is incremented by 1 or by 8 depending on the error condition as defined by the CAN standard. After every successful reception the counter is decremented by 1 or reset to 120 if its value was higher than 128. When the counter value exceeds 127, the CAN controller enters the error passive state.

Bits 23:16 **TEC[7:0]**: Least significant byte of the 9-bit transmit error counter

The implementing part of the fault confinement mechanism of the CAN protocol.

Bits 15:7 Reserved, must be kept at reset value.

Bits 6:4 **LEC[2:0]**: Last error code

This field is set by hardware and holds a code which indicates the error condition of the last error detected on the CAN bus. If a message has been transferred (reception or transmission) without error, this field will be cleared to '0'.

The LEC[2:0] bits can be set to value 0b111 by software. They are updated by hardware to indicate the current communication status.

000: No Error

001: Stuff Error

010: Form Error

011: Acknowledgment Error

100: Bit recessive Error

101: Bit dominant Error

110: CRC Error

111: Set by software

Bit 3 Reserved, must be kept at reset value.

Bit 2 **BOFF**: Bus-off flag

This bit is set by hardware when it enters the bus-off state. The bus-off state is entered on TEC overflow, greater than 255, refer to [Section 31.7.6 on page 1138](#).

Bit 1 **EPVF**: Error passive flag

This bit is set by hardware when the Error Passive limit has been reached (Receive Error Counter or Transmit Error Counter>127).

Bit 0 **EWGF**: Error warning flag

This bit is set by hardware when the warning limit has been reached (Receive Error Counter or Transmit Error Counter≥96).

CAN bit timing register (CAN_BTR)

Address offset: 0x1C

Reset value: 0x0123 0000

This register can only be accessed by the software when the CAN hardware is in initialization mode.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SILM	LBKM	Res.	Res.	Res.	Res.	SJW[1:0]		Res.	TS2[2:0]			TS1[3:0]			
rw	rw					rw	rw		rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	BRP[9:0]									
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Bit 31 **SILM**: Silent mode (debug)
 0: Normal operation
 1: Silent Mode
- Bit 30 **LBKM**: Loop back mode (debug)
 0: Loop Back Mode disabled
 1: Loop Back Mode enabled
- Bits 29:26 Reserved, must be kept at reset value.
- Bits 25:24 **SJW[1:0]**: Resynchronization jump width
 These bits define the maximum number of time quanta the CAN hardware is allowed to lengthen or shorten a bit to perform the resynchronization.
 $t_{RJW} = t_q \times (SJW[1:0] + 1)$
- Bit 23 Reserved, must be kept at reset value.
- Bits 22:20 **TS2[2:0]**: Time segment 2
 These bits define the number of time quanta in Time Segment 2.
 $t_{BS2} = t_q \times (TS2[2:0] + 1)$
- Bits 19:16 **TS1[3:0]**: Time segment 1
 These bits define the number of time quanta in Time Segment 1
 $t_{BS1} = t_q \times (TS1[3:0] + 1)$
 For more information on bit timing, refer to [Section 31.7.7: Bit timing on page 1138](#).
- Bits 15:10 Reserved, must be kept at reset value.
- Bits 9:0 **BRP[9:0]**: Baud rate prescaler
 These bits define the length of a time quanta.
 $t_q = (BRP[9:0]+1) \times t_{PCLK}$

31.9.3 CAN mailbox registers

This chapter describes the registers of the transmit and receive mailboxes. Refer to [Section 31.7.5: Message storage on page 1136](#) for detailed register mapping.

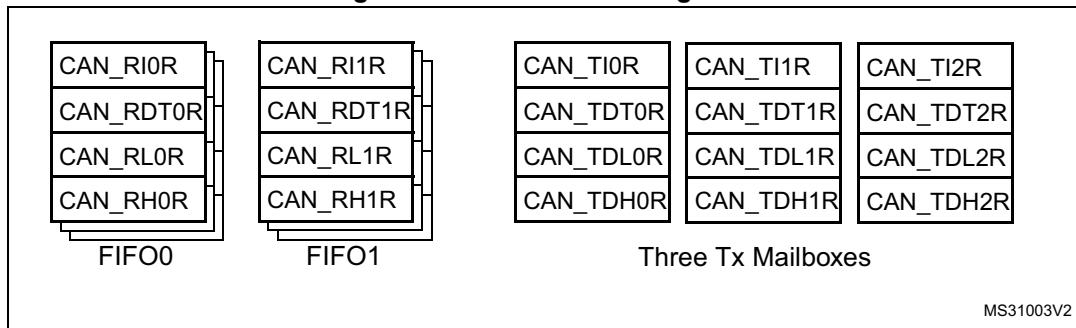
Transmit and receive mailboxes have the same registers except:

- 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.

Figure 425. CAN mailbox registers

**CAN TX mailbox identifier register (CAN_TIxR) (x = 0..2)**

Address offsets: 0x180, 0x190, 0x1A0

Reset value: 0xFFFF XXXX (except bit 0, TXRQ = 0)

All TX registers are write protected when the mailbox is pending transmission (TMEx reset).

This register also implements the TX request control (bit 0) - reset value 0.

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

Bits 31:21 **STID[10:0]/EXID[28:18]**: Standard identifier or extended identifier

The standard identifier or the MSBs of the extended identifier (depending on the IDE bit value).

Bit 20:3 **EXID[17:0]**: Extended identifier

The LSBs of the extended identifier.

Bit 2 **IDE**: Identifier extension

This bit defines the identifier type of message in the mailbox.

0: Standard identifier.

1: Extended identifier.

Bit 1 **RTR**: Remote transmission request

0: Data frame

1: Remote frame

Bit 0 **TXRQ**: Transmit mailbox request

Set by software to request the transmission for the corresponding mailbox.

Cleared by hardware when the mailbox becomes empty.

CAN mailbox data length control and time stamp register (CAN_TDTxR) (x = 0..2)

All bits of this register are write protected when the mailbox is not in empty state.

Address offsets: 0x184, 0x194, 0x1A4

Reset value: 0xFFFF XXXX

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

Bits 31:16 **TIME[15:0]**: Message time stamp

This field contains the 16-bit timer value captured at the SOF transmission.

Bits 15:9 Reserved, must be kept at reset value.

Bit 8 **TGT**: Transmit global time

This bit is active only when the hardware is in the Time Trigger Communication mode, 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 7 and TIME[15:8] in data byte 6, replacing the data written in CAN_TDHR[31:16] register (DATA6[7:0] and DATA7[7:0]). DLC must be programmed as 8 in order these two bytes to be sent over the CAN bus.

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

Bits 3:0 **DLC[3:0]**: Data length code

This field defines the number of data bytes a data frame contains or a remote frame request. A message can contain from 0 to 8 data bytes, depending on the value in the DLC field.

CAN mailbox data low register (CAN_TDLxR) (x = 0..2)

All bits of this register are write protected when the mailbox is not in empty state.

Address offsets: 0x188, 0x198, 0x1A8

Reset value: 0XXXXX XXXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA3[7:0]								DATA2[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA1[7:0]								DATA0[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **DATA3[7:0]**: Data byte 3

Data byte 3 of the message.

Bits 23:16 **DATA2[7:0]**: Data byte 2

Data byte 2 of the message.

Bits 15:8 **DATA1[7:0]**: Data byte 1

Data byte 1 of the message.

Bits 7:0 **DATA0[7:0]**: Data byte 0

Data byte 0 of the message.

A message can contain from 0 to 8 data bytes and starts with byte 0.

CAN mailbox data high register (CAN_TDhxR) (x = 0..2)

All bits of this register are write protected when the mailbox is not in empty state.

Address offsets: 0x18C, 0x19C, 0x1AC

Reset value: 0XXXXX XXXXX

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.

CAN receive FIFO mailbox identifier register (CAN_RIxR) (x = 0..1)

Address offsets: 0x1B0, 0x1C0

Reset value: 0xFFFF XXXX

All RX registers are write protected.

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

Bits 31:21 **STID[10:0]/EXID[28:18]**: Standard identifier or extended identifier

The standard identifier or the MSBs of the extended identifier (depending on the IDE bit value).

Bits 20:3 **EXID[17:0]**: Extended identifier

The LSBs of the extended identifier.

Bit 2 **IDE**: Identifier extension

This bit defines the identifier type of message in the mailbox.

0: Standard identifier.

1: Extended identifier.

Bit 1 **RTR**: Remote transmission request

0: Data frame

1: Remote frame

Bit 0 Reserved, must be kept at reset value.

**CAN receive FIFO mailbox data length control and time stamp register
(CAN_RDTxR) (x = 0..1)**

Address offsets: 0x1B4, 0x1C4
Reset value: 0XXXX XXXX

All RX registers are write protected.

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

Bits 31:16 **TIME[15:0]**: Message time stamp

This field contains the 16-bit timer value captured at the SOF detection.

Bits 15:8 **FMI[7:0]**: Filter match index

This register contains the index of the filter the message stored in the mailbox passed through. For more details on identifier filtering refer to [Section 31.7.4: Identifier filtering on page 1132 - Filter Match Index](#) paragraph.

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

Bits 3:0 **DLC[3:0]**: Data length code

This field defines the number of data bytes a data frame contains (0 to 8). It is 0 in the case of a remote frame request.

CAN receive FIFO mailbox data low register (CAN_RDLxR) (x = 0..1)

All bits of this register are write protected when the mailbox is not in empty state.

Address offsets: 0x1B8, 0x1C8

Reset value: 0XXXX XXXX

All RX registers are write protected.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA3[7:0]								DATA2[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA1[7:0]								DATA0[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:24 **DATA3[7:0]**: Data Byte 3

Data byte 3 of the message.

Bits 23:16 **DATA2[7:0]**: Data Byte 2

Data byte 2 of the message.

Bits 15:8 **DATA1[7:0]**: Data Byte 1

Data byte 1 of the message.

Bits 7:0 **DATA0[7:0]**: Data Byte 0

Data byte 0 of the message.

A message can contain from 0 to 8 data bytes and starts with byte 0.

CAN receive FIFO mailbox data high register (CAN_RDHxR) (x = 0..1)

Address offsets: 0x1BC, 0x1CC

Reset value: 0XXXX XXXX

All RX registers are write protected.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA7[7:0]								DATA6[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA5[7:0]								DATA4[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:24 **DATA7[7:0]**: Data Byte 7

Data byte 3 of the message.

Bits 23:16 **DATA6[7:0]**: Data Byte 6
Data byte 2 of the message.

Bits 15:8 **DATA5[7:0]**: Data Byte 5
Data byte 1 of the message.

Bits 7:0 **DATA4[7:0]**: Data Byte 4
Data byte 0 of the message.

31.9.4 CAN filter registers

CAN filter master register (CAN_FMR)

Address offset: 0x200

Reset value: 0x2A1C 0E01

All bits of this register are set and cleared by software.

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

Bits 31:1 Reserved, must be kept at reset value.

Bit 0 **FINIT**: Filter initialization mode

Initialization mode for filter banks

0: Active filters mode.

1: Initialization mode for the filters.

CAN filter mode register (CAN_FM1R)

Address offset: 0x204

Reset value: 0x0000 0000

This register can be written only when the filter initialization mode is set (FINIT=1) in the CAN_FMR register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	FBM13	FBM12	FBM11	FBM10	FBM9	FBM8	FBM7	FBM6	FBM5	FBM4	FBM3	FBM2	FBM1	FBM0
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Note: Refer to [Figure 418: Filter bank scale configuration - register organization on page 1134](#).

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

Bits 13:0 **FBMx**: Filter mode

Mode of the registers of Filter x.

0: Two 32-bit registers of filter bank x are in Identifier Mask mode.

1: Two 32-bit registers of filter bank x are in Identifier List mode.

CAN filter scale register (CAN_FSI1R)

Address offset: 0x20C

Reset value: 0x0000 0000

This register can be written only when the filter initialization mode is set (FINIT=1) in the CAN_FMR register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	Res	FSC13	FSC12	FSC11	FSC10	FSC9	FSC8	FSC7	FSC6	FSC5	FSC4	FSC3	FSC2	FSC1	FSC0
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 13:0 **FSCx**: Filter scale configuration

These bits define the scale configuration of Filters 13-0.

0: Dual 16-bit scale configuration

1: Single 32-bit scale configuration

Note: Refer to [Figure 418: Filter bank scale configuration - register organization on page 1134](#).

CAN filter FIFO assignment register (CAN_FFA1R)

Address offset: 0x214

Reset value: 0x0000 0000

This register can be written only when the filter initialization mode is set (FINIT=1) in the CAN_FMR register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	Res	FFA13	FFA12	FFA11	FFA10	FFA9	FFA8	FFA7	FFA6	FFA5	FFA4	FFA3	FFA2	FFA1	FFA0
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 13:0 **FFAx**: Filter FIFO assignment for filter x

The message passing through this filter 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: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	Res	FACT1 3	FACT1 2	FACT1 1	FACT1 0	FACT9	FACT8	FACT7	FACT6	FACT5	FACT4	FACT3	FACT2	FACT1	FACT0
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 13:0 **FACTx**: Filter active

The software sets this bit to activate Filter x. To modify the Filter x registers (CAN_FxR[0:7]), the FACTx bit must be cleared or the FINIT bit of the CAN_FMR register must be set.

0: Filter x is not active

1: Filter x is active

Filter bank i register x (CAN_FiRx) (i = 0..13, x = 1, 2)

Address offsets: 0x240 to 0x2AC

Reset value: 0XXXXX XXXXX

There are 14 filter banks, i= 0 to 13. Each filter bank i is composed of two 32-bit registers, CAN_FiR[2:1].

This register can only be modified when the FACTx bit of the CAN_FAxR register is cleared or when the FINIT bit of the CAN_FMR register is set.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FB31	FB30	FB29	FB28	FB27	FB26	FB25	FB24	FB23	FB22	FB21	FB20	FB19	FB18	FB17	FB16
rw															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FB15	FB14	FB13	FB12	FB11	FB10	FB9	FB8	FB7	FB6	FB5	FB4	FB3	FB2	FB1	FB0
rw															

In all configurations:

Bits 31:0 **FB[31:0]**: Filter bits

Identifier

Each bit of the register specifies the level of the corresponding bit of the expected identifier.

0: Dominant bit is expected

1: Recessive bit is expected

Mask

Each bit of the register specifies whether the bit of the associated identifier register must match with the corresponding bit of the expected identifier or not.

0: Do not care, the bit is not used for the comparison

1: Must match, the bit of the incoming identifier must have the same level has specified in the corresponding identifier register of the filter.

Note: Depending on the scale and mode configuration of the filter the function of each register can differ. For the filter mapping, functions description and mask registers association, refer to [Section 31.7.4: Identifier filtering on page 1132](#).

A Mask/Identifier register in **mask mode** has the same bit mapping as in **identifier list mode**.

For the register mapping/addresses of the filter banks refer to [Table 219 on page 1163](#).

31.9.5 bxCAN register map

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

Table 219. bxCAN register map and reset values

Offset	Register	Reset value	31
0x000	CAN_MCR	Res.	Res.
	Reset value	0	30
0x004	CAN_MSR	Res.	Res.
	Reset value	0	29
0x008	CAN_TSR	Low[2:0]	Res.
	Reset value	0	28
0x00C	CAN_RF0R	TME[2:0]	Res.
	Reset value	1	27
0x010	CAN_RF1R	CODE[1:0]	Res.
	Reset value	0	26
0x014	CAN_IER	ABRQ2	Res.
	Reset value	0	25
0x018	CAN_ESR	Res.	Res.
	Reset value	0	24
0x01C	CAN_BTR	Res.	Res.
	Reset value	0	23
0x020-0x17F	-	REC[7:0]	Res.
	Reset value	0	22
0x180	CAN_TI0R	TEC[7:0]	Res.
	Reset value	0	21
0x01C	CAN_BTR	TS2[2:0]	Res.
	Reset value	0	20
0x020-0x17F	-	TS1[3:0]	Res.
	Reset value	0	19
0x180	CAN_TI0R	WRUIE	Res.
	Reset value	0	18
0x01C	CAN_BTR	ERRIE	Res.
	Reset value	0	17
0x020-0x17F	-	SLKIE	Res.
	Reset value	0	16
0x180	CAN_TI0R	WRFIE	Res.
	Reset value	0	15
0x01C	CAN_BTR	ERR01	Res.
	Reset value	0	14
0x020-0x17F	-	ALST1	Res.
	Reset value	0	13
0x180	CAN_TI0R	SAMP	Res.
	Reset value	0	12
0x01C	CAN_BTR	TERR1	Res.
	Reset value	0	11
0x020-0x17F	-	RX	Res.
	Reset value	0	10
0x180	CAN_TI0R	TXM	Res.
	Reset value	0	9
0x01C	CAN_BTR	TXM0	Res.
	Reset value	0	8
0x020-0x17F	-	RCQP1	Res.
	Reset value	0	7
0x180	CAN_TI0R	ABOM	Res.
	Reset value	0	6
0x01C	CAN_BTR	RFOM0	Res.
	Reset value	0	5
0x020-0x17F	-	AWUM	Res.
	Reset value	0	4
0x180	CAN_TI0R	NART	Res.
	Reset value	0	3
0x01C	CAN_BTR	RFLM	Res.
	Reset value	0	2
0x020-0x17F	-	TXFP	Res.
	Reset value	0	1
0x180	CAN_TI0R	SLEEP	Res.
	Reset value	0	0
0x01C	CAN_BTR	INRQ	Res.
	Reset value	0	0

Table 219. bxCAN register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x184	CAN_TDT0R	TIME[15:0]															Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TGT	Res.	Res.	Res.	Res.	DLC[3:0]			
0x188	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	x	-	-	-	x	x	x	x	x
	CAN_TDL0R	DATA3[7:0]					DATA2[7:0]					DATA1[7:0]					DATA0[7:0]																
0x18C	CAN_TDHO R	DATA7[7:0]					DATA6[7:0]					DATA5[7:0]					DATA4[7:0]																
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		
0x190	CAN_TI1R	STID[10:0]/EXID[28:18]										EXID[17:0]															IDE	RTR	TXRQ				
	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	0		
0x194	CAN_TDT1R	TIME[15:0]															Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TGT	Res.	Res.	Res.	Res.	Res.	DLC[3:0]		
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	x	-	-	-	-	-	x	x	x
0x198	CAN_TDL1R	DATA3[7:0]					DATA2[7:0]					DATA1[7:0]					DATA0[7:0]																
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x			
0x19C	CAN_TDHO R	DATA7[7:0]					DATA6[7:0]					DATA5[7:0]					DATA4[7:0]																
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x			
0x1A0	CAN_TI2R	STID[10:0]/EXID[28:18]										EXID[17:0]															IDE	RTR	TXRQ				
	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	TIME[15:0]															Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TGT	Res.	Res.	Res.	Res.	Res.	DLC[3:0]		
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	x	-	-	-	-	-	x	x	x
0x1A8	CAN_TDL2R	DATA3[7:0]					DATA2[7:0]					DATA1[7:0]					DATA0[7:0]																
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x			
0x1AC	CAN_TDHO R	DATA7[7:0]					DATA6[7:0]					DATA5[7:0]					DATA4[7:0]																
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x			
0x1B0	CAN_RI0R	STID[10:0]/EXID[28:18]										EXID[17:0]															IDE	RTR	Res.				
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	-		

Table 219. 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
0x1B4	CAN_RDT0R	TIME[15:0]															FMI[7:0]				Res.	Res.	Res.	Res.	DLC[3:0]								
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	-	-	-	-	x	x	x			
0x1B8	CAN_RDL0R	DATA3[7:0]							DATA2[7:0]							DATA1[7:0]				DATA0[7:0]													
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		
0x1BC	CAN_RDH0R	DATA7[7:0]							DATA6[7:0]							DATA5[7:0]				DATA4[7:0]													
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		
0x1C0	CAN_RI1R	STID[10:0]/EXID[28:18]															EXID[17:0]																
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	-		
0x1C4	CAN_RDT1R	TIME[15:0]															FMI[7:0]				Res.	Res.	Res.	Res.	DLC[3:0]								
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		
0x1C8	CAN_RDL1R	DATA3[7:0]							DATA2[7:0]							DATA1[7:0]				DATA0[7:0]													
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		
0x1CC	CAN_RDH1R	DATA7[7:0]							DATA6[7:0]							DATA5[7:0]				DATA4[7:0]													
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		
0x1D0-0x1FF	-	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
0x200	CAN_FMR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x204	CAN_FM1R	FBM[13:0]																															
	Reset value																																
0x208	-	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	Reset value																																
0x20C	CAN_FS1R	FSC[13:0]																															
	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			
0x210	-	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			

Table 219. bxCAN register map and reset values (continued)

32 USB on-the-go full-speed/high-speed (OTG_FS/OTG_HS)

32.1 Introduction

Portions Copyright (c) Synopsys, Inc. All rights reserved. Used with permission.

This section presents the architecture and the programming model of the OTG_FS/OTG_HS controller.

The following acronyms are used throughout the section:

FS	Full-speed
LS	Low-speed
HS	High-speed
MAC	Media access controller
OTG	On-the-go
PFC	Packet FIFO controller
PHY	Physical layer
USB	Universal serial bus
UTMI	USB 2.0 Transceiver Macrocell interface (UTMI)
ULPI	UTMI+ Low Pin Interface
LPM	Link power management
BCD	Battery charging detector
HNP	Host negotiation protocol
SRP	Session request protocol

References are made to the following documents:

- USB On-The-Go Supplement, Revision 2.0
- Universal Serial Bus Revision 2.0 Specification
- USB 2.0 Link Power Management Addendum Engineering Change Notice to the USB 2.0 specification, July 16, 2007
- Errata for USB 2.0 ECN: Link Power Management (LPM) - 7/2007
- Battery Charging Specification, Revision 1.2

The USB OTG is a dual-role device (DRD) controller that supports both device and host functions and is fully compliant with the *On-The-Go Supplement to the USB 2.0 Specification*. It can also be configured as a host-only or device-only controller, fully compliant with the *USB 2.0 Specification*. OTG_HS supports the speeds defined in the [Table 220: OTG_HS speeds supported](#) below. OTG_FS supports the speeds defined in the [Table 221: OTG_FS speeds supported](#) below. The USB OTG supports both HNP and SRP. The only external device required is a charge pump for V_{BUS} in OTG mode.

Table 220. OTG_HS speeds supported

-	HS (480 Mb/s)	FS (12 Mb/s)	LS (1.5 Mb/s)
Host mode	X	X	X
Device mode	X	X	-

Table 221. OTG_FS speeds supported

-	HS (480 Mb/s)	FS (12 Mb/s)	LS (1.5 Mb/s)
Host mode	-	X	X
Device mode	-	X	-

32.2 OTG main features

The main features can be divided into three categories: general, host-mode and device-mode features.

32.2.1 General features

The OTG_FS/OTG_HS interface general features are the following:

- It is USB-IF certified to the Universal Serial Bus Specification Rev 2.0
- OTG_HS supports the following PHY interfaces:
 - An on-chip full-speed PHY
 - An ULPI interface for external high-speed PHY
 - A UTMI interface for internal HS PHY
- It includes full support (PHY) for the optional On-The-Go (OTG) protocol detailed in the On-The-Go Supplement Rev 2.0 specification
 - Integrated support for A-B device identification (ID line)
 - Integrated support for host Negotiation protocol (HNP) and session request protocol (SRP)
 - It allows host to turn V_{BUS} off to conserve battery power in OTG applications
 - It supports OTG monitoring of V_{BUS} levels with internal comparators
 - It supports dynamic host-peripheral switch of role
- It is software-configurable to operate as:
 - SRP capable USB FS/HS Peripheral (B-device)
 - SRP capable USB FS/HS/LS host (A-device)
 - USB On-The-Go Full-Speed Dual Role device
- It supports FS/HS SOF and LS Keep-alives with
 - SOF pulse PAD connectivity
 - SOF pulse internal connection to timer (TIMx)
 - Configurable framing period
 - Configurable end of frame interrupt
- OTG_HS embeds an internal DMA with shareholding support and software selectable AHB burst type in DMA mode.
- It includes power saving features such as system stop during USB suspend, switch-off of clock domains internal to the digital core, PHY and DFIFO power management.
- It features a dedicated RAM of 1.25[FS] / 4[HS] Kbytes with advanced FIFO control:
 - Configurable partitioning of RAM space into different FIFOs for flexible and efficient use of RAM
 - Each FIFO can hold multiple packets
 - Dynamic memory allocation
 - Configurable FIFO sizes that are not powers of 2 to allow the use of contiguous memory locations
- It guarantees max USB bandwidth for up to one frame (1 ms) without system intervention.
- It supports charging port detection as described in Battery Charging Specification Revision 1.2 on the FS PHY transceiver only.

32.2.2 Host-mode features

The OTG_FS/OTG_HS interface main features and requirements in host-mode are the following:

- External charge pump for V_{BUS} voltage generation.
- Up to 12[FS] / 16[HS] host channels (pipes): each channel is dynamically reconfigurable to allocate any type of USB transfer.
- Built-in hardware scheduler holding:
 - Up to 12[FS] / 16[HS] interrupt plus isochronous transfer requests in the periodic hardware queue
 - Up to 12[FS] / 16[HS] control plus bulk transfer requests in the non-periodic hardware queue
- Management of a shared Rx FIFO, a periodic Tx FIFO and a nonperiodic Tx FIFO for efficient usage of the USB data RAM.

32.2.3 Peripheral-mode features

The OTG_FS/OTG_HS interface main features in peripheral-mode are the following:

- 1 bidirectional control endpoint0
- 5[FS] / 8[HS] IN endpoints (EPs) configurable to support bulk, interrupt or isochronous transfers
- 5[FS] / 8[HS] OUT endpoints configurable to support bulk, interrupt or isochronous transfers
- Management of a shared Rx FIFO and a Tx-OUT FIFO for efficient usage of the USB data RAM
- Management of up to 6[FS] / 9[HS] dedicated Tx-IN FIFOs (one for each active IN EP) to put less load on the application
- Support for the soft disconnect feature.

32.3 OTG implementation

Table 222. OTG implementation⁽¹⁾

USB features	OTG_FS	OTG_HS for STM32F7x2xx and STM32F730xx ⁽²⁾	OTG_HS for STM32F7x3xx and STM32F730xx ⁽²⁾
Device bidirectional endpoints (including EP0)	6	9	9
Host mode channels	12	16	16
Size of dedicated SRAM	1.2 Kbytes	4 Kbytes	4 Kbytes
USB 2.0 link power management (LPM) support	X	X	X
OTG revision supported		2.0	
Attach detection protocol (ADP) support	-	-	-
Battery charging detection (BCD) support	X	-	-
ULPI available to primary IOs via muxing	-	X	-
Integrated PHY	FS	FS	HS

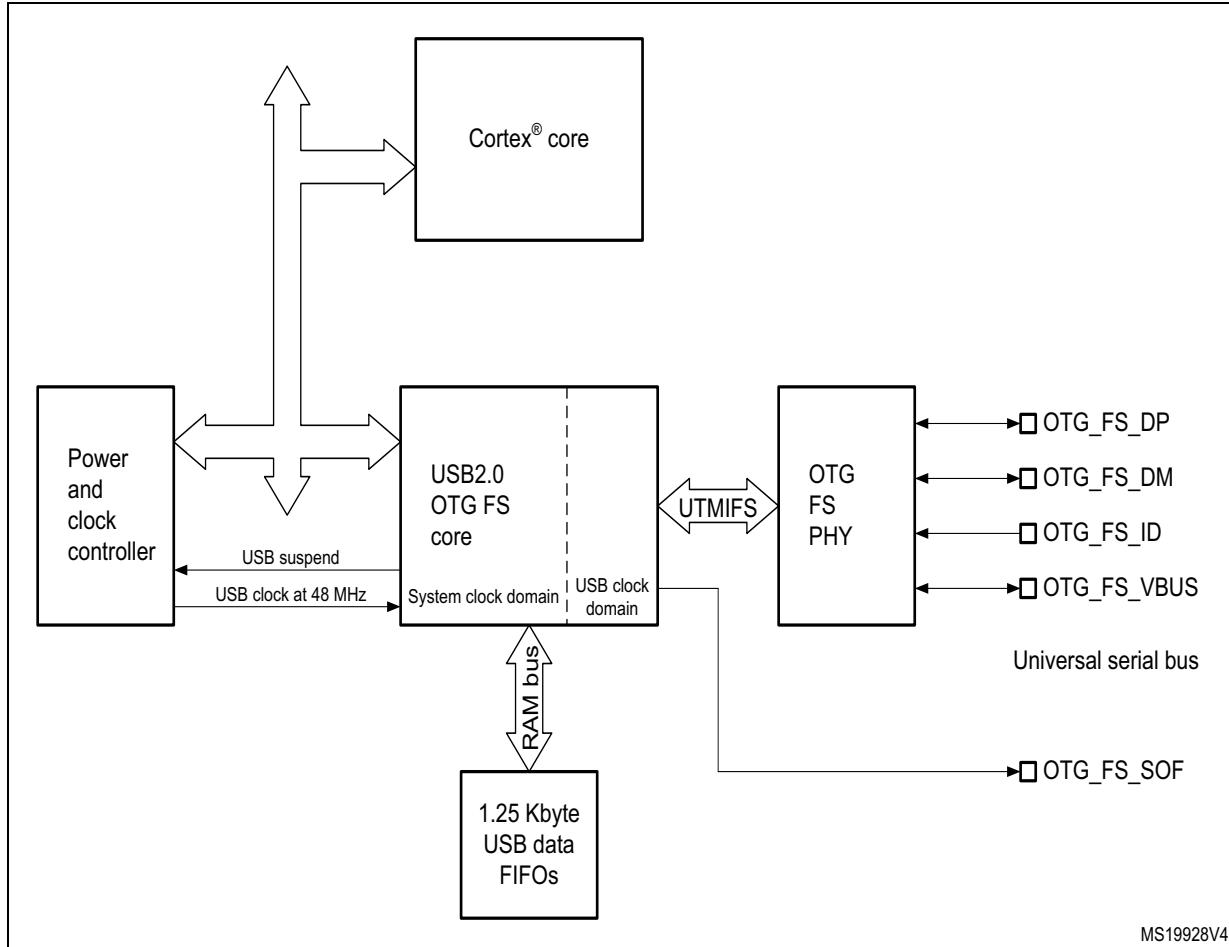
1. "X" = supported, “-” = not supported, “FS” = supported in FS mode, “HS” = supported in HS mode.

2. Refer to STM32F730xx datasheet for more details on packages supporting the OTG_HS ULPI versus the integrated PHY.

32.4 OTG functional description

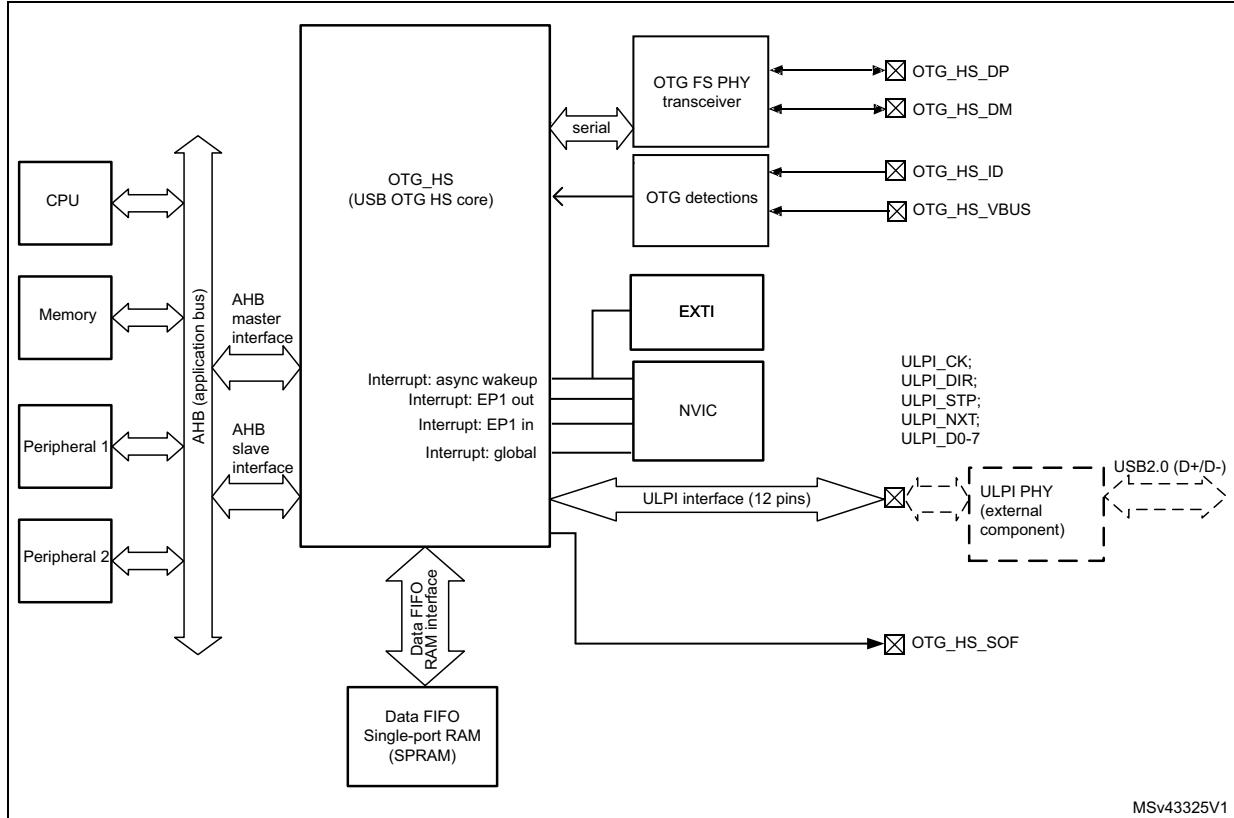
32.4.1 OTG block diagram

Figure 426. OTG full-speed block diagram



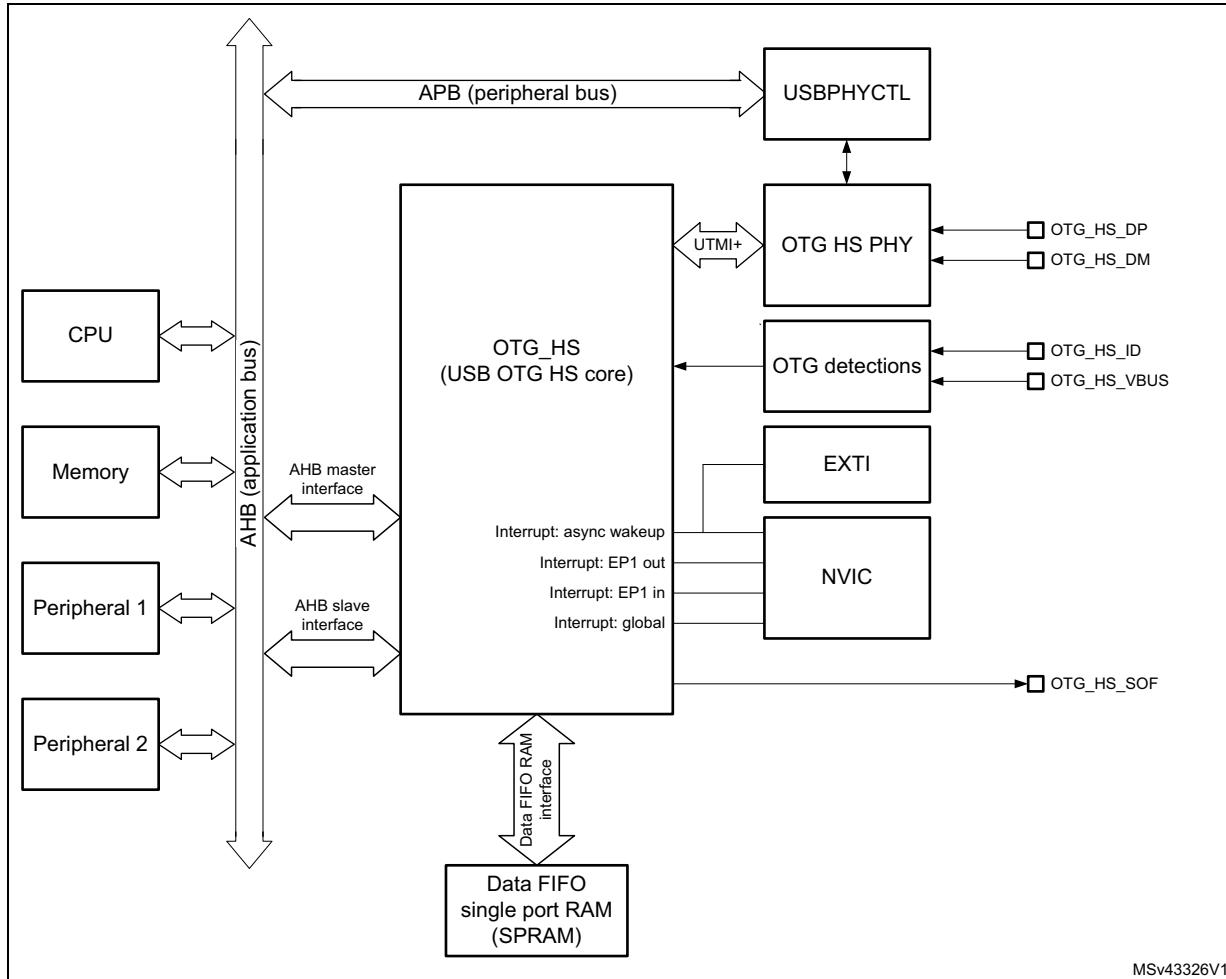
In STM32F7x2xx the configuration of the USB OTG_HS is shown here:

Figure 427. OTG high-speed block diagram for STM32F7x2xx



In STM32F7x3xx the configuration of the OTG_HS is shown here:

Figure 428. OTG high-speed block diagram for STM32F7x3xx



32.4.2 USB OTG pin and internal signals

Table 223. OTG_FS input/output pins

Signal name	Signal type	Description
OTG_FS_DP	Digital input/output	USB OTG D+ line
OTG_FS_DM	Digital input/output	USB OTG D- line
OTG_FS_ID	Digital input	USB OTG ID
OTG_FS_VBUS	Analog input	USB OTG VBUS
OTG_FS_SOF	Digital output	USB OTG Start Of Frame (visibility)

Table 224. OTG_HS input/output pins

Signal name	Signal type	Description
OTG_HS_DP	Digital input/output	USB OTG D+ line
OTG_HS_DM	Digital input/output	USB OTG D- line
OTG_HS_ID	Digital input	USB OTG ID
OTG_HS_VBUS	Analog input	USB OTG VBUS
OTG_HS_SOF	Digital output	USB OTG Start Of Frame (visibility)
OTG_HS_ULPI_CK	Digital input	USB OTG ULPI clock
OTG_HS_ULPI_DIR	Digital input	USB OTG ULPI data bus direction control
OTG_HS_ULPI_STP	Digital output	USB OTG ULPI data stream stop
OTG_HS_ULPI_NXT	Digital input	USB OTG ULPI next data stream request
OTG_HS_ULPI_D[0..7]	Digital input/output	USB OTG ULPI 8-bit bi-directional data bus

Table 225. OTG_FS/OTG_HS input/output signals

Signal name	Signal type	Description
usb_sof	Digital output	USB OTG start-of-frame event for on chip peripherals
usb_wkup	Digital output	USB OTG wakeup event output
usb_gbl_it	Digital output	USB OTG global interrupt
usb_ep1_in_it	Digital output	USB OTG endpoint 1 in interrupt
usb_ep1_out_it	Digital output	USB OTG endpoint 1 out interrupt

32.4.3 OTG core

The USB OTG receives the 48 MHz clock from the reset and clock controller (RCC). The USB clock is used for driving the 48 MHz domain at full-speed (12 Mbit/s) and must be enabled prior to configuring the OTG core.

The CPU reads and writes from/to the OTG core registers through the AHB peripheral bus. It is informed of USB events through the single USB OTG interrupt line described in [Section 32.13: OTG_FS/OTG_HS interrupts](#).

The CPU submits data over the USB by writing 32-bit words to dedicated OTG locations (push registers). The data are then automatically stored into Tx-data FIFOs configured within the USB data RAM. There is one Tx FIFO push register for each in-endpoint (peripheral mode) or out-channel (host mode).

The CPU receives the data from the USB by reading 32-bit words from dedicated OTG addresses (pop registers). The data are then automatically retrieved from a shared Rx FIFO configured within the 1.25[FS] / 4[HS]-Kbyte USB data RAM. There is one Rx FIFO pop register for each out-endpoint or in-channel.

The USB protocol layer is driven by the serial interface engine (SIE) and serialized over the USB by the transceiver module within the on-chip physical layer (PHY) or external HS PHY.

32.4.4 Full-speed OTG PHY^(a)

The embedded full-speed OTG PHY is controlled by the OTG FS core and conveys USB control & data signals through the full-speed subset of the UTMI+ Bus (UTMIFS). It provides the physical support to USB connectivity.

The full-speed OTG PHY includes the following components:

- FS/LS transceiver module used by both host and device. It directly drives transmission and reception on the single-ended USB lines.
- Integrated ID pull-up resistor used to sample the ID line for A/B device identification.
- DP/DM integrated pull-up and pull-down resistors controlled by the OTG_FS core depending on the current role of the device. As a peripheral, it enables the DP pull-up resistor to signal full-speed peripheral connections as soon as V_{BUS} is sensed to be at a valid level (B-session valid). In host mode, pull-down resistors are enabled on both DP/DM. Pull-up and pull-down resistors are dynamically switched when the role of the device is changed via the host negotiation protocol (HNP).
- Pull-up/pull-down resistor ECN circuit. The DP pull-up consists of two resistors controlled separately from the OTG_FS as per the resistor Engineering Change Notice applied to USB Rev2.0. The dynamic trimming of the DP pull-up strength allows for better noise rejection and Tx/Rx signal quality.
- V_{BUS} sensing comparators with hysteresis used to detect V_{BUS} valid, A-B session valid and session-end voltage thresholds. They are used to drive the session request protocol (SRP), detect valid startup and end-of-session conditions, and constantly monitor the V_{BUS} supply during USB operations.
- V_{BUS} pulsing method circuit used to charge/discharge V_{BUS} through resistors during the SRP (weak drive).

Caution: To guarantee a correct operation for the USB OTG FS peripheral, the AHB frequency should be higher than 14.2 MHz.

32.4.5 Embedded full speed OTG PHY^(b)

The full-speed OTG PHY includes the following components:

- FS/LS transceiver module used by both host and device. It directly drives transmission and reception on the single-ended USB lines.
- integrated ID pull-up resistor used to sample the ID line for A/B device identification.
- DP/DM integrated pull-up and pull-down resistors controlled by the OTG_HS core depending on the current role of the device. As a peripheral, it enables the DP pull-up resistor to signal full-speed peripheral connections as soon as V_{BUS} is sensed to be at a valid level (B-session valid). In host mode, pull-down resistors are enabled on both DP/DM. Pull-up and pull-down resistors are dynamically switched when the peripheral role is changed via the host negotiation protocol (HNP).
- Pull-up/pull-down resistor ECN circuit. The DP pull-up consists of 2 resistors controlled separately from the OTG_HS as per the resistor Engineering Change Notice applied to

a. The content of this section applies only to USB OTG FS.

b. The content of this section applies only to USB OTG HS.

USB Rev2.0. The dynamic trimming of the DP pull-up strength allows to achieve a better noise rejection and Tx/Rx signal quality.

- V_{BUS} sensing comparators with hysteresis used to detect V_{BUS} valid, A-B session valid and session-end voltage thresholds. They are used to drive the session request protocol (SRP), detect valid startup and end-of-session conditions, and constantly monitor the V_{BUS} supply during USB operations.

To guarantee a correct operation for the USB OTG_HS peripheral, the AHB frequency should be higher than 30 MHz.

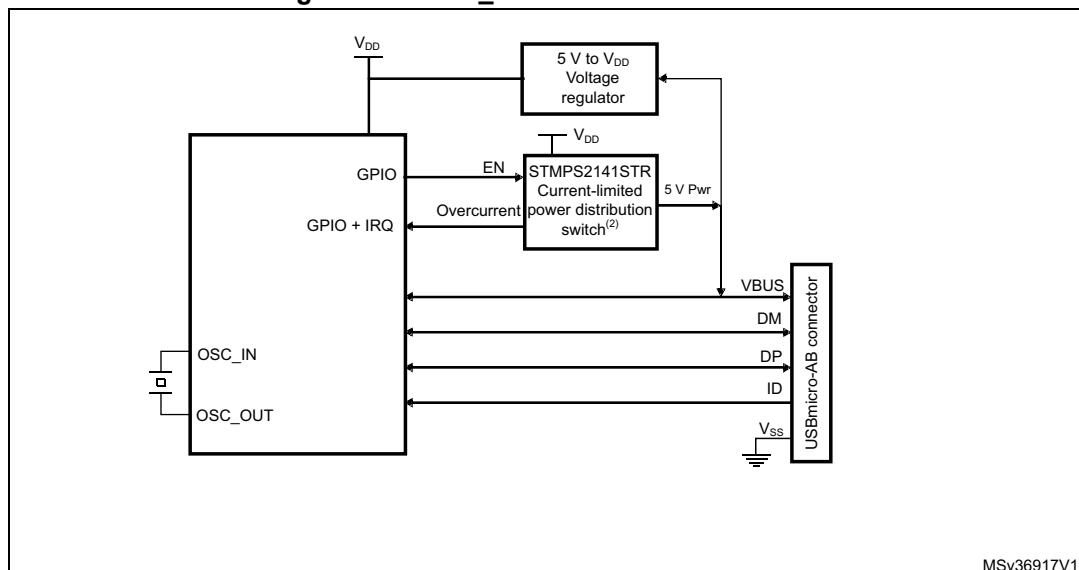
32.4.6 High-speed OTG PHY^(a)

The USB OTG core includes an internal UTMI interface which is connected to an internal HS PHY (see [Section 32.4.1: OTG block diagram](#)).

The USB OTG_HS core includes an ULPI interface to connect an external HS PHY.

32.5 OTG dual role device (DRD)

Figure 429. OTG_FS A-B device connection



1. External voltage regulator only needed when building a VBUS powered device.
2. STMPS2141STR needed only if the application has to support a VBUS powered device. A basic power switch can be used if 5 V are available on the application board.

a. The content of this section applies only to USB OTG HS.

32.5.1 ID line detection

The host or peripheral (the default) role is assumed depending on the ID input pin. The ID line status is determined on plugging in the USB cable, depending on whether a MicroA or MicroB plug is connected to the micro-AB receptacle.

- If the B-side of the USB cable is connected with a floating ID wire, the integrated pull-up resistor detects a high ID level and the default peripheral role is confirmed. In this configuration the OTG_FS/OTG_HS complies with the standard FSM described in section 4.2.4: ID pin of the On-the-Go specification Rev2.0, supplement to the USB2.0.
- If the A-side of the USB cable is connected with a grounded ID, the OTG_FS/OTG_HS issues an ID line status change interrupt (CIDSCHG bit in OTG_GINTSTS) for host software initialization, and automatically switches to the host role. In this configuration the OTG_FS/OTG_HS complies with the standard FSM described by section 4.2.4: ID pin of the On-the-Go specification Rev2.0, supplement to the USB2.0.

32.5.2 HNP dual role device

The HNP capable bit in the Global USB configuration register (HNPCAP bit in OTG_GUSBCFG) enables the OTG_FS/OTG_HS core to dynamically change its role from A-host to A-peripheral and vice-versa, or from B-Peripheral to B-host and vice-versa according to the host negotiation protocol (HNP). The current device status can be read by the combined values of the connector ID status bit in the Global OTG control and status register (CIDSTS bit in OTG_GOTGCTL) and the current mode of operation bit in the global interrupt and status register (CMOD bit in OTG_GINTSTS).

The HNP program model is described in detail in [Section 32.16: OTG_FS/OTG_HS programming model](#).

32.5.3 SRP dual role device

The SRP capable bit in the global USB configuration register (SRPCAP bit in OTG_GUSBCFG) enables the OTG_FS/OTG_HS core to switch off the generation of V_{BUS} for the A-device to save power. Note that the A-device is always in charge of driving V_{BUS} regardless of the host or peripheral role of the OTG_FS/OTG_HS.

The SRP A/B-device program model is described in detail in [Section 32.16: OTG_FS/OTG_HS programming model](#).

32.6 USB peripheral

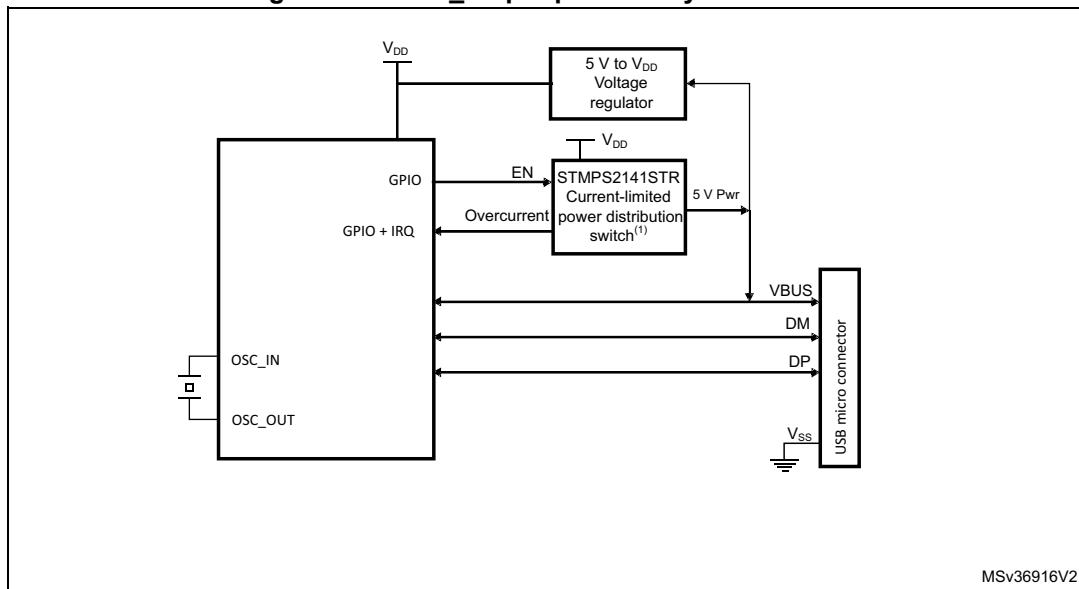
This section gives the functional description of the OTG_FS/OTG_HS in the USB peripheral mode. The OTG_FS/OTG_HS works as an USB peripheral in the following circumstances:

- OTG B-Peripheral
 - OTG B-device default state if B-side of USB cable is plugged in
- OTG A-Peripheral
 - OTG A-device state after the HNP switches the OTG_FS/OTG_HS to its peripheral role
- B-device
 - If the ID line is present, functional and connected to the B-side of the USB cable, and the HNP-capable bit in the Global USB Configuration register (HNPCAP bit in OTG_GUSBCFG) is cleared.
- Peripheral only (see [Figure 430: USB_FS peripheral-only connection](#))
 - The force device mode bit (FDMOD) in the [Section 32.15.4: OTG USB configuration register \(OTG_GUSBCFG\)](#) is set to 1, forcing the OTG_FS/OTG_HS core to work as an USB peripheral-only. In this case, the ID line is ignored even if it is present on the USB connector.

Note:

To build a bus-powered device implementation in case of the B-device or peripheral-only configuration, an external regulator has to be added, that generates the necessary power-supply from V_{BUS} .

Figure 430. USB_FS peripheral-only connection



MSv36916V2

1. Use a regulator to build a bus-powered device.

32.6.1 SRP-capable peripheral

The SRP capable bit in the Global USB configuration register (SRPCAP bit in OTG_GUSBCFG) enables the OTG_FS/OTG_HS to support the session request protocol (SRP). In this way, it allows the remote A-device to save power by switching off V_{BUS} while the USB session is suspended.

The SRP peripheral mode program model is described in detail in the [B-device session request protocol](#) section.

32.6.2 Peripheral states

Powered state

The V_{BUS} input detects the B-session valid voltage by which the USB peripheral is allowed to enter the powered state (see USB2.0 section 9.1). The OTG_FS/OTG_HS then automatically connects the DP pull-up resistor to signal full-speed device connection to the host and generates the session request interrupt (SRQINT bit in OTG_GINTSTS) to notify the powered state.

The V_{BUS} input also ensures that valid V_{BUS} levels are supplied by the host during USB operations. If a drop in V_{BUS} below B-session valid happens to be detected (for instance because of a power disturbance or if the host port has been switched off), the OTG_FS/OTG_HS automatically disconnects and the session end detected (SEDET bit in OTG_GOTGINT) interrupt is generated to notify that the OTG_FS/OTG_HS has exited the powered state.

In the powered state, the OTG_FS/OTG_HS expects to receive some reset signaling from the host. No other USB operation is possible. When a reset signaling is received the reset detected interrupt (USBRST in OTG_GINTSTS) is generated. When the reset signaling is complete, the enumeration done interrupt (ENUMDNE bit in OTG_GINTSTS) is generated and the OTG_FS/OTG_HS enters the Default state.

Soft disconnect

The powered state can be exited by software with the soft disconnect feature. The DP pull-up resistor is removed by setting the soft disconnect bit in the device control register (SDIS bit in OTG_DCTL), causing a device disconnect detection interrupt on the host side even though the USB cable was not really removed from the host port.

Default state

In the Default state the OTG_FS/OTG_HS expects to receive a SET_ADDRESS command from the host. No other USB operation is possible. When a valid SET_ADDRESS command is decoded on the USB, the application writes the corresponding number into the device address field in the device configuration register (DAD bit in OTG_DCFG). The OTG_FS/OTG_HS then enters the address state and is ready to answer host transactions at the configured USB address.

Suspended state

The OTG_FS/OTG_HS peripheral constantly monitors the USB activity. After counting 3 ms of USB idleness, the early suspend interrupt (ESUSP bit in OTG_GINTSTS) is issued, and confirmed 3 ms later, if appropriate, by the suspend interrupt (USBSUSP bit in OTG_GINTSTS). The device suspend bit is then automatically set in the device status register (SUSPSTS bit in OTG_DSTS) and the OTG_FS/OTG_HS enters the suspended state.

The suspended state may optionally be exited by the device itself. In this case the application sets the remote wakeup signaling bit in the device control register (RWUSIG bit in OTG_DCTL) and clears it after 1 to 15 ms.

When a resume signaling is detected from the host, the resume interrupt (WKUPINT bit in OTG_GINTSTS) is generated and the device suspend bit is automatically cleared.

32.6.3 Peripheral endpoints

The OTG_FS/OTG_HS core instantiates the following USB endpoints:

- Control endpoint 0:
 - Bidirectional and handles control messages only
 - Separate set of registers to handle in and out transactions
 - Proper control (OTG_DIEPCTL0/OTG_DOEPCTL0), transfer configuration (OTG_DIEPTSIZ0/OTG_DOEPTSIZ0), and status-interrupt (OTG_DIEPINT0/OTG_DOEPINT0) registers. The available set of bits inside the control and transfer size registers slightly differs from that of other endpoints
- 5[FS] / 8[HS] IN endpoints
 - Each of them can be configured to support the isochronous, bulk or interrupt transfer type
 - Each of them has proper control (OTG_DIEPCTLx), transfer configuration (OTG_DIEPTSIZx), and status-interrupt (OTG_DIEPINTx) registers
 - The device IN endpoints common interrupt mask register (OTG_DIEPMSK) is available to enable/disable a single kind of endpoint interrupt source on all of the IN endpoints (EP0 included)
 - Support for incomplete isochronous IN transfer interrupt (IISOIXFR bit in OTG_GINTSTS), asserted when there is at least one isochronous IN endpoint on which the transfer is not completed in the current frame. This interrupt is asserted along with the end of periodic frame interrupt (OTG_GINTSTS/EOPF).
- 5[FS] / 8[HS] OUT endpoints
 - Each of them can be configured to support the isochronous, bulk or interrupt transfer type
 - Each of them has a proper control (OTG_DOEPCTLx), transfer configuration (OTG_DOEPTSIZx) and status-interrupt (OTG_DOEPINTx) register
 - Device OUT endpoints common interrupt mask register (OTG_DOEPMSK) is available to enable/disable a single kind of endpoint interrupt source on all of the OUT endpoints (EP0 included)
 - Support for incomplete isochronous OUT transfer interrupt (INCOMPISOOUT bit in OTG_GINTSTS), asserted when there is at least one isochronous OUT endpoint on which the transfer is not completed in the current frame. This interrupt is asserted along with the end of periodic frame interrupt (OTG_GINTSTS/EOPF).

Endpoint control

- The following endpoint controls are available to the application through the device endpoint-x IN/OUT control register (OTG_DIEPCTLx/OTG_DOEPCTLx):
 - Endpoint enable/disable
 - Endpoint activate in current configuration
 - Program USB transfer type (isochronous, bulk, interrupt)
 - Program supported packet size
 - Program Tx FIFO number associated with the IN endpoint
 - Program the expected or transmitted data0/data1 PID (bulk/interrupt only)
 - Program the even/odd frame during which the transaction is received or transmitted (isochronous only)
 - Optionally program the NAK bit to always negative-acknowledge the host regardless of the FIFO status
 - Optionally program the STALL bit to always stall host tokens to that endpoint
 - Optionally program the SNOOP mode for OUT endpoint not to check the CRC field of received data

Endpoint transfer

The device endpoint-x transfer size registers (OTG_DIEPTSIZx/OTG_DOEPTSIZx) allow the application to program the transfer size parameters and read the transfer status. Programming must be done before setting the endpoint enable bit in the endpoint control register. Once the endpoint is enabled, these fields are read-only as the OTG_FS/OTG_HS core updates them with the current transfer status.

The following transfer parameters can be programmed:

- Transfer size in bytes
- Number of packets that constitute the overall transfer size

Endpoint status/interrupt

The device endpoint-x interrupt registers (OTG_DIEPINTx/OTG_DOEPINTx) indicate the status of an endpoint with respect to USB- and AHB-related events. The application must read these registers when the OUT endpoint interrupt bit or the IN endpoint interrupt bit in the core interrupt register (OEPINT bit in OTG_GINTSTS or IEPINT bit in OTG_GINTSTS, respectively) is set. Before the application can read these registers, it must first read the device all endpoints interrupt (OTG_DAINT) register to get the exact endpoint number for the device endpoint-x interrupt register. The application must clear the appropriate bit in this register to clear the corresponding bits in the OTG_DAINT and OTG_GINTSTS registers

The peripheral core provides the following status checks and interrupt generation:

- Transfer completed interrupt, indicating that data transfer was completed on both the application (AHB) and USB sides
- Setup stage has been done (control-out only)
- Associated transmit FIFO is half or completely empty (in endpoints)
- NAK acknowledge has been transmitted to the host (isochronous-in only)
- IN token received when Tx FIFO was empty (bulk-in/interrupt-in only)
- Out token received when endpoint was not yet enabled
- Babble error condition has been detected
- Endpoint disable by application is effective
- Endpoint NAK by application is effective (isochronous-in only)
- More than 3 back-to-back setup packets were received (control-out only)
- Timeout condition detected (control-in only)
- Isochronous out packet has been dropped, without generating an interrupt

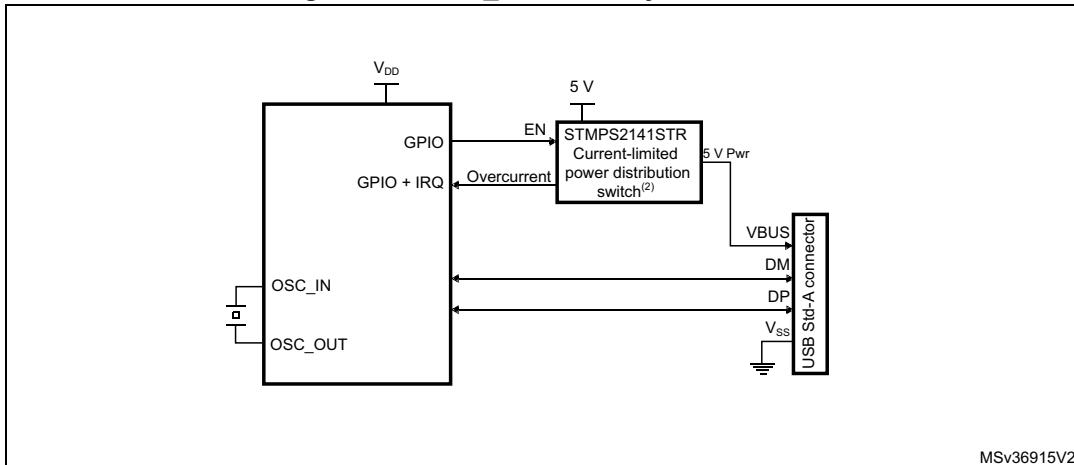
32.7 USB host

This section gives the functional description of the OTG_FS/OTG_HS in the USB host mode. The OTG_FS/OTG_HS works as a USB host in the following circumstances:

- OTG A-host
 - OTG A-device default state when the A-side of the USB cable is plugged in
- OTG B-host
 - OTG B-device after HNP switching to the host role
- A-device
 - If the ID line is present, functional and connected to the A-side of the USB cable, and the HNP-capable bit is cleared in the Global USB Configuration register (HNPCAP bit in OTG_GUSBCFG). Integrated pull-down resistors are automatically set on the DP/DM lines.
- Host only
 - The force host mode bit (FHMOD) in the [OTG USB configuration register \(OTG_GUSBCFG\)](#) forces the OTG_FS/OTG_HS core to work as a USB host-only. In this case, the ID line is ignored even if present on the USB connector. Integrated pull-down resistors are automatically set on the DP/DM lines.

Note: *On-chip 5 V V_{BUS} generation is not supported. For this reason, a charge pump or, if 5 V are available on the application board, a basic power switch must be added externally to drive the 5 V V_{BUS} line. The external charge pump can be driven by any GPIO output. This is required for the OTG A-host, A-device and host-only configurations.*

Figure 431. USB_FS host-only connection



1. V_{DD} range is between 2 V and 3.6 V.

32.7.1 SRP-capable host

SRP support is available through the SRP capable bit in the global USB configuration register (SRPCAP bit in OTG_GUSBCFG). With the SRP feature enabled, the host can save power by switching off the V_{BUS} power while the USB session is suspended.

The SRP host mode program model is described in detail in the [A-device session request protocol](#) section.

32.7.2 USB host states

Host port power

On-chip 5 V V_{BUS} generation is not supported. For this reason, a charge pump or, if 5 V are available on the application board, a basic power switch, must be added externally to drive the 5 V V_{BUS} line. The external charge pump can be driven by any GPIO output or via an I²C interface connected to an external PMIC (power management IC). When the application decides to power on V_{BUS} , it must also set the port power bit in the host port control and status register (PPWR bit in OTG_HPORT).

V_{BUS} valid

When HNP or SRP is enabled the V_{BUS} sensing pin should be connected to V_{BUS} . The V_{BUS} input ensures that valid V_{BUS} levels are supplied by the charge pump during USB operations. Any unforeseen V_{BUS} voltage drop below the V_{BUS} valid threshold (4.4 V) leads to an OTG interrupt triggered by the session end detected bit (SEDET bit in OTG_GOTGINT). The application is then required to remove the V_{BUS} power and clear the port power bit.

When HNP and SRP are both disabled, the V_{BUS} sensing pin does not need to be connected to V_{BUS} .

The charge pump overcurrent flag can also be used to prevent electrical damage. Connect the overcurrent flag output from the charge pump to any GPIO input and configure it to generate a port interrupt on the active level. The overcurrent ISR must promptly disable the V_{BUS} generation and clear the port power bit.

Host detection of a peripheral connection

If SRP or HNP are enabled, even if USB peripherals or B-devices can be attached at any time, the OTG_FS/OTG_HS will not detect any bus connection until V_{BUS} is no longer sensed at a valid level (5 V). When V_{BUS} is at a valid level and a remote B-device is attached, the OTG_FS/OTG_HS core issues a host port interrupt triggered by the device connected bit in the host port control and status register (PCDET bit in OTG_HPRT).

When HNP and SRP are both disabled, USB peripherals or B-device are detected as soon as they are connected. The OTG_FS/OTG_HS core issues a host port interrupt triggered by the device connected bit in the host port control and status (PCDET bit in OTG_HPRT).

Host detection of peripheral a disconnection

The peripheral disconnection event triggers the disconnect detected interrupt (DISCINT bit in OTG_GINTSTS).

Host enumeration

After detecting a peripheral connection the host must start the enumeration process by sending USB reset and configuration commands to the new peripheral.

Before starting to drive a USB reset, the application waits for the OTG interrupt triggered by the debounce done bit (DBCDNE bit in OTG_GOTGINT), which indicates that the bus is stable again after the electrical debounce caused by the attachment of a pull-up resistor on DP (FS) or DM (LS).

The application drives a USB reset signaling (single-ended zero) over the USB by keeping the port reset bit set in the host port control and status register (PRST bit in OTG_HPRT) for a minimum of 10 ms and a maximum of 20 ms. The application takes care of the timing count and then of clearing the port reset bit.

Once the USB reset sequence has completed, the host port interrupt is triggered by the port enable/disable change bit (PENCHNG bit in OTG_HPRT). This informs the application that the speed of the enumerated peripheral can be read from the port speed field in the host port control and status register (PSPD bit in OTG_HPRT) and that the host is starting to drive SOFs (FS) or Keep alives (LS). The host is now ready to complete the peripheral enumeration by sending peripheral configuration commands.

Host suspend

The application decides to suspend the USB activity by setting the port suspend bit in the host port control and status register (PSUSP bit in OTG_HPRT). The OTG_FS/OTG_HS core stops sending SOFs and enters the suspended state.

The suspended state can be optionally exited on the remote device's initiative (remote wakeup). In this case the remote wakeup interrupt (WKUPINT bit in OTG_GINTSTS) is generated upon detection of a remote wakeup signaling, the port resume bit in the host port control and status register (PRES bit in OTG_HPRT) self-sets, and resume signaling is automatically driven over the USB. The application must time the resume window and then clear the port resume bit to exit the suspended state and restart the SOF.

If the suspended state is exited on the host initiative, the application must set the port resume bit to start resume signaling on the host port, time the resume window and finally clear the port resume bit.

32.7.3 Host channels

The OTG_FS/OTG_HS core instantiates 12[FS] / 16[HS] host channels. Each host channel supports an USB host transfer (USB pipe). The host is not able to support more than 12[FS] / 16[HS] transfer requests at the same time. If more than 12[FS] / 16[HS] transfer requests are pending from the application, the host controller driver (HCD) must re-allocate channels when they become available from previous duty, that is, after receiving the transfer completed and channel halted interrupts.

Each host channel can be configured to support in/out and any type of periodic/nonperiodic transaction. Each host channel makes use of proper control (OTG_HCCHARx), transfer configuration (OTG_HCTSIZx) and status/interrupt (OTG_HCINTx) registers with associated mask (OTG_HCINTMSKx) registers.

Host channel control

- The following host channel controls are available to the application through the host channel-x characteristics register (OTG_HCCHARx):
 - Channel enable/disable
 - Program the HS/FS/LS speed of target USB peripheral
 - Program the address of target USB peripheral
 - Program the endpoint number of target USB peripheral
 - Program the transfer IN/OUT direction
 - Program the USB transfer type (control, bulk, interrupt, isochronous)
 - Program the maximum packet size (MPS)
 - Program the periodic transfer to be executed during odd/even frames

Host channel transfer

The host channel transfer size registers (OTG_HCTSIZx) allow the application to program the transfer size parameters, and read the transfer status. Programming must be done before setting the channel enable bit in the host channel characteristics register. Once the endpoint is enabled the packet count field is read-only as the OTG_FS/OTG_HS core updates it according to the current transfer status.

- The following transfer parameters can be programmed:
 - transfer size in bytes
 - number of packets making up the overall transfer size
 - initial data PID

Host channel status/interrupt

The host channel-x interrupt register (OTG_HCINTx) indicates the status of an endpoint with respect to USB- and AHB-related events. The application must read these register when the host channels interrupt bit in the core interrupt register (HCINT bit in OTG_GINTSTS) is set. Before the application can read these registers, it must first read the host all channels interrupt (OTG_HAINT) register to get the exact channel number for the host channel-x interrupt register. The application must clear the appropriate bit in this register to clear the corresponding bits in the OTG_HAINT and OTG_GINTSTS registers.

The mask bits for each interrupt source of each channel are also available in the OTG_HCINTMSKx register.

- The host core provides the following status checks and interrupt generation:
 - Transfer completed interrupt, indicating that the data transfer is complete on both the application (AHB) and USB sides
 - Channel has stopped due to transfer completed, USB transaction error or disable command from the application
 - Associated transmit FIFO is half or completely empty (IN endpoints)
 - ACK response received
 - NAK response received
 - STALL response received
 - USB transaction error due to CRC failure, timeout, bit stuff error, false EOP
 - Babble error
 - frame overrun
 - data toggle error

32.7.4 Host scheduler

The host core features a built-in hardware scheduler which is able to autonomously re-order and manage the USB transaction requests posted by the application. At the beginning of each frame the host executes the periodic (isochronous and interrupt) transactions first, followed by the nonperiodic (control and bulk) transactions to achieve the higher level of priority granted to the isochronous and interrupt transfer types by the USB specification.

The host processes the USB transactions through request queues (one for periodic and one for nonperiodic). Each request queue can hold up to 8 entries. Each entry represents a pending transaction request from the application, and holds the IN or OUT channel number along with other information to perform a transaction on the USB. The order in which the requests are written to the queue determines the sequence of the transactions on the USB interface.

At the beginning of each frame, the host processes the periodic request queue first, followed by the nonperiodic request queue. The host issues an incomplete periodic transfer interrupt (IPXFR bit in OTG_GINTSTS) if an isochronous or interrupt transaction scheduled for the current frame is still pending at the end of the current frame. The OTG_FS/OTG_HS core is fully responsible for the management of the periodic and nonperiodic request queues. The periodic transmit FIFO and queue status register (OTG_HPTXSTS) and nonperiodic transmit FIFO and queue status register (OTG_HNPTXSTS) are read-only registers which can be used by the application to read the status of each request queue. They contain:

- The number of free entries currently available in the periodic (nonperiodic) request queue (8 max)
- Free space currently available in the periodic (nonperiodic) Tx FIFO (out-transactions)
- IN/OUT token, host channel number and other status information.

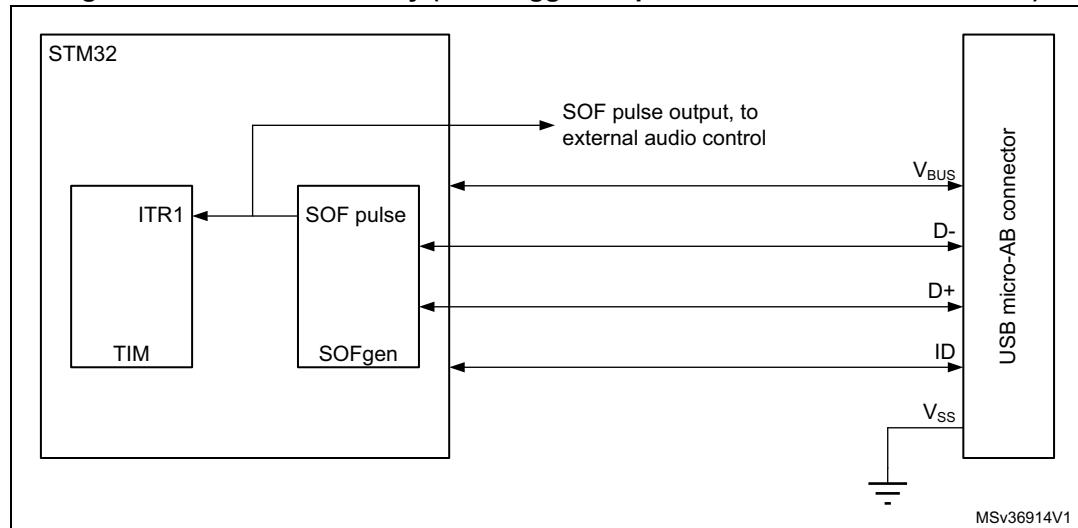
As request queues can hold a maximum of 8 entries each, the application can push to schedule host transactions in advance with respect to the moment they physically reach the SB for a maximum of 8 pending periodic transactions plus 8 pending non-periodic transactions.

To post a transaction request to the host scheduler (queue) the application must check that there is at least 1 entry available in the periodic (nonperiodic) request queue by reading the

PTXQSAV bits in the OTG_HNPTXSTS register or NPTQXSAV bits in the OTG_HNPTXSTS register.

32.8 SOF trigger

Figure 432. SOF connectivity (SOF trigger output to TIM and ITR1 connection)



The OTG_FS/OTG_HS core provides means to monitor, track and configure SOF framing in the host and peripheral, as well as an SOF pulse output connectivity feature.

Such utilities are especially useful for adaptive audio clock generation techniques, where the audio peripheral needs to synchronize to the isochronous stream provided by the PC, or the host needs to trim its framing rate according to the requirements of the audio peripheral.

32.8.1 Host SOFs

In host mode the number of PHY clocks occurring between the generation of two consecutive SOF (HS/FS) or Keep-alive (LS) tokens is programmable in the host frame interval register (HFIR), thus providing application control over the SOF framing period. An interrupt is generated at any start of frame (SOF bit in OTG_GINTSTS). The current frame number and the time remaining until the next SOF are tracked in the host frame number register (HFNUM).

A SOF pulse signal, is generated at any SOF starting token and with a width of 20 HCLK cycles. The SOF pulse is also internally connected to the input trigger of the timer, so that the input capture feature, the output compare feature and the timer can be triggered by the SOF pulse.

32.8.2 Peripheral SOFs

In device mode, the start of frame interrupt is generated each time an SOF token is received on the USB (SOF bit in OTG_GINTSTS). The corresponding frame number can be read from the device status register (FNSOF bit in OTG_DSTS). A SOF pulse signal with a width of 20 HCLK cycles is also generated. The SOF pulse signal is also internally connected to the TIM input trigger, so that the input capture feature, the output compare feature and the timer can be triggered by the SOF pulse.

The end of periodic frame interrupt (OTG_GINTSTS/EOPF) is used to notify the application when 80%, 85%, 90% or 95% of the time frame interval elapsed depending on the periodic frame interval field in the device configuration register (PFIVL bit in OTG_DCFG). This feature can be used to determine if all of the isochronous traffic for that frame is complete.

32.9 OTG low-power modes

Table 226 below defines the STM32 low power modes and their compatibility with the OTG.

Table 226. Compatibility of STM32 low power modes with the OTG

Mode	Description	USB compatibility
Run	MCU fully active	Required when USB not in suspend state.
Sleep	USB suspend exit causes the device to exit Sleep mode. Peripheral registers content is kept.	Available while USB is in suspend state.
Stop	USB suspend exit causes the device to exit Stop mode. Peripheral registers content is kept ⁽¹⁾ .	Available while USB is in suspend state.
Standby	Powered-down. The peripheral must be reinitialized after exiting Standby mode.	Not compatible with USB applications.

1. Within Stop mode there are different possible settings. Some restrictions may also exist, please refer to [Section 4: Power controller \(PWR\)](#) to understand which (if any) restrictions apply when using OTG.

The following bits and procedures reduce power consumption.

The power consumption of the OTG PHY is controlled by two or three bits in the general core configuration register, depending on OTG revision supported.

- PHY power down (OTG_GCCFG/PWRDWN)
It switches on/off the full-speed transceiver module of the PHY. It must be preliminarily set to allow any USB operation
- V_{BUS} detection enable (OTG_GCCFG/VBDEN)
It switches on/off the V_{BUS} sensing comparators associated with OTG operations

Power reduction techniques are available while in the USB suspended state, when the USB session is not yet valid or the device is disconnected.

- Stop PHY clock (STPPCLK bit in OTG_PCGCCTL)
When setting the stop PHY clock bit in the clock gating control register, most of the 48 MHz clock domain internal to the OTG full-speed core is switched off by clock gating. The dynamic power consumption due to the USB clock switching activity is cut even if the 48 MHz clock input is kept running by the application
Most of the transceiver is also disabled, and only the part in charge of detecting the asynchronous resume or remote wakeup event is kept alive.
- Gate HCLK (GATEHCLK bit in OTG_PCGCCTL)
When setting the Gate HCLK bit in the clock gating control register, most of the system clock domain internal to the OTG_FS/OTG_HS core is switched off by clock gating. Only the register read and write interface is kept alive. The dynamic power

consumption due to the USB clock switching activity is cut even if the system clock is kept running by the application for other purposes.

- USB system stop

When the OTG_FS/OTG_HS is in the USB suspended state, the application may decide to drastically reduce the overall power consumption by a complete shut down of all the clock sources in the system. USB System Stop is activated by first setting the Stop PHY clock bit and then configuring the system deep sleep mode in the power control system module (PWR).

The OTG_FS/OTG_HS core automatically reactivates both system and USB clocks by asynchronous detection of remote wakeup (as an host) or resume (as a device) signaling on the USB.

To save dynamic power, the USB data FIFO is clocked only when accessed by the OTG_FS/OTG_HS core.

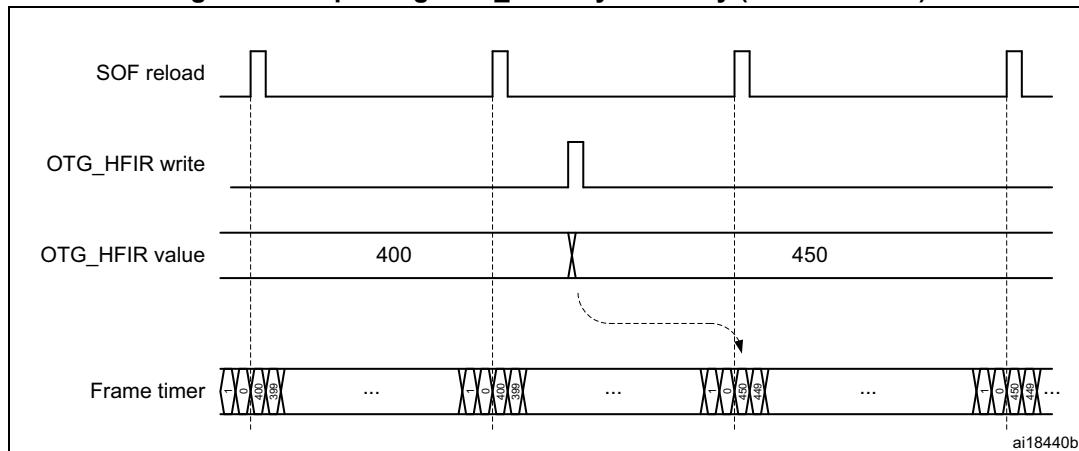
32.10 Dynamic update of the OTG_HFIR register

The USB core embeds a dynamic trimming capability of micro-SOF[HS] / SOF[FS] framing period in host mode allowing to synchronize an external device with the micro-SOF[HS] / SOF[FS] frames.

When the OTG_HFIR register is changed within a current micro-SOF[HS] / SOF[FS] frame, the SOF period correction is applied in the next frame as described in [Figure 433](#).

For a dynamic update, it is required to set RLDCTRL=0.

Figure 433. Updating OTG_HFIR dynamically (RLDCTRL = 0)



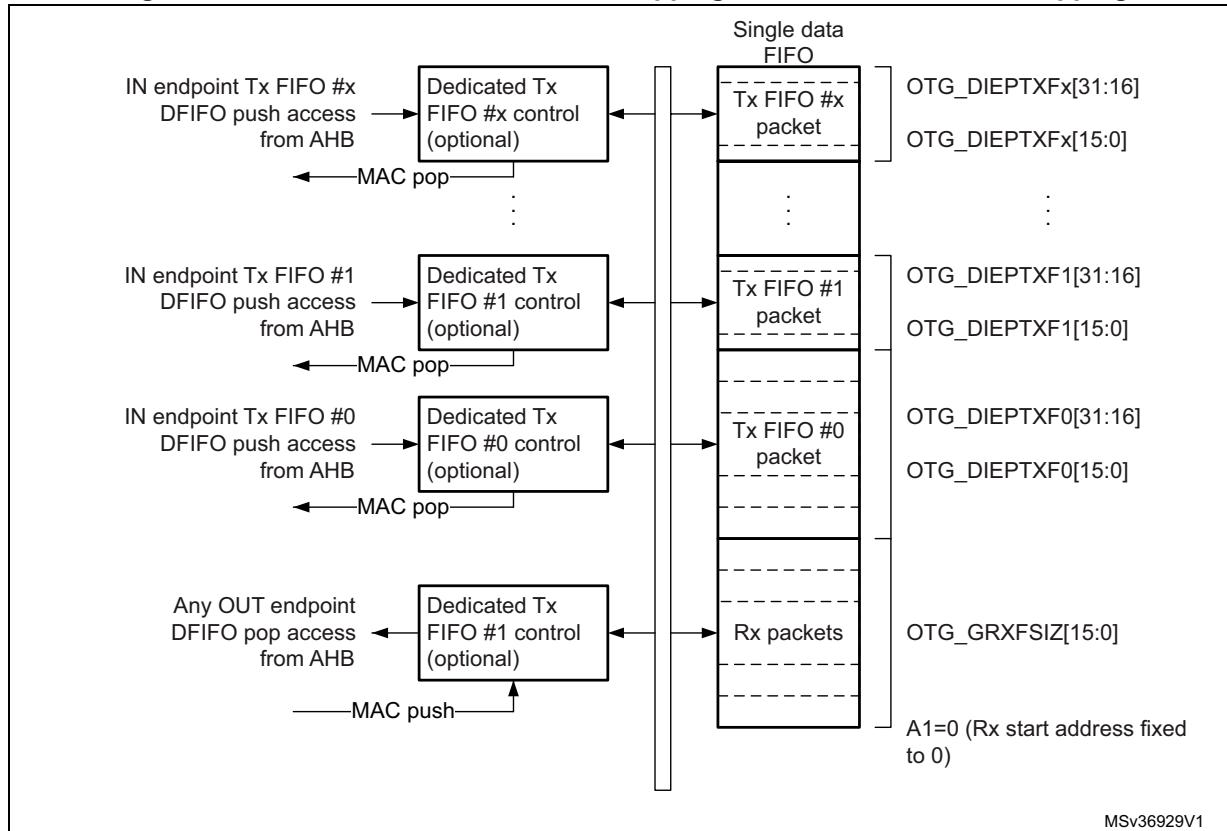
32.11 USB data FIFOs

The USB system features 1.25[FS] / 4[HS] Kbytes of dedicated RAM with a sophisticated FIFO control mechanism. The packet FIFO controller module in the OTG_FS/OTG_HS core organizes RAM space into Tx FIFOs into which the application pushes the data to be temporarily stored before the USB transmission, and into a single Rx FIFO where the data received from the USB are temporarily stored before retrieval (popped) by the application. The number of instructed FIFOs and how these are organized inside the RAM depends on

the device's role. In peripheral mode an additional Tx FIFO is instructed for each active IN endpoint. Any FIFO size is software configured to better meet the application requirements.

32.11.1 Peripheral FIFO architecture

Figure 434. Device-mode FIFO address mapping and AHB FIFO access mapping



Peripheral Rx FIFO

The OTG peripheral uses a single receive FIFO that receives the data directed to all OUT endpoints. Received packets are stacked back-to-back until free space is available in the Rx FIFO. The status of the received packet (which contains the OUT endpoint destination number, the byte count, the data PID and the validity of the received data) is also stored by the core on top of the data payload. When no more space is available, host transactions are NACKed and an interrupt is received on the addressed endpoint. The size of the receive FIFO is configured in the receive FIFO size register (OTG_GRXFSIZ).

The single receive FIFO architecture makes it more efficient for the USB peripheral to fill in the receive RAM buffer:

- All OUT endpoints share the same RAM buffer (shared FIFO)
- The OTG_FS/OTG_HS core can fill in the receive FIFO up to the limit for any host sequence of OUT tokens

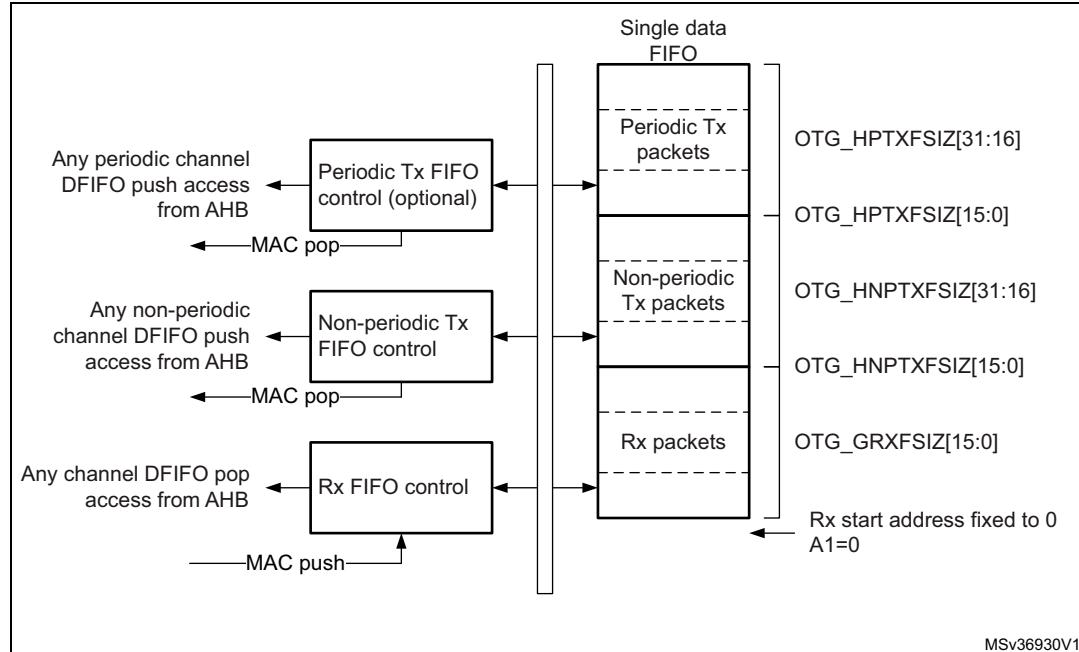
The application keeps receiving the Rx FIFO non-empty interrupt (RXFLVL bit in OTG_GINTSTS) as long as there is at least one packet available for download. It reads the packet information from the receive status read and pop register (OTG_GRXSTSP) and finally pops data off the receive FIFO by reading from the endpoint-related pop address.

Peripheral Tx FIFOs

The core has a dedicated FIFO for each IN endpoint. The application configures FIFO sizes by writing the endpoint 0 transmit FIFO size register (OTG_DIEPTXF0) for IN endpoint0 and the device IN endpoint transmit FIFOx registers (OTG_DIEPTXF x) for IN endpoint- x .

32.11.2 Host FIFO architecture

Figure 435. Host-mode FIFO address mapping and AHB FIFO access mapping



Host Rx FIFO

The host uses one receiver FIFO for all periodic and nonperiodic transactions. The FIFO is used as a receive buffer to hold the received data (payload of the received packet) from the USB until it is transferred to the system memory. Packets received from any remote IN endpoint are stacked back-to-back until free space is available. The status of each received packet with the host channel destination, byte count, data PID and validity of the received data are also stored into the FIFO. The size of the receive FIFO is configured in the receive FIFO size register (OTG_GRXFSIZ).

The single receive FIFO architecture makes it highly efficient for the USB host to fill in the receive data buffer:

- All IN configured host channels share the same RAM buffer (shared FIFO)
- The OTG_FS/OTG_HS core can fill in the receive FIFO up to the limit for any sequence of IN tokens driven by the host software

The application receives the Rx FIFO not-empty interrupt as long as there is at least one packet available for download. It reads the packet information from the receive status read and pop register and finally pops the data off the receive FIFO.

Host Tx FIFOs

The host uses one transmit FIFO for all non-periodic (control and bulk) OUT transactions and one transmit FIFO for all periodic (isochronous and interrupt) OUT transactions. FIFOs are used as transmit buffers to hold the data (payload of the transmit packet) to be transmitted over the USB. The size of the periodic (nonperiodic) Tx FIFO is configured in the host periodic (nonperiodic) transmit FIFO size OTG_HPTXFSIZ / OTG_HNPTXFSIZ register.

The two Tx FIFO implementation derives from the higher priority granted to the periodic type of traffic over the USB frame. At the beginning of each frame, the built-in host scheduler processes the periodic request queue first, followed by the nonperiodic request queue.

The two transmit FIFO architecture provides the USB host with separate optimization for periodic and nonperiodic transmit data buffer management:

- All host channels configured to support periodic (nonperiodic) transactions in the OUT direction share the same RAM buffer (shared FIFOs)
- The OTG_FS/OTG_HS core can fill in the periodic (nonperiodic) transmit FIFO up to the limit for any sequence of OUT tokens driven by the host software

The OTG_FS/OTG_HS core issues the periodic Tx FIFO empty interrupt (PTXFE bit in OTG_GINTSTS) as long as the periodic Tx FIFO is half or completely empty, depending on the value of the periodic Tx FIFO empty level bit in the AHB configuration register (PTXFELVL bit in OTG_GAHBCFG). The application can push the transmission data in advance as long as free space is available in both the periodic Tx FIFO and the periodic request queue. The host periodic transmit FIFO and queue status register (OTG_HPTXSTS) can be read to know how much space is available in both.

OTG_FS/OTG_HS core issues the non periodic Tx FIFO empty interrupt (NPTXFE bit in OTG_GINTSTS) as long as the nonperiodic Tx FIFO is half or completely empty depending on the non periodic Tx FIFO empty level bit in the AHB configuration register (TXFELVL bit in OTG_GAHBCFG). The application can push the transmission data as long as free space is available in both the nonperiodic Tx FIFO and nonperiodic request queue. The host nonperiodic transmit FIFO and queue status register (OTG_HNPTXSTS) can be read to know how much space is available in both.

32.11.3 FIFO RAM allocation

Device mode

Receive FIFO RAM allocation: the application should allocate RAM for SETUP packets:

- 10 locations must be reserved in the receive FIFO to receive SETUP packets on control endpoint. The core does not use these locations, which are reserved for SETUP packets, to write any other data.
- One location is to be allocated for Global OUT NAK.
- Status information is written to the FIFO along with each received packet. Therefore, a minimum space of (largest packet size / 4) + 1 must be allocated to receive packets. If multiple isochronous endpoints are enabled, then at least two (largest packet size / 4) + 1 spaces must be allocated to receive back-to-back packets. Typically, two (largest packet size / 4) + 1 spaces are recommended so that when the previous packet is being transferred to the CPU, the USB can receive the subsequent packet.
- Along with the last packet for each endpoint, transfer complete status information is also pushed to the FIFO. One location for each OUT endpoint is recommended.

Device RxFIFO =

$(5 * \text{number of control endpoints} + 8) + ((\text{largest USB packet used} / 4) + 1 \text{ for status information}) + (2 * \text{number of OUT endpoints}) + 1 \text{ for Global NAK}$

Example: The MPS is 1,024 bytes for a periodic USB packet and 512 bytes for a non-periodic USB packet. There are three OUT endpoints, three IN endpoints, one control endpoint, and three host channels.

Device RxFIFO = $(5 * 1 + 8) + ((1,024 / 4) + 1) + (2 * 4) + 1 = 279$

Transmit FIFO RAM allocation: the minimum RAM space required for each IN endpoint Transmit FIFO is the maximum packet size for that particular IN endpoint.

Note: More space allocated in the transmit IN endpoint FIFO results in better performance on the USB.

Host mode

Receive FIFO RAM allocation:

Status information is written to the FIFO along with each received packet. Therefore, a minimum space of $(\text{largest packet size} / 4) + 1$ must be allocated to receive packets. If multiple isochronous channels are enabled, then at least two $(\text{largest packet size} / 4) + 1$ spaces must be allocated to receive back-to-back packets. Typically, two $(\text{largest packet size} / 4) + 1$ spaces are recommended so that when the previous packet is being transferred to the CPU, the USB can receive the subsequent packet.

Along with the last packet in the host channel, transfer complete status information is also pushed to the FIFO. So one location must be allocated for this.

Host RxFIFO = $(\text{largest USB packet used} / 4) + 1 \text{ for status information} + 1 \text{ transfer complete}$

Example: Host RxFIFO = $((1,024 / 4) + 1) + 1 = 258$

Transmit FIFO RAM allocation:

The minimum amount of RAM required for the host Non-periodic Transmit FIFO is the largest maximum packet size among all supported non-periodic OUT channels.

Typically, two largest packet sizes worth of space is recommended, so that when the current packet is under transfer to the USB, the CPU can get the next packet.

Non-Periodic TxFIFO = largest non-periodic USB packet used / 4

Example: Non-Periodic TxFIFO = $(512 / 4) = 128$

The minimum amount of RAM required for host periodic Transmit FIFO is the largest maximum packet size out of all the supported periodic OUT channels. If there is at least one isochronous OUT endpoint, then the space must be at least two times the maximum packet size of that channel.

Host Periodic TxFIFO = largest periodic USB packet used / 4

Example: Host Periodic TxFIFO = $(1,024 / 4) = 256$

Note: More space allocated in the Transmit Non-periodic FIFO results in better performance on the USB.

32.12 OTG_FS system performance

Best USB and system performance is achieved owing to the large RAM buffers, the highly configurable FIFO sizes, the quick 32-bit FIFO access through AHB push/pop registers and, especially, the advanced FIFO control mechanism. Indeed, this mechanism allows the OTG_FS to fill in the available RAM space at best regardless of the current USB sequence. With these features:

- The application gains good margins to calibrate its intervention in order to optimize the CPU bandwidth usage:
 - It can accumulate large amounts of transmission data in advance compared to when they are effectively sent over the USB
 - It benefits of a large time margin to download data from the single receive FIFO
- The USB core is able to maintain its full operating rate, that is to provide maximum full-speed bandwidth with a great margin of autonomy versus application intervention:
 - It has a large reserve of transmission data at its disposal to autonomously manage the sending of data over the USB
 - It has a lot of empty space available in the receive buffer to autonomously fill it in with the data coming from the USB

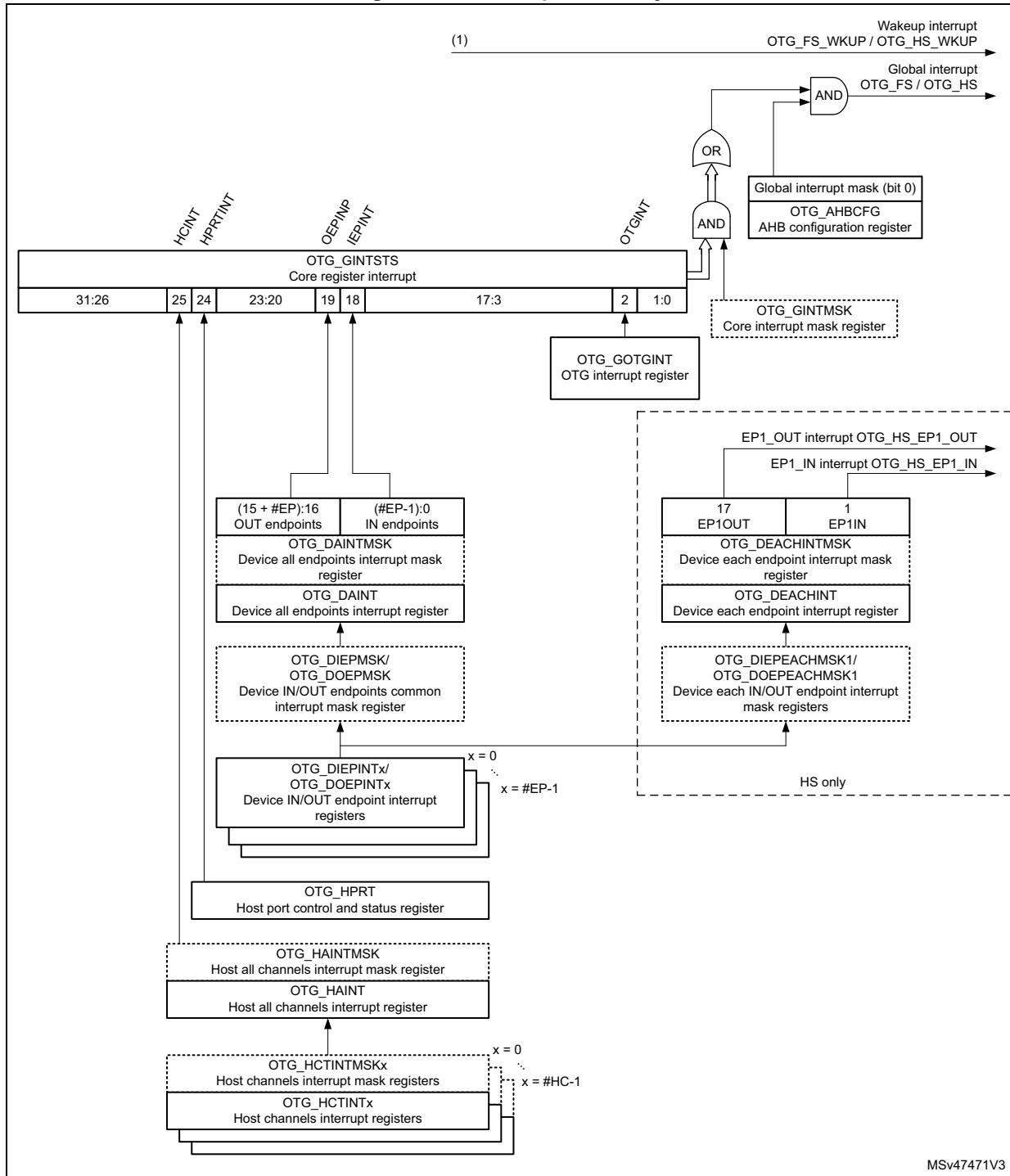
As the OTG_FS core is able to fill in the 1.25-Kbyte RAM buffer very efficiently, and as 1.25-Kbyte of transmit/receive data is more than enough to cover a full speed frame, the USB system is able to withstand the maximum full-speed data rate for up to one USB frame (1 ms) without any CPU intervention.

32.13 OTG_FS/OTG_HS interrupts

When the OTG_FS/OTG_HS controller is operating in one mode, either device or host, the application must not access registers from the other mode. If an illegal access occurs, a mode mismatch interrupt is generated and reflected in the core interrupt register (MMIS bit in the OTG_GINTSTS register). When the core switches from one mode to the other, the registers in the new mode of operation must be reprogrammed as they would be after a power-on reset.

Figure 436 shows the interrupt hierarchy.

Figure 436. Interrupt hierarchy



- OTG_FS_WKUP / OTG_HS_WKUP become active (high state) when resume condition occurs during L1 SLEEP or L2 SUSPEND states.

32.14 OTG_FS/OTG_HS control and status registers

By reading from and writing to the control and status registers (CSRs) through the AHB slave interface, the application controls the OTG_FS/OTG_HS controller. These registers are 32 bits wide, and the addresses are 32-bit block aligned. The OTG_FS/OTG_HS registers must be accessed by words (32 bits).

CSRs are classified as follows:

- Core global registers
- Host-mode registers
- Host global registers
- Host port CSRs
- Host channel-specific registers
- Device-mode registers
- Device global registers
- Device endpoint-specific registers
- Power and clock-gating registers
- Data FIFO (DFIFO) access registers

Only the core global, power and clock-gating, data FIFO access, and host port control and status registers can be accessed in both host and device modes. When the OTG_FS/OTG_HS controller is operating in one mode, either device or host, the application must not access registers from the other mode. If an illegal access occurs, a mode mismatch interrupt is generated and reflected in the core interrupt register (MMIS bit in the OTG_GINTSTS register). When the core switches from one mode to the other, the registers in the new mode of operation must be reprogrammed as they would be after a power-on reset.

32.14.1 CSR memory map

The host and device mode registers occupy different addresses. All registers are implemented in the AHB clock domain.

Global CSR map

These registers are available in both host and device modes.

Table 227. Core global control and status registers (CSRs)

Acronym	Address offset	Register name
OTG_GOTGCTL	0x000	Section 32.15.1: OTG control and status register (OTG_GOTGCTL)
OTG_GOTGINT	0x004	Section 32.15.2: OTG interrupt register (OTG_GOTGINT)
OTG_GAHBCFG	0x008	Section 32.15.3: OTG AHB configuration register (OTG_GAHBCFG)
OTG_GUSBCFG	0x00C	Section 32.15.4: OTG USB configuration register (OTG_GUSBCFG)
OTG_GRSTCTL	0x010	Section 32.15.5: OTG reset register (OTG_GRSTCTL)
OTG_GINTSTS	0x014	Section 32.15.6: OTG core interrupt register (OTG_GINTSTS)
OTG_GINTMSK	0x018	Section 32.15.7: OTG interrupt mask register (OTG_GINTMSK)

Table 227. Core global control and status registers (CSRs) (continued)

Acronym	Address offset	Register name
OTG_GRXSTSR	0x01C	Section 32.15.8: OTG receive status debug read/OTG status read and pop registers (OTG_GRXSTSR/OTG_GRXSTSP)
OTG_GRXSTSP	0x020	
OTG_GRXFSIZ	0x024	Section 32.15.9: OTG receive FIFO size register (OTG_GRXFSIZ)
OTG_HNPTXFSIZ/ OTG_DIEPTXF0 ⁽¹⁾	0x028	Section 32.15.10: OTG host non-periodic transmit FIFO size register (OTG_HNPTXFSIZ)/Endpoint 0 Transmit FIFO size (OTG_DIEPTXF0)
OTG_HNPTXSTS	0x02C	Section 32.15.11: OTG non-periodic transmit FIFO/queue status register (OTG_HNPTXSTS)
OTG_GCCFG	0x038	Section 32.15.12: OTG general core configuration register (OTG_GCCFG)
OTG_CID	0x03C	Section 32.15.13: OTG core ID register (OTG_CID)
OTG_GLPMCFG	0x54	Section 32.15.14: OTG core LPM configuration register (OTG_GLPMCFG)
OTG_HPTXFSIZ	0x100	Section 32.15.15: OTG host periodic transmit FIFO size register (OTG_HPTXFSIZ)
OTG_DIEPTXF _x	0x104 0x108 ... 0x114	Section 32.15.16: OTG device IN endpoint transmit FIFO size register (OTG_DIEPTXF_x) (x = 1..5[FS]/8[HS], where x is the FIFO number) for USB_OTG FS
OTG_DIEPTXF _x	0x104 0x108 ... 0x120	Section 32.15.16: OTG device IN endpoint transmit FIFO size register (OTG_DIEPTXF_x) (x = 1..5[FS]/8[HS], where x is the FIFO number) for USB_OTG HS

1. The general rule is to use OTG_HNPTXFSIZ for host mode and OTG_DIEPTXF0 for device mode.

Host-mode CSR map

These registers must be programmed every time the core changes to host mode.

Table 228. Host-mode control and status registers (CSRs)

Acronym	Offset address	Register name
OTG_HCFG	0x400	Section 32.15.18: OTG host configuration register (OTG_HCFG)
OTG_HFIR	0x404	Section 32.15.19: OTG host frame interval register (OTG_HFIR)
OTG_HFNUM	0x408	Section 32.15.20: OTG host frame number/frame time remaining register (OTG_HFNUM)
OTG_HPTXSTS	0x410	Section 32.15.21: OTG_Host periodic transmit FIFO/queue status register (OTG_HPTXSTS)
OTG_HAINT	0x414	Section 32.15.22: OTG host all channels interrupt register (OTG_HAINT)
OTG_HAINTMSK	0x418	Section 32.15.23: OTG host all channels interrupt mask register (OTG_HAINTMSK)

Table 228. Host-mode control and status registers (CSRs) (continued)

Acronym	Offset address	Register name
OTG_HPRT	0x440	Section 32.15.24: OTG host port control and status register (OTG_HPRT)
OTG_HCCHARx	0x500 0x520 ... 0x660	Section 32.15.25: OTG host channel x characteristics register (OTG_HCCHARx) (x = 0..15[HS] / 11[FS], where x = Channel number) for USB_OTG FS
OTG_HCCHARx	0x500 0x520 ... 0x6E0	Section 32.15.25: OTG host channel x characteristics register (OTG_HCCHARx) (x = 0..15[HS] / 11[FS], where x = Channel number) for USB_OTG HS
OTG_HCSPLTx	0x504 0x524 0x6E4	Section 32.15.26: OTG host channel x split control register (OTG_HCSPLTx) (x = 0..15, where x = Channel number)
OTG_HCINTx	0x508 0x528 0x668	Section 32.15.27: OTG host channel x interrupt register (OTG_HCINTx) (x = 0..15[HS] / 11[FS], where x = Channel number) for USB_OTG FS
OTG_HCINTx	0x508 0x528 0x6E8	Section 32.15.27: OTG host channel x interrupt register (OTG_HCINTx) (x = 0..15[HS] / 11[FS], where x = Channel number) for USB_OTG HS
OTG_HCINTMSKx	0x50C 0x52C 0x66C	Section 32.15.28: OTG host channel x interrupt mask register (OTG_HCINTMSKx) (x = 0..15[HS] / 11[FS], where x = Channel number) for USB_OTG FS
OTG_HCINTMSKx	0x50C 0x52C 0x6EC	Section 32.15.28: OTG host channel x interrupt mask register (OTG_HCINTMSKx) (x = 0..15[HS] / 11[FS], where x = Channel number) for USB_OTG HS
OTG_HCTSIZx	0x510 0x530 0x670	Section 32.15.29: OTG host channel x transfer size register (OTG_HCTSIZx) (x = 0..15[HS] / 11[FS], where x = Channel number) for USB_OTG FS
OTG_HCTSIZx	0x510 0x530 0x6F0	Section 32.15.29: OTG host channel x transfer size register (OTG_HCTSIZx) (x = 0..15[HS] / 11[FS], where x = Channel number) for USB_OTG HS
OTG_HCDMAX	0x514 0x534 0x6F4	Section 32.15.30: OTG host channel x DMA address register (OTG_HCDMAX) (x = 0..15, where x = Channel number)

Device-mode CSR map

These registers must be programmed every time the core changes to device mode.

Table 229. Device-mode control and status registers

Acronym	Offset address	Register name
OTG_DCFG	0x800	Section 32.15.32: OTG device configuration register (OTG_DCFG)
OTG_DCTL	0x804	Section 32.15.33: OTG device control register (OTG_DCTL)
OTG_DSTS	0x808	Section 32.15.34: OTG device status register (OTG_DSTS)
OTG_DIEPMSK	0x810	Section 32.15.35: OTG device IN endpoint common interrupt mask register (OTG_DIEPMSK)
OTG_DOEPMSK	0x814	Section 32.15.36: OTG device OUT endpoint common interrupt mask register (OTG_DOEPMSK)
OTG_DAINT	0x818	Section 32.15.37: OTG device all endpoints interrupt register (OTG_DAINT)
OTG_DAINTMSK	0x81C	Section 32.15.38: OTG all endpoints interrupt mask register (OTG_DAINTMSK)
OTG_DVBUSDIS	0x828	Section 32.15.39: OTG device VBUS discharge time register (OTG_DVBUSDIS)
OTG_DVBUSPULSE	0x82C	Section 32.15.40: OTG device VBUS pulsing time register (OTG_DVBUSPULSE)
OTG_DTHRCTL	0x830	Section 32.15.41: OTG device threshold control register (OTG_DTHRCTL)
OTG_DIEPEMPMSK	0x834	Section 32.15.42: OTG device IN endpoint FIFO empty interrupt mask register (OTG_DIEPEMPMSK)
OTG_DEACHINT	0x838	Section 32.15.43: OTG device each endpoint interrupt register (OTG_DEACHINT)
OTG_DEACHINTMSK	0x83C	Section 32.15.44: OTG device each endpoint interrupt mask register (OTG_DEACHINTMSK)
OTG_HS_DIEPEACHM SK1	0x844	Section 32.15.45: OTG device each IN endpoint-1 interrupt mask register (OTG_HS_DIEPEACHMSK1)
OTG_HS_DOEPEACHM SK1	0x884	Section 32.15.46: OTG device each OUT endpoint-1 interrupt mask register (OTG_HS_DOEPEACHMSK1)
OTG_DIEPCTL0	0x900	Section 32.15.47: OTG device control IN endpoint 0 control register (OTG_DIEPCTL0) for USB_OTG FS
OTG_DIEPCTLx	0x920 0x940 ... 0x9A0	Section 32.15.48: OTG device IN endpoint x control register (OTG_DIEPCTLx) (x = 1..5[FS] / 0..8[HS], where x = endpoint number) for USB_OTG FS

Table 229. Device-mode control and status registers (continued)

Acronym	Offset address	Register name
OTG_DIEPCTLx	0x900 0x920 ... 0xA00	Section 32.15.48: OTG device IN endpoint x control register (OTG_DIEPCTLx) (x = 1..5[FS] / 0..8[HS], where x = endpoint number) for USB_OTG HS
OTG_DIEPINTx	0x908 0x928 ... 0x988	Section 32.15.49: OTG device IN endpoint x interrupt register (OTG_DIEPINTx) (x = 0..5[FS] / 8[HS], where x = Endpoint number) for USB_OTG FS
OTG_DIEPINTx	0x908 0x928 ... 0x9E8	Section 32.15.49: OTG device IN endpoint x interrupt register (OTG_DIEPINTx) (x = 0..5[FS] / 8[HS], where x = Endpoint number) for USB_OTG HS
OTG_DIEPTSIZ0	0x910	Section 32.15.50: OTG device IN endpoint 0 transfer size register (OTG_DIEPTSIZ0)
OTG_DIEPDMAx	0x914 0x934 ... 0x9F4	Section 32.15.51: OTG device IN endpoint x DMA address register (OTG_DIEPDMAx) (x = 0..8, where x = endpoint number)
OTG_DTXFSTSx	0x918 0x938 ... 0x998	Section 32.15.52: OTG device IN endpoint transmit FIFO status register (OTG_DTXFSTSx) (x = 0..5[FS] / 8[HS], where x = endpoint number) for USB_OTG FS
OTG_DTXFSTSx	0x918 0x938 ... 0x9F8	Section 32.15.52: OTG device IN endpoint transmit FIFO status register (OTG_DTXFSTSx) (x = 0..5[FS] / 8[HS], where x = endpoint number) for USB_OTG HS
OTG_DIEPTSIZx	0x930 0x950 ... 0x9B0	Section 32.15.53: OTG device IN endpoint x transfer size register (OTG_DIEPTSIZx) (x = 1..5[FS] / 8[HS], where x = endpoint number) for USB_OTG FS
OTG_DIEPTSIZx	0x930 0x950 ... 0x9F0	Section 32.15.53: OTG device IN endpoint x transfer size register (OTG_DIEPTSIZx) (x = 1..5[FS] / 8[HS], where x = endpoint number) for USB_OTG HS
OTG_DOEPCTL0	0xB00	Section 32.15.54: OTG device control OUT endpoint 0 control register (OTG_DOEPCTL0)
OTG_DOEPINTx	0xB08 0xB28 ... 0xBA8	Section 32.15.55: OTG device OUT endpoint x interrupt register (OTG_DOEPINTx) (x = 0..5[FS] / 8[HS], where x = Endpoint number) for USB_OTG FS

Table 229. Device-mode control and status registers (continued)

Acronym	Offset address	Register name
OTG_DOEPINTx	0xB08 0XB28 ... 0xC08	<i>Section 32.15.55: OTG device OUT endpoint x interrupt register (OTG_DOEPINTx) (x = 0..5[FS]/8[HS], where x = Endpoint number) for USB_OTG HS</i>
OTG_DOEPTSIZ0	0xB10	<i>Section 32.15.56: OTG device OUT endpoint 0 transfer size register (OTG_DOEPTSIZ0)</i>
OTG_DOEPDMAx	0xB14 0xB34 ... 0xC14	<i>Section 32.15.57: OTG device OUT endpoint x DMA address register (OTG_DOEPDMAx) (x = 0..8, where x = endpoint number)</i>
OTG_DOEPCTLx	0xB20 0xB40 ... 0xBA0	<i>Section 32.15.58: OTG device OUT endpoint x control register (OTG_DOEPCTLx) (x = 1..5[FS]/8[HS], where x = endpoint number) for USB_OTG FS</i>
OTG_DOEPCTLx	0xB20 0xB40 ... 0xC00	<i>Section 32.15.58: OTG device OUT endpoint x control register (OTG_DOEPCTLx) (x = 1..5[FS]/8[HS], where x = endpoint number) for USB_OTG HS</i>
OTG_DOEPTSIZx	0xB30 0xB50 ... 0xBB0	<i>Section 32.15.59: OTG device OUT endpoint x transfer size register (OTG_DOEPTSIZx) (x = 1..5[FS]/8[HS], where x = Endpoint number) for USB_OTG FS</i>
OTG_DOEPTSIZx	0xB30 0xB50 ... 0xBF0	<i>Section 32.15.59: OTG device OUT endpoint x transfer size register (OTG_DOEPTSIZx) (x = 1..5[FS]/8[HS], where x = Endpoint number) for USB_OTG HS</i>

Data FIFO (DFIFO) access register map

These registers, available in both host and device modes, are used to read or write the FIFO space for a specific endpoint or a channel, in a given direction. If a host channel is of type IN, the FIFO can only be read on the channel. Similarly, if a host channel is of type OUT, the FIFO can only be written on the channel.

Table 230. Data FIFO (DFIFO) access register map

FIFO access register section	Offset address	Access
Device IN endpoint 0/Host OUT Channel 0: DFIFO write access Device OUT endpoint 0/Host IN Channel 0: DFIFO read access	0x1000–0x1FFC	w r
Device IN endpoint 1/Host OUT Channel 1: DFIFO write access Device OUT endpoint 1/Host IN Channel 1: DFIFO read access	0x2000–0x2FFC	w r

Table 230. Data FIFO (DFIFO) access register map (continued)

FIFO access register section	Offset address	Access
...
Device IN endpoint x ⁽¹⁾ /Host OUT Channel x ⁽¹⁾ : DFIFO write access Device OUT endpoint x ⁽¹⁾ /Host IN Channel x ⁽¹⁾ : DFIFO read access	0xX000–0xXFFC	w r

1. Where x is 5[FS] / 8[HS]in device mode and 11[FS] / 15[HS]in host mode.

Power and clock gating CSR map

There is a single register for power and clock gating. It is available in both host and device modes.

Table 231. Power and clock gating control and status registers

Acronym	Offset address	Register name
OTG_PCGCCTL	0xE00–0xE04	Section 32.15.60: OTG power and clock gating control register (OTG_PCGCCTL)

32.15 OTG_FS/OTG_HS registers

These registers are available in both host and device modes, and do not need to be reprogrammed when switching between these modes.

Bit values in the register descriptions are expressed in binary unless otherwise specified.

32.15.1 OTG control and status register (OTG_GOTGCTL)

Address offset: 0x000

Reset value: 0x0001 0000

The OTG_GOTGCTL register controls the behavior and reflects the status of the OTG function of the core.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CUR MOD	OTG VER	BSVLD	ASVLD	DBCT	CID STS
										r	rw	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	EHEN	DHNP EN	HSHNP EN	HNP RQ	HNG SCS	BVALO VAL	BVALO EN	AVALO VAL	AVALO EN	VBVAL OVAL	VBVAL OEN	SRQ	SRQ SCS
			rw	rw	rw	rw	r	rw	rw	rw	rw	rw	rw	rw	r

Bits 31:22 Reserved, must be kept at reset value.

Bit 21 **CURMOD:** Current mode of operation

Indicates the current mode (host or device).

0: Device mode

1: Host mode

Bit 20 **OTGVER:** OTG version

Selects the OTG revision.

0:OTG Version 1.3. OTG1.3 is obsolete for new product development.

1:OTG Version 2.0. In this version the core supports only data line pulsing for SRP.

Bit 19 **BSVLD:** B-session valid

Indicates the device mode transceiver status.

0: B-session is not valid.

1: B-session is valid.

In OTG mode, the user can use this bit to determine if the device is connected or disconnected.

Note: Only accessible in device mode.

Bit 18 **ASVLD:** A-session valid

Indicates the host mode transceiver status.

0: A-session is not valid

1: A-session is valid

Note: Only accessible in host mode.

Bit 17 **DBCT:** Long/short debounce time

Indicates the debounce time of a detected connection.

0: Long debounce time, used for physical connections (100 ms + 2.5 µs)

1: Short debounce time, used for soft connections (2.5 µs)

Note: Only accessible in host mode.

Bit 16 **CIDSTS:** Connector ID status

Indicates the connector ID status on a connect event.

0: The OTG_FS/OTG_HS controller is in A-device mode

1: The OTG_FS/OTG_HS controller is in B-device mode

Note: Accessible in both device and host modes.

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

Bit 12 **EHEN:** Embedded host enable

It is used to select between OTG A device state machine and embedded host state machine.

0: OTG A device state machine is selected

1: Embedded host state machine is selected

Bit 11 **DHNPN:** Device HNP enabled

The application sets this bit when it successfully receives a SetFeature.SethNPEnable command from the connected USB host.

0: HNP is not enabled in the application

1: HNP is enabled in the application

Note: Only accessible in device mode.

Bit 10 **HSHNPEN:** host set HNP enable

The application sets this bit when it has successfully enabled HNP (using the SetFeature.SetHNPEnable command) on the connected device.

- 0: Host Set HNP is not enabled
- 1: Host Set HNP is enabled

Note: Only accessible in host mode.

Bit 9 **HNPRQ:** HNP request

The application sets this bit to initiate an HNP request to the connected USB host. The application can clear this bit by writing a 0 when the host negotiation success status change bit in the OTG_GOTGINT register (HNSSCHG bit in OTG_GOTGINT) is set. The core clears this bit when the HNSSCHG bit is cleared.

- 0: No HNP request
- 1: HNP request

Note: Only accessible in device mode.

Bit 8 **HNGSCS:** Host negotiation success

The core sets this bit when host negotiation is successful. The core clears this bit when the HNP request (HNPRQ) bit in this register is set.

- 0: Host negotiation failure
- 1: Host negotiation success

Note: Only accessible in device mode.

Bit 7 **BVALOVAL:** B-peripheral session valid override value.

This bit is used to set override value for Bvalid signal when BVALOEN bit is set.

- 0: Bvalid value is '0' when BVALOEN = 1
- 1: Bvalid value is '1' when BVALOEN = 1

Note: Only accessible in device mode.

Bit 6 **BVALOEN:** B-peripheral session valid override enable.

This bit is used to enable/disable the software to override the Bvalid signal using the BVALOVAL bit.

- 0:Override is disabled and Bvalid signal from the respective PHY selected is used internally by the core
- 1:Internally Bvalid received from the PHY is overridden with BVALOVAL bit value

Note: Only accessible in device mode.

Bit 5 **AVALOVAL:** A-peripheral session valid override value.

This bit is used to set override value for Avalid signal when AVALOEN bit is set.

- 0: Avalid value is '0' when AVALOEN = 1
- 1: Avalid value is '1' when AVALOEN = 1

Note: Only accessible in host mode.

Bit 4 **AVALOEN:** A-peripheral session valid override enable.

This bit is used to enable/disable the software to override the Avalid signal using the AVALOVAL bit.

- 0:Override is disabled and Avalid signal from the respective PHY selected is used internally by the core
- 1:Internally Avalid received from the PHY is overridden with AVALOVAL bit value

Note: Only accessible in host mode.

Bit 3 **VBVALOVAL**: V_{BUS} valid override value.

This bit is used to set override value for vbusvalid signal when VBVALOEN bit is set.

0: vbusvalid value is '0' when VBVALOEN = 1

1: vbusvalid value is '1' when VBVALOEN = 1

Note: Only accessible in host mode.

Bit 2 **VBVALOEN**: V_{BUS} valid override enable.

This bit is used to enable/disable the software to override the vbusvalid signal using the VBVALOVAL bit.

0: Override is disabled and vbusvalid signal from the respective PHY selected is used internally by the core

1: Internally vbusvalid received from the PHY is overridden with VBVALOVAL bit value

Note: Only accessible in host mode.

Bit 1 **SRQ**: Session request

The application sets this bit to initiate a session request on the USB. The application can clear this bit by writing a 0 when the host negotiation success status change bit in the OTG_GOTGINT register (HNSSCHG bit in OTG_GOTGINT) is set. The core clears this bit when the HNSSCHG bit is cleared.

If the user uses the USB 1.1 full-speed serial transceiver interface to initiate the session request, the application must wait until V_{BUS} discharges to 0.2 V, after the B-session valid bit in this register (BSVLD bit in OTG_GOTGCTL) is cleared.

0: No session request

1: Session request

Note: Only accessible in device mode.

Bit 0 **SRQSCS**: Session request success

The core sets this bit when a session request initiation is successful.

0: Session request failure

1: Session request success

Note: Only accessible in device mode.

32.15.2 OTG interrupt register (OTG_GOTGINT)

Address offset: 0x04

Reset value: 0x0000 0000

The application reads this register whenever there is an OTG interrupt and clears the bits in this register to clear the OTG interrupt.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	ID CHNG	DBC DNE	ADTO CHG	HNG DET	Res.						
											rc_w1	rc_w1	rc_w1	rc_w1	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	HNSS CHG	SRSS CHG	Res.	Res.	Res.	Res.	SEDET	Res.	Res.	
						rc_w1	rc_w1					rc_w1			

Bits 31:21 Reserved, must be kept at reset value.

Bit 20 **IDCHNG:**

This bit when set indicates that there is a change in the value of the ID input pin.

Bit 19 **DBCDNE:** Debounce done

The core sets this bit when the debounce is completed after the device connect. The application can start driving USB reset after seeing this interrupt. This bit is only valid when the HNP Capable or SRP Capable bit is set in the OTG_GUSBCFG register (HNPCAP bit or SRPCAP bit in OTG_GUSBCFG, respectively).

Note: Only accessible in host mode.

Bit 18 **ADTOCHG:** A-device timeout change

The core sets this bit to indicate that the A-device has timed out while waiting for the B-device to connect.

Note: Accessible in both device and host modes.

Bit 17 **HNGDET:** Host negotiation detected

The core sets this bit when it detects a host negotiation request on the USB.

Note: Accessible in both device and host modes.

Bits 16:10 Reserved, must be kept at reset value.

Bit 9 **HNSSCHG:** Host negotiation success status change

The core sets this bit on the success or failure of a USB host negotiation request. The application must read the host negotiation success bit of the OTG_GOTGCTL register (HNGSCS bit in OTG_GOTGCTL) to check for success or failure.

Note: Accessible in both device and host modes.

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

Bit 8 **SRSSCHG:** Session request success status change

The core sets this bit on the success or failure of a session request. The application must read the session request success bit in the OTG_GOTGCTL register (SRQSCS bit in OTG_GOTGCTL) to check for success or failure.

Note: Accessible in both device and host modes.

Bit 2 **SEDET:** Session end detected

The core sets this bit to indicate that the level of the voltage on V_{BUS} is no longer valid for a B-Peripheral session when V_{BUS} < 0.8 V.

Note: Accessible in both device and host modes.

Bits 1:0 Reserved, must be kept at reset value.

32.15.3 OTG AHB configuration register (OTG_GAHBCFG)

Address offset: 0x008

Reset value: 0x0000 0000

This register can be used to configure the core after power-on or a change in mode. This register mainly contains AHB system-related configuration parameters. Do not change this register after the initial programming. The application must program this register before starting any transactions on either the AHB or the USB.

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

Note: Configuration register for USB OTG FS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	PTXFE LVL	TXFE LVL	Res.	DMAEN	HBSTLEN[3:0]				GINT MSK						
							rw	rw		rw	rw	rw	rw	rw	rw

Note: Configuration register for USB OTG HS

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

Bit 8 **PTXFELVL:** Periodic Tx FIFO empty level

Indicates when the periodic Tx FIFO empty interrupt bit in the OTG_GINTSTS register (PTXFE bit in OTG_GINTSTS) is triggered.

- 0: PTXFE (in OTG_GINTSTS) interrupt indicates that the Periodic Tx FIFO is half empty
- 1: PTXFE (in OTG_GINTSTS) interrupt indicates that the Periodic Tx FIFO is completely empty

Note: Only accessible in host mode.

Bit 7 **TXFELVL:** Tx FIFO empty level

In device mode, this bit indicates when IN endpoint Transmit FIFO empty interrupt (TXFE in OTG_DIEPINTx) is triggered:

- 0: The TXFE (in OTG_DIEPINTx) interrupt indicates that the IN endpoint Tx FIFO is half empty
- 1: The TXFE (in OTG_DIEPINTx) interrupt indicates that the IN endpoint Tx FIFO is completely empty

In host mode, this bit indicates when the nonperiodic Tx FIFO empty interrupt (NPTXFE bit in OTG_GINTSTS) is triggered:

- 0: The NPTXFE (in OTG_GINTSTS) interrupt indicates that the nonperiodic Tx FIFO is half empty
- 1: The NPTXFE (in OTG_GINTSTS) interrupt indicates that the nonperiodic Tx FIFO is completely empty

Bits 6:1 Reserved, must be kept at reset value for USB OTG FS.

Bit 6 Reserved, must be kept at reset value for USB OTG HS.

Bit 5 **DMAEN:** DMA enabled for USB OTG HS

- 0: The core operates in slave mode
- 1: The core operates in DMA mode

Bits 4:1 **HBSTLEN[3:0]:** Burst length/type for USB OTG HS

- 0000 Single: Bus transactions use single 32 bit accesses (not recommended)
- 0001 INCR: Bus transactions use unspecified length accesses (not recommended, uses the INCR AHB bus command)
- 0011 INCR4: Bus transactions target 4x 32 bit accesses
- 0101 INCR8: Bus transactions target 8x 32 bit accesses

0111 INCR16: Bus transactions based on 16x 32 bit accesses
Others: Reserved

Bit 0 **GINTMSK:** Global interrupt mask

The application uses this bit to mask or unmask the interrupt line assertion to itself.
Irrespective of this bit's setting, the interrupt status registers are updated by the core.

- 0: Mask the interrupt assertion to the application.
- 1: Unmask the interrupt assertion to the application.

Note: Accessible in both device and host modes.

32.15.4 OTG USB configuration register (OTG_GUSBCFG)

Address offset: 0x00C

Reset value: 0x0000 1440 for USB OTG FS

Reset value: 0x0000 1400 for USB OTG HS

This register can be used to configure the core after power-on or a changing to host mode or device mode. It contains USB and USB-PHY related configuration parameters. The application must program this register before starting any transactions on either the AHB or the USB. Do not make changes to this register after the initial programming.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	FD MOD	FH MOD	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
	rw	rw													
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	TRDT				HNP CAP	SRP CAP	Res.	PHY SEL	Res.	Res.	Res.	TOCAL		
		rw				rw	rw		r				rw		

Note: Configuration register for USB OTG FS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	FD MOD	FH MOD	Res.	Res.	Res.	ULPI IPD	PTCI	PCCI	TSDPS	ULPIE VBUSI	ULPIE VBUSD	ULPI CSM	ULPI AR	ULPI FSL	Res.
	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
PHYL PC	Res.	TRDT[3:0]				HNP CAP	SRP CAP	Res.	PHY SEL	Res.	ULPI SEL	Res.	TOCAL[2:0]		
rw		rw	rw	rw	rw	rw	rw		rw		rw		rw	rw	rw

Note: Configuration register for USB OTG HS

Bit 31 Reserved, must be kept at reset value.

Bit 30 **FDMOD:** Force device mode

Writing a 1 to this bit, forces the core to device mode irrespective of the OTG_ID input pin.

0: Normal mode

1: Force device mode

After setting the force bit, the application must wait at least 25 ms before the change takes effect.

Note: Accessible in both device and host modes.

Bit 29 **FHMOD:** Force host mode

Writing a 1 to this bit, forces the core to host mode irrespective of the OTG_ID input pin.

0: Normal mode

1: Force host mode

After setting the force bit, the application must wait at least 25 ms before the change takes effect.

Note: Accessible in both device and host modes.

Bits 28:26 Reserved, must be kept at reset value.

Bit 25 **ULPIIPD: ULPI interface protect disable for USB OTG HS**

This bit controls the circuitry built in the PHY to protect the ULPI interface when the link tri-states stp and data. Any pull-up or pull-down resistors employed by this feature can be disabled. Refer to the ULPI specification for more details.

- 0: Enables the interface protection circuit
- 1: Disables the interface protection circuit

Bit 24 **PTCI: Indicator pass through for USB OTG HS**

This bit controls whether the complement output is qualified with the internal V_{BUS} valid comparator before being used in the V_{BUS} state in the RX CMD. Refer to the ULPI specification for more details.

- 0: Complement Output signal is qualified with the Internal V_{BUS} valid comparator
- 1: Complement Output signal is not qualified with the Internal V_{BUS} valid comparator

Bit 23 **PCCI: Indicator complement for USB OTG HS**

This bit controls the PHY to invert the ExternalVbusIndicator input signal, and generate the complement output. Refer to the ULPI specification for more details.

- 0: PHY does not invert the ExternalVbusIndicator signal
- 1: PHY inverts ExternalVbusIndicator signal

Bit 22 **TSDPS: TermSel DLine pulsing selection for USB OTG HS**

This bit selects utmi_termselect to drive the data line pulse during SRP (session request protocol).

- 0: Data line pulsing using utmi_txvalid (default)
- 1: Data line pulsing using utmi_termsel

Bit 21 **ULPIEVBUSI: ULPI external V_{BUS} indicator for USB OTG HS**

This bit indicates to the ULPI PHY to use an external V_{BUS} overcurrent indicator.

- 0: PHY uses an internal V_{BUS} valid comparator
- 1: PHY uses an external V_{BUS} valid comparator

Bit 20 **ULPIEVBUSD: ULPI External V_{BUS} Drive for USB OTG HS**

This bit selects between internal or external supply to drive 5 V on V_{BUS} , in the ULPI PHY.

- 0: PHY drives V_{BUS} using internal charge pump (default)
- 1: PHY drives V_{BUS} using external supply.

Bit 19 **ULPICSM: ULPI clock SuspendM for USB OTG HS**

This bit sets the ClockSuspendM bit in the interface control register on the ULPI PHY. This bit applies only in the serial and carkit modes.

- 0: PHY powers down the internal clock during suspend
- 1: PHY does not power down the internal clock

Bit 18 **ULPIAR: ULPI Auto-resume for USB OTG HS**

This bit sets the AutoResume bit in the interface control register on the ULPI PHY.

- 0: PHY does not use AutoResume feature
- 1: PHY uses AutoResume feature

Bit 17 **ULPIFSLS: ULPI FS/LS select for USB OTG HS**

The application uses this bit to select the FS/LS serial interface for the ULPI PHY. This bit is valid only when the FS serial transceiver is selected on the ULPI PHY.

- 0: ULPI interface
- 1: ULPI FS/LS serial interface

Bit 16 Reserved, must be kept at reset value.

Bit 15 **PHYLPC:** PHY Low-power clock select for USB OTG HS

This bit selects either 480 MHz or 48 MHz (low-power) PHY mode. In FS and LS modes, the PHY can usually operate on a 48 MHz clock to save power.

0: 480 MHz internal PLL clock

1: 48 MHz external clock

In 480 MHz mode, the UTMI interface operates at either 60 or 30 MHz, depending on whether the 8- or 16-bit data width is selected. In 48 MHz mode, the UTMI interface operates at 48 MHz in FS and LS modes.

Bit 14 Reserved, must be kept at reset value.

Bits 13:10 **TRDT[3:0]:** USB turnaround time

These bits allows to set the turnaround time in PHY clocks. They must be configured according to [Table 232: TRDT values \(FS\)](#) or [Table 233: TRDT values \(HS\)](#), depending on the application AHB frequency. Higher TRDT values allow stretching the USB response time to IN tokens in order to compensate for longer AHB read access latency to the data FIFO.

Note: Only accessible in device mode.

Bit 9 **HNPCAP:** HNP-capable

The application uses this bit to control the OTG_FS/OTG_HS controller's HNP capabilities.

0: HNP capability is not enabled.

1: HNP capability is enabled.

Note: Accessible in both device and host modes.

Bit 8 **SRPCAP:** SRP-capable

The application uses this bit to control the OTG_FS/OTG_HS controller's SRP capabilities. If the core operates as a non-SRP-capable

B-device, it cannot request the connected A-device (host) to activate V_{BUS} and start a session.

0: SRP capability is not enabled.

1: SRP capability is enabled.

Note: Accessible in both device and host modes.

Bit 7 Reserved, must be kept at reset value.

Bit 6 **PHYSEL:** Full Speed serial transceiver select for USB OTG FS

This bit is always 1 with read-only access.

Bit 6 **PHYSEL:** Full speed serial transceiver select for USB OTG HS

0: USB 2.0 external ULPI high-speed PHY or internal UTMI high-speed PHY (see also ULPISEL).

1: USB 1.1 full-speed serial transceiver.

Bit 5 Reserved, must be kept at reset value.

Bit 4 **ULPISEL**: Select which high speed interface is to be used

- 0: UTMI interface is selected (internal high speed PHY)
- 1: ULPI interface is selected (external high speed PHY)

Bit 3 Reserved, must be kept at reset value.

Bits 2:0 **TOCAL[2:0]**: FS timeout calibration

The number of PHY clocks that the application programs in this field is added to the full-speed interpacket timeout duration in the core to account for any additional delays introduced by the PHY. This can be required, because the delay introduced by the PHY in generating the line state condition can vary from one PHY to another.

The USB standard timeout value for full-speed operation is 16 to 18 (inclusive) bit times. The application must program this field based on the speed of enumeration. The number of bit times added per PHY clock is 0.25 bit times.

Table 232. TRDT values (FS)

AHB frequency range (MHz)		TRDT minimum value
Min	Max	
14.2	15	0xF
15	16	0xE
16	17.2	0xD
17.2	18.5	0xC
18.5	20	0xB
20	21.8	0xA
21.8	24	0x9
24	27.5	0x8
27.5	32	0x7
32	-	0x6

Table 233. TRDT values (HS)

AHB frequency range (MHz)		TRDT minimum value
Min	Max	
30	-	0x9

32.15.5 OTG reset register (OTG_GRSTCTL)

Address offset: 0x10

Reset value: 0x8000 0000

The application uses this register to reset various hardware features inside the core.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
AHB IDL	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	
r															r	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	Res.	Res.	Res.	Res.	TXFNUM						TXF FLSH	RXF FLSH	Res.	FCRST	PSRST	CSRST
					rw						rs	rs		rs	rs	r

Note: Configuration register for USB OTG FS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
AHB IDL	DMAR EQ	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	
r	r															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	Res.	Res.	Res.	Res.	TXFNUM[4:0]						TXF FLSH	RXF FLSH	Res.	Res.	PSRST	CSRST
					rw	rw	rw	rw	rw	rs	rs			rs	rs	

Note: Configuration register for USB OTG HS

Bit 31 **AHBIDL:** AHB master idle

Indicates that the AHB master state machine is in the Idle condition.

Note: Accessible in both device and host modes.

Bits 30:11 Reserved, must be kept at reset value for USB OTG FS.

Bit 30 **DMAREQ:** DMA request signal enabled for USB OTG HS

This bit indicates that the DMA request is in progress. Used for debug.

Bits 29:11 Reserved, must be kept at reset value for USB OTG HS.

Bits 10:6 **TXFNUM[4:0]:** Tx FIFO number

This is the FIFO number that must be flushed using the Tx FIFO Flush bit. This field must not be changed until the core clears the Tx FIFO Flush bit.

00000:

- Non-periodic Tx FIFO flush in host mode
- Tx FIFO 0 flush in device mode

00001:

- Periodic Tx FIFO flush in host mode
- Tx FIFO 1 flush in device mode

00010: Tx FIFO 2 flush in device mode

...

01111: Tx FIFO 15 flush in device mode

10000: Flush all the transmit FIFOs in device or host mode.

Note: Accessible in both device and host modes.

Bit 5 TXFFLSH: Tx FIFO flush

This bit selectively flushes a single or all transmit FIFOs, but cannot do so if the core is in the midst of a transaction.

The application must write this bit only after checking that the core is neither writing to the Tx FIFO nor reading from the Tx FIFO. Verify using these registers:

Read—NAK Effective interrupt ensures the core is not reading from the FIFO

Write—AHBIDL bit in OTG_GRSTCTL ensures the core is not writing anything to the FIFO.

Flushing is normally recommended when FIFOs are reconfigured. FIFO flushing is also recommended during device endpoint disable. The application must wait until the core clears this bit before performing any operations. This bit takes eight clocks to clear, using the slower clock of phy_clk or hclk.

Note: Accessible in both device and host modes.

Bit 4 RXFFLSH: Rx FIFO flush

The application can flush the entire Rx FIFO using this bit, but must first ensure that the core is not in the middle of a transaction.

The application must only write to this bit after checking that the core is neither reading from the Rx FIFO nor writing to the Rx FIFO.

The application must wait until the bit is cleared before performing any other operations. This bit requires 8 clocks (slowest of PHY or AHB clock) to clear.

Note: Accessible in both device and host modes.

Bit 3 Reserved, must be kept at reset value.**Bit 2 FCRST: Host frame counter reset for USB OTG FS**

The application writes this bit to reset the frame number counter inside the core. When the frame counter is reset, the subsequent SOF sent out by the core has a frame number of 0.

When application writes '1' to the bit, it might not be able to read back the value as it will get cleared by the core in a few clock cycles.

Note: Only accessible in host mode.

Bit 2 Reserved, must be kept at reset value for USB OTG HS.

Bit 1 **PSRST:** Partial soft reset

Resets the internal state machines but keeps the enumeration info. Could be used to recover some specific PHY errors.

Note: Accessible in both device and host modes.

Bit 0 **CSRST:** Core soft reset

Resets the HCLK and PHY clock domains as follows:

Clears the interrupts and all the CSR register bits except for the following bits:

- GATEHCLK bit in OTG_PCGCCTL
- STPPCLK bit in OTG_PCGCCTL
- FSLSPCS bits in OTG_HCFG
- DSPD bit in OTG_DCFG
- SDIS bit in OTG_DCTL
- OTG_GCCFG register

All module state machines (except for the AHB slave unit) are reset to the Idle state, and all the transmit FIFOs and the receive FIFO are flushed.

Any transactions on the AHB Master are terminated as soon as possible, after completing the last data phase of an AHB transfer. Any transactions on the USB are terminated immediately. The application can write to this bit any time it wants to reset the core. This is a self-clearing bit and the core clears this bit after all the necessary logic is reset in the core, which can take several clocks, depending on the current state of the core. Once this bit has been cleared, the software must wait at least 3 PHY clocks before accessing the PHY domain (synchronization delay). The software must also check that bit 31 in this register is set to 1 (AHB Master is Idle) before starting any operation.

Typically, the software reset is used during software development and also when the user dynamically changes the PHY selection bits in the above listed USB configuration registers. When you change the PHY, the corresponding clock for the PHY is selected and used in the PHY domain. Once a new clock is selected, the PHY domain has to be reset for proper operation.

Note: Accessible in both device and host modes.

32.15.6 OTG core interrupt register (OTG_GINTSTS)

Address offset: 0x014

Reset value: 0x1400 0020

This register interrupts the application for system-level events in the current mode (device mode or host mode).

Some of the bits in this register are valid only in host mode, while others are valid in device mode only. This register also indicates the current mode. To clear the interrupt status bits of the rc_w1 type, the application must write 1 into the bit.

The FIFO status interrupts are read-only; once software reads from or writes to the FIFO while servicing these interrupts, FIFO interrupt conditions are cleared automatically.

The application must clear the OTG_GINTSTS register at initialization before unmasking the interrupt bit to avoid any interrupts generated prior to initialization.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WKUP INT	SRQ INT	DISC INT	CIDS CHG	LPM INT	PTXFE	HCINT	HPRT INT	RST DET	Res.	IPXFR/ IN COMP ISO OUT	IISOI XFR	OEP INT	IEPINT	Res.	Res.
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	r	r	r	rc_w1		rc_w1	rc_w1	r	r		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EOPF	ISOO DRP	ENUM DNE	USB RST	USB SUSP	ESUSP	Res.	Res.	GO NAK EFF	GI NAK EFF	NPTXF E	RXF LVL	SOF	OTG INT	MMIS	CMOD
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1			r	r	r	r	rc_w1	r	rc_w1	r

Note: Configuration register for USB OTG FS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WKUP INT	SRQ INT	DISC INT	CIDS CHG	Res.	PTXFE	HCINT	HPRT INT	Res.	DATAF SUSP	IPXFR/ IN COMP ISO OUT	IISOI XFR	OEP INT	IEPINT	Res.	Res.
rc_w1	rc_w1	rc_w1	rc_w1		r	r	r		rc_w1	rc_w1	rc_w1	r	r		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EOPF	ISOO DRP	ENUM DNE	USB RST	USB SUSP	ESUSP	Res.	Res.	GO NAK EFF	GI NAK EFF	NPTXF E	RXF LVL	SOF	OTG INT	MMIS	CMOD
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1			r	r	r	r	rc_w1	r	rc_w1	r

Note: Configuration register for USB OTG HS

Bit 31 **WKUPINT:** Resume/remote wakeup detected interrupt

Wakeup interrupt during suspend(L2) or LPM(L1) state.

- During suspend(L2):

In device mode, this interrupt is asserted when a resume is detected on the USB. In host mode, this interrupt is asserted when a remote wakeup is detected on the USB.

- During LPM(L1):

This interrupt is asserted for either host initiated resume or device initiated remote wakeup on USB.

Note: Accessible in both device and host modes.

Bit 30 **SRQINT:** Session request/new session detected interrupt

In host mode, this interrupt is asserted when a session request is detected from the device.

In device mode, this interrupt is asserted when V_{BUS} is in the valid range for a B-peripheral device. Accessible in both device and host modes.

Bit 29 **DISCINT:** Disconnect detected interrupt

Asserted when a device disconnect is detected.

Note: Only accessible in host mode.

Bit 28 **CIDSCHG:** Connector ID status change

The core sets this bit when there is a change in connector ID status.

Note: Accessible in both device and host modes.

Bit 27 **LPMINT:** LPM interrupt

In device mode, this interrupt is asserted when the device receives an LPM transaction and responds with a non-ERRORed response.

In host mode, this interrupt is asserted when the device responds to an LPM transaction with a non-ERRORed response or when the host core has completed LPM transactions for the programmed number of times (RETRYCNT bit in OTG_GLPMCFG).

This field is valid only if the LPMEN bit in OTG_GLPMCFG is set to 1.

Bit 27 Reserved, must be kept at reset value for USB OTG FS.

Bit 26 **PTXFE:** Periodic Tx FIFO empty

Asserted when the periodic transmit FIFO is either half or completely empty and there is space for at least one entry to be written in the periodic request queue. The half or completely empty status is determined by the periodic Tx FIFO empty level bit in the OTG_GAHBCFG register (PTXFELVL bit in OTG_GAHBCFG).

Note: Only accessible in host mode.

Bit 25 **HCINT:** Host channels interrupt

The core sets this bit to indicate that an interrupt is pending on one of the channels of the core (in host mode). The application must read the OTG_HAINT register to determine the exact number of the channel on which the interrupt occurred, and then read the corresponding OTG_HCINTx register to determine the exact cause of the interrupt. The application must clear the appropriate status bit in the OTG_HCINTx register to clear this bit.

Note: Only accessible in host mode.

Bit 24 **HPRTINT:** Host port interrupt

The core sets this bit to indicate a change in port status of one of the OTG_FS/OTG_HS controller ports in host mode. The application must read the OTG_HPRT register to determine the exact event that caused this interrupt. The application must clear the appropriate status bit in the OTG_HPRT register to clear this bit.

Note: Only accessible in host mode.

Bit 23 **RSTDET:** Reset detected interrupt

In device mode, this interrupt is asserted when a reset is detected on the USB in partial power-down mode when the device is in suspend.

Note: Only accessible in device mode.

Bit 23 Reserved, must be kept at reset value for USB OTG HS.

Bit 22 Reserved, must be kept at reset value for USB OTG FS.

Bit 22 **DATAFSUSP:** Data fetch suspended for USB OTG HS

This interrupt is valid only in DMA mode. This interrupt indicates that the core has stopped fetching data for IN endpoints due to the unavailability of TxFIFO space or request queue space. This interrupt is used by the application for an endpoint mismatch algorithm. For example, after detecting an endpoint mismatch, the application:

- Sets a global nonperiodic IN NAK handshake
- Disables IN endpoints
- Flushes the FIFO
- Determines the token sequence from the IN token sequence learning queue
- Re-enables the endpoints

Clears the global nonperiodic IN NAK handshake If the global nonperiodic IN NAK is cleared, the core has not yet fetched data for the IN endpoint, and the IN token is received: the core generates an “IN token received when FIFO empty” interrupt. The OTG then sends a NAK response to the host. To avoid this scenario, the application can check the FetSusp interrupt in OTG_GINTSTS, which ensures that the FIFO is full before clearing a global NAK handshake. Alternatively, the application can mask the “IN token received when FIFO empty” interrupt when clearing a global IN NAK handshake.

Bit 21 **IPXFR:** Incomplete periodic transfer

In host mode, the core sets this interrupt bit when there are incomplete periodic transactions still pending, which are scheduled for the current frame.

INCOMPISOOUT: Incomplete isochronous OUT transfer

In device mode, the core sets this interrupt to indicate that there is at least one isochronous OUT endpoint on which the transfer is not completed in the current frame. This interrupt is asserted along with the End of periodic frame interrupt (EOPF) bit in this register.

Bit 20 **IISOIXFR:** Incomplete isochronous IN transfer

The core sets this interrupt to indicate that there is at least one isochronous IN endpoint on which the transfer is not completed in the current frame. This interrupt is asserted along with the End of periodic frame interrupt (EOPF) bit in this register.

Note: Only accessible in device mode.

Bit 19 **OEPINT:** OUT endpoint interrupt

The core sets this bit to indicate that an interrupt is pending on one of the OUT endpoints of the core (in device mode). The application must read the OTG_DAINT register to determine the exact number of the OUT endpoint on which the interrupt occurred, and then read the corresponding OTG_DOEPINTx register to determine the exact cause of the interrupt. The application must clear the appropriate status bit in the corresponding OTG_DOEPINTx register to clear this bit.

Note: Only accessible in device mode.

Bit 18 IEPINT: IN endpoint interrupt

The core sets this bit to indicate that an interrupt is pending on one of the IN endpoints of the core (in device mode). The application must read the OTG_DAINTR register to determine the exact number of the IN endpoint on which the interrupt occurred, and then read the corresponding OTG_DIEPINTx register to determine the exact cause of the interrupt. The application must clear the appropriate status bit in the corresponding OTG_DIEPINTx register to clear this bit.

Note: Only accessible in device mode.

Bits 17:16 Reserved, must be kept at reset value.

Bit 15 EOPF: End of periodic frame interrupt

Indicates that the period specified in the periodic frame interval field of the OTG_DCFG register (PFIVL bit in OTG_DCFG) has been reached in the current frame.

Note: Only accessible in device mode.

Bit 14 ISOODRP: Isochronous OUT packet dropped interrupt

The core sets this bit when it fails to write an isochronous OUT packet into the Rx FIFO because the Rx FIFO does not have enough space to accommodate a maximum size packet for the isochronous OUT endpoint.

Note: Only accessible in device mode.

Bit 13 ENUMDNE: Enumeration done

The core sets this bit to indicate that speed enumeration is complete. The application must read the OTG_DSTS register to obtain the enumerated speed.

Note: Only accessible in device mode.

Bit 12 USBRST: USB reset

The core sets this bit to indicate that a reset is detected on the USB.

Note: Only accessible in device mode.

Bit 11 USBSUSP: USB suspend

The core sets this bit to indicate that a suspend was detected on the USB. The core enters the suspended state when there is no activity on the data lines for an extended period of time.

Note: Only accessible in device mode.

Bit 10 ESUSP: Early suspend

The core sets this bit to indicate that an Idle state has been detected on the USB for 3 ms.

Note: Only accessible in device mode.

Bits 9:8 Reserved, must be kept at reset value.

Bit 7 GONAKEFF: Global OUT NAK effective

Indicates that the Set global OUT NAK bit in the OTG_DCTL register (SGONAK bit in OTG_DCTL), set by the application, has taken effect in the core. This bit can be cleared by writing the Clear global OUT NAK bit in the OTG_DCTL register (CGONAK bit in OTG_DCTL).

Note: Only accessible in device mode.

Bit 6 GINAKEFF: Global IN non-periodic NAK effective

Indicates that the Set global non-periodic IN NAK bit in the OTG_DCTL register (SGINAK bit in OTG_DCTL), set by the application, has taken effect in the core. That is, the core has sampled the Global IN NAK bit set by the application. This bit can be cleared by clearing the Clear global non-periodic IN NAK bit in the OTG_DCTL register (CGINAK bit in OTG_DCTL).

This interrupt does not necessarily mean that a NAK handshake is sent out on the USB. The STALL bit takes precedence over the NAK bit.

Note: Only accessible in device mode.

Bit 5 NPTXFE: Non-periodic Tx FIFO empty

This interrupt is asserted when the non-periodic Tx FIFO is either half or completely empty, and there is space for at least one entry to be written to the non-periodic transmit request queue. The half or completely empty status is determined by the non-periodic Tx FIFO empty level bit in the OTG_GAHBCFG register (TXFELVL bit in OTG_GAHBCFG).

Note: Accessible in host mode only.

Bit 4 RXFLVL: Rx FIFO non-empty

Indicates that there is at least one packet pending to be read from the Rx FIFO.

Note: Accessible in both host and device modes.

Bit 3 SOF: Start of frame

In host mode, the core sets this bit to indicate that an SOF (FS), or Keep-Alive (LS) is transmitted on the USB. The application must write a 1 to this bit to clear the interrupt.

In device mode, in the core sets this bit to indicate that an SOF token has been received on the USB. The application can read the OTG_DSTS register to get the current frame number. This interrupt is seen only when the core is operating in FS.

Note: This register may return '1' if read immediately after power on reset. If the register bit reads '1' immediately after power on reset it does not indicate that an SOF has been sent (in case of host mode) or SOF has been received (in case of device mode). The read value of this interrupt is valid only after a valid connection between host and device is established. If the bit is set after power on reset the application can clear the bit.

Note: Accessible in both host and device modes.

Bit 2 OTGINT: OTG interrupt

The core sets this bit to indicate an OTG protocol event. The application must read the OTG interrupt status (OTG_GOTGINT) register to determine the exact event that caused this interrupt. The application must clear the appropriate status bit in the OTG_GOTGINT register to clear this bit.

Note: Accessible in both host and device modes.

Bit 1 MMIS: Mode mismatch interrupt

The core sets this bit when the application is trying to access:

- A host mode register, when the core is operating in device mode
- A device mode register, when the core is operating in host mode

The register access is completed on the AHB with an OKAY response, but is ignored by the core internally and does not affect the operation of the core.

Note: Accessible in both host and device modes.

Bit 0 CMOD: Current mode of operation

Indicates the current mode.

- 0: Device mode
- 1: Host mode

Note: Accessible in both host and device modes.

32.15.7 OTG interrupt mask register (OTG_GINTMSK)

Address offset: 0x018

Reset value: 0x0000 0000

This register works with the core interrupt register to interrupt the application. When an interrupt bit is masked, the interrupt associated with that bit is not generated. However, the core interrupt (OTG_GINTSTS) register bit corresponding to that interrupt is still set.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WUIM	SRQIM	DISCINT	CIDSC HGM	LPMIN TM	PTXFE M	HCIM	PRTIM	RSTDE TM	Res.	IPXFR M/IISO OXFR M	IISOIX FRM	OEPIN T	IEPINT	Res.	Res.
rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EOPFM	ISOOD RPM	ENUM DNEM	USBRST	USBSU SPM	ESUSPM	Res.	Res.	GONA KEFFM	GINAK EFFM	NPTXF EM	RXFLV LM	SOFM	OTGIN T	MMISM	Res.
rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw	rw	

Note: Configuration register for USB OTG FS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WUIM	SRQIM	DISCINT	CIDSC HGM	LPMIN TM	PTXFE M	HCIM	PRTIM	RSTDE TM	FSUS PM	IPXFR M/IISO OXFR M	IISOIX FRM	OEPIN T	IEPINT	Res.	Res.
rw	rw	rw	rw	rw	rw	rw	r	rw	rw	rw	rw	rw	rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EOPFM	ISOOD RPM	ENUM DNEM	USBRST	USBSU SPM	ESUSPM	Res.	Res.	GONA KEFFM	GINAK EFFM	NPTXF EM	RXFLV LM	SOFM	OTGIN T	MMISM	Res.
rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw	rw	

Note: Configuration register for USB OTG HS

Bit 31 **WUIM**: Resume/remote wakeup detected interrupt mask

0: Masked interrupt

1: Unmasked interrupt

Note: Accessible in both host and device modes.

Bit 30 **SRQIM**: Session request/new session detected interrupt mask

0: Masked interrupt

1: Unmasked interrupt

Note: Accessible in both host and device modes.

Bit 29 **DISCINT**: Disconnect detected interrupt mask

0: Masked interrupt

1: Unmasked interrupt

Note: Only accessible in host mode.

Bit 28 **CIDSC HGM**: Connector ID status change mask

0: Masked interrupt

1: Unmasked interrupt

Note: Accessible in both host and device modes.

Bit 27 **LPMINTM:** LPM interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Accessible in both host and device modes.

Bit 26 **PTXFEM:** Periodic Tx FIFO empty mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Only accessible in host mode.

Bit 25 **HCIM:** Host channels interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Only accessible in host mode.

Bit 24 **PRTIM:** Host port interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Only accessible in host mode.

Bit 23 **RSTDETM:** Reset detected interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Only accessible in device mode.

Bit 22 Reserved, must be kept at reset value for USB OTG FS.

Bit 22 **FSUSPM:** Data fetch suspended mask for USB OTG HS

- 0: Masked interrupt
- 1: Unmasked interrupt

Only accessible in peripheral mode.

Bit 21 **IPXFRM:** Incomplete periodic transfer mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Only accessible in host mode.

IISOXXFRM: Incomplete isochronous OUT transfer mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Only accessible in device mode.

Bit 20 **IISOIXFRM:** Incomplete isochronous IN transfer mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Only accessible in device mode.

Bit 19 **OEPINT:** OUT endpoints interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Only accessible in device mode.

Bit 18 **IEPINT:** IN endpoints interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Only accessible in device mode.

Bits 17:16 Reserved, must be kept at reset value.

Bit 15 **EOPFM**: End of periodic frame interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Only accessible in device mode.

Bit 14 **ISOODRPM**: Isochronous OUT packet dropped interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Only accessible in device mode.

Bit 13 **ENUMDNEM**: Enumeration done mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Only accessible in device mode.

Bit 12 **USBRST**: USB reset mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Only accessible in device mode.

Bit 11 **USBSUSPM**: USB suspend mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Only accessible in device mode.

Bit 10 **ESUSPM**: Early suspend mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Only accessible in device mode.

Bits 9:8 Reserved, must be kept at reset value.

Bit 7 **GONAKEFFM**: Global OUT NAK effective mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Only accessible in device mode.

Bit 6 **GINAKEFFM**: Global non-periodic IN NAK effective mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Only accessible in device mode.

Bit 5 **NPTXFEM**: Non-periodic Tx FIFO empty mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Only accessible in host mode.

Bit 4 **RXFLVLM**: Receive FIFO non-empty mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Accessible in both device and host modes.

Bit 3 **SOFM:** Start of frame mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Accessible in both device and host modes.

Bit 2 **OTGINT:** OTG interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Accessible in both device and host modes.

Bit 1 **MMISM:** Mode mismatch interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Note: Accessible in both device and host modes.

Bit 0 Reserved, must be kept at reset value.

32.15.8 OTG receive status debug read/OTG status read and pop registers (OTG_GRXSTS[0]/OTG_GRXSTS[1])

Address offset for read: 0x01C

Address offset for pop: 0x020

Reset value: 0x0000 0000

A read to the receive status debug read register returns the contents of the top of the receive FIFO. A read to the receive status read and pop register additionally pops the top data entry out of the Rx FIFO.

The receive status contents must be interpreted differently in host and device modes. The core ignores the receive status pop/read when the receive FIFO is empty and returns a value of 0x0000 0000. The application must only pop the receive status FIFO when the receive FIFO non-empty bit of the core interrupt register (RXFLVL bit in OTG_GINTSTS) is asserted.

Host mode:

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

DPID	BCNT[10:0]												CHNUM[3:0]		
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:21 Reserved, must be kept at reset value.

Bits 20:17 **PKTSTS[3:0]:** Packet status

Indicates the status of the received packet

0010: IN data packet received

0011: IN transfer completed (triggers an interrupt)

0101: Data toggle error (triggers an interrupt)

0111: Channel halted (triggers an interrupt)

Others: Reserved

Bits 16:15 **DPID:** Data PID

Indicates the data PID of the received packet

00: DATA0

10: DATA1

01: DATA2

11: MDATA

Bits 14:4 **BCNT[10:0]:** Byte count

Indicates the byte count of the received IN data packet.

Bits 3:0 **CHNUM[3:0]:** Channel number

Indicates the channel number to which the current received packet belongs.

Device mode:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	STSPH ST	Res.	Res.	FRMNUM[3:0]				PKTSTS[3:0]				DPID[1]
				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
DPID[0]	BCNT[10:0]										EPNUM[3:0]				
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:28 Reserved, must be kept at reset value.

Bit 27 **STSPHST:** Status phase start

Indicates the start of the status phase for a control write transfer. This bit is set along with the OUT transfer completed PKTSTS pattern.

Bits 26:25 Reserved, must be kept at reset value.

Bits 24:21 **FRMNUM[3:0]:** Frame number

This is the least significant 4 bits of the frame number in which the packet is received on the USB. This field is supported only when isochronous OUT endpoints are supported.

Bits 20:17 **PKTSTS[3:0]:** Packet status

Indicates the status of the received packet

0001: Global OUT NAK (triggers an interrupt)

0010: OUT data packet received

0011: OUT transfer completed (triggers an interrupt)

0100: SETUP transaction completed (triggers an interrupt)

0110: SETUP data packet received

Others: Reserved

Bits 16:15 **DPID[1:0]:** Data PID

Indicates the data PID of the received OUT data packet

00: DATA0

10: DATA1

01: DATA2

11: MDATA

Bits 14:4 **BCNT[10:0]:** Byte count

Indicates the byte count of the received data packet.

Bits 3:0 **EPNUM[3:0]:** Endpoint number

Indicates the endpoint number to which the current received packet belongs.

32.15.9 OTG receive FIFO size register (OTG_GRXFSIZ)

Address offset: 0x024

Reset value: 0x0000 0200 for USB OTG FS

Reset value: 0x0000 0400 for USB OTG HS

The application can program the RAM size that must be allocated to the Rx FIFO.

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

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

Bits 15:0 **RXFD[15:0]: Rx FIFO depth**

This value is in terms of 32-bit words.

Minimum value is 16

Maximum value is 1024

Programmed values must respect the available FIFO memory allocation and must not exceed the power-on value.

32.15.10 OTG host non-periodic transmit FIFO size register (OTG_HNPTXFSIZ)/Endpoint 0 Transmit FIFO size (OTG_DIEPTXF0)

Address offset: 0x028

Reset value: 0x0200 0200

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NPTXFD/TX0FD[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
NPTXFSA/TX0FSA[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Host mode

Bits 31:16 **NPTXFD[15:0]: Non-periodic Tx FIFO depth**

This value is in terms of 32-bit words.

Minimum value is 16

Programmed values must respect the available FIFO memory allocation and must not exceed the power-on value.

Bits 15:0 **NPTXFSA[15:0]: Non-periodic transmit RAM start address**

This field configures the memory start address for non-periodic transmit FIFO RAM.

Device mode

Bits 31:16 **TX0FD**: Endpoint 0 Tx FIFO depth

This value is in terms of 32-bit words.

Minimum value is 16

Programmed values must respect the available FIFO memory allocation and must not exceed the power-on value.

Bits 15:0 **TX0FSA**: Endpoint 0 transmit RAM start address

This field configures the memory start address for the endpoint 0 transmit FIFO RAM.

32.15.11 OTG non-periodic transmit FIFO/queue status register (OTG_HNPTXSTS)

Address offset: 0x02C

Reset value: 0x0008 0200 for USB OTG FS

Reset value: 0x0008 0400 for USB OTG HS

Note: In device mode, this register is not valid.

This read-only register contains the free space information for the non-periodic Tx FIFO and the non-periodic transmit request queue.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	NPTXQTOP[6:0]							NPTQXSAR[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
NPTXFSAV[15:0]																
	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bit 31 Reserved, must be kept at reset value.

Bits 30:24 **NPTXQTOP[6:0]**: Top of the non-periodic transmit request queue

Entry in the non-periodic Tx request queue that is currently being processed by the MAC.

Bits 30:27: Channel/endpoint number

Bits 26:25:

00: IN/OUT token

01: Zero-length transmit packet (device IN/host OUT)

11: Channel halt command

Bit 24: Terminate (last entry for selected channel/endpoint)

Bits 23:16 **NPTQXSAR[7:0]**: Non-periodic transmit request queue space available

Indicates the amount of free space available in the non-periodic transmit request queue.

This queue holds both IN and OUT requests.

0: Non-periodic transmit request queue is full

1: 1 location available

2: locations available

n: n locations available ($0 \leq n \leq 8$)

Others: Reserved

Bits 15:0 **NPTXFSAV[15:0]**: Non-periodic Tx FIFO space available

Indicates the amount of free space available in the non-periodic Tx FIFO.

Values are in terms of 32-bit words.

0: Non-periodic Tx FIFO is full

1: 1 word available

2: 2 words available

n: n words available (where $0 \leq n \leq 512$)

Others: Reserved

32.15.12 OTG general core configuration register (OTG_GCCFG)

Address offset: 0x038

Reset value: 0x0000 XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	VBDEN	SDEN	PDEN	DCDEN	BCDEN	PWR DWN									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	rw	rw	rw	rw	rw	rw									
Res.	Res.	PS2 DET	SDET	PDET	DCDET										
											r	r	r	r	

For USB OTG FS

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

For USB OTG HS

Bits 31:22 Reserved, must be kept at reset value.

Bit 21 **VBDEN:** USB V_{BUS} detection enable

Enables V_{BUS} sensing comparators to detect V_{BUS} valid levels on the V_{BUS} PAD for USB host and device operation. If HNP and/or SRP support is enabled, V_{BUS} comparators are automatically enabled independently of VBDEN value.

0 = V_{BUS} detection disabled
1 = V_{BUS} detection enabled

Bit 20 **SDEN:** Secondary detection (SD) mode enable

This bit is set by the software to put the BCD into SD mode. Only one detection mode (DCD, PD, SD or OFF) should be selected to work correctly

Bit 19 **PDEN:** Primary detection (PD) mode enable

This bit is set by the software to put the BCD into PD mode. Only one detection mode (DCD, PD, SD or OFF) should be selected to work correctly.

Bit 18 **DCDEN:** Data contact detection (DCD) mode enable

This bit is set by the software to put the BCD into DCD mode. Only one detection mode (DCD, PD, SD or OFF) should be selected to work correctly.

Bit 17 **BCDEN:** Battery charging detector (BCD) enable

This bit is set by the software to enable the BCD support within the USB device. When enabled, the USB PHY is fully controlled by BCD and cannot be used for normal communication. Once the BCD discovery is finished, the BCD should be placed in OFF mode by clearing this bit to '0' in order to allow the normal USB operation.

Bit 16 **PWRDWN:** Power down control

Used to activate the transceiver in transmission/reception. When reset, the transceiver is kept in power-down. When set, the BCD function must be off (BCDEN=0).

0 = USB FS transceiver disabled
1 = USB FS transceiver enabled

Bits 15:4 Reserved, must be kept at reset value.

Bit 3 **PS2DET:** DM pull-up detection status

This bit is active only during PD and gives the result of comparison between DM voltage level and VLGC threshold. In normal situation, the DM level should be below this threshold. If it is above, it means that the DM is externally pulled high. This can be caused by connection to a PS2 port (which pulls-up both DP and DM lines) or to some proprietary charger not following the BCD specification.

0: Normal port detected (connected to SDP, CDP or DCP)
1: PS2 port or proprietary charger detected

Bit 2 **SDET**: Secondary detection (SD) status

This bit gives the result of SD.

0: CDP detected

1: DCP detected

Bit 1 **PDET**: Primary detection (PD) status

This bit gives the result of PD.

0: no BCD support detected (connected to SDP or proprietary device).

1: BCD support detected (connected to CDP or DCP).

Bit 0 **DCDET**: Data contact detection (DCD) status

This bit gives the result of DCD.

0: data lines contact not detected

1: data lines contact detected

32.15.13 OTG core ID register (OTG_CID)

Address offset: 0x03C

Reset value: 0x0000 3000 for USB OTG FS

Reset value: 0x0000 3100 for USB OTG HS

This is a register containing the Product ID as reset value.

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

Bits 31:0 **PRODUCT_ID[31:0]**: Product ID field

Application-programmable ID field.

32.15.14 OTG core LPM configuration register (OTG_GLPMCFG)

Address offset: 0x54

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	EN BESL	LPMRCNTSTS[2:0]			SND LPM	LPMRCNT[2:0]			LPMCHIDX[3:0]				L1RSM OK
				rw	r	r	r	rs	rw	rw	rw	rw	rw	rw	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SLP STS	LPMRSP[1:0]		L1DS EN	BESLTHRS[3:0]				L1SS EN	REM WAKE	BESL[3:0]				LPM ACK	LPM EN
r	r	r	rw	rw	rw	rw	rw	rw	rw/r	rw/r	rw/r	rw/r	rw/r	rw	rw

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

Bit 28 **ENBESL:** Enable best effort service latency

This bit enables the BESL feature as defined in the LPM errata:

0:The core works as described in the following document:

USB 2.0 Link Power Management Addendum Engineering Change Notice to the USB 2.0 specification, July 16, 2007

1:The core works as described in the LPM Errata:

Errata for USB 2.0 ECN: Link Power Management (LPM) - 7/2007

Note: Only the updated behavior (described in LPM Errata) is considered in this document and so the ENBESL bit should be set to '1' by application SW.

Bits 27:25 **LPMRCNTSTS[2:0]:** LPM retry count status

Number of LPM host retries still remaining to be transmitted for the current LPM sequence.

Note: Accessible only in host mode.

Bit 24 **SNDLPM:** Send LPM transaction

When the application software sets this bit, an LPM transaction containing two tokens, EXT and LPM is sent. The hardware clears this bit once a valid response (STALL, NYET, or ACK) is received from the device or the core has finished transmitting the programmed number of LPM retries.

Note: This bit must be set only when the host is connected to a local port.

Note: Accessible only in host mode.

Bits 23:21 **LPMRCNT:[2:0]** LPM retry count

When the device gives an ERROR response, this is the number of additional LPM retries that the host performs until a valid device response (STALL, NYET, or ACK) is received.

Note: Accessible only in host mode.

Bits 20:17 **LPMCHIDX[3:0]:** LPM Channel Index

The channel number on which the LPM transaction has to be applied while sending an LPM transaction to the local device. Based on the LPM channel index, the core automatically inserts the device address and endpoint number programmed in the corresponding channel into the LPM transaction.

Note: Accessible only in host mode.

Bit 16 **L1RSMOK:** Sleep state resume OK

Indicates that the device or host can start resume from Sleep state. This bit is valid in LPM sleep (L1) state. It is set in sleep mode after a delay of 50 µs ($T_{L1Residency}$).

This bit is reset when SLPSTS is 0.

1: The application or host can start resume from Sleep state

0: The application or host cannot start resume from Sleep state

Bit 15 **SLPSTS:** Port sleep status

Device mode:

This bit is set as long as a Sleep condition is present on the USB bus. The core enters the Sleep state when an ACK response is sent to an LPM transaction and the $T_{L1TokenRetry}$ timer has expired. To stop the PHY clock, the application must set the STPPCLK bit in OTG_PCGCCTL, which asserts the PHY suspend input signal.

The application must rely on SLPSTS and not ACK in LPMRSP to confirm transition into sleep.

The core comes out of sleep:

- When there is any activity on the USB linestate
- When the application writes to the RWUSIG bit in OTG_DCTL or when the application resets or soft-disconnects the device.

Host mode:

The host transitions to Sleep (L1) state as a side-effect of a successful LPM transaction by the core to the local port with ACK response from the device. The read value of this bit reflects the current Sleep status of the port.

The core clears this bit after:

- The core detects a remote L1 wakeup signal,
- The application sets the PRST bit or the PRES bit in the OTG_HPRT register, or
- The application sets the L1Resume/ remote wakeup detected interrupt bit or disconnect detected interrupt bit in the core interrupt register (WKUPINT or DISCINT bit in OTG_GINTSTS, respectively).

0: Core not in L1

1: Core in L1

Bits 14:13 **LPMRST[1:0]:** LPM response

Device mode:

The response of the core to LPM transaction received is reflected in these two bits.

Host mode:

Handshake response received from local device for LPM transaction

11: ACK

10: NYET

01: STALL

00: ERROR (No handshake response)

Bit 12 **L1DSEN:** L1 deep sleep enable

Enables suspending the PHY in L1 Sleep mode. For maximum power saving during L1 Sleep mode, this bit should be set to '1' by application SW in all the cases.

Bits11:8 **BESLTHRS[3:0]:** BESL threshold

Device mode:

The core puts the PHY into deep low power mode in L1 when BESL value is greater than or equal to the value defined in this field BESL_Thres[3:0].

Host mode:

The core puts the PHY into deep low power mode in L1. BESLTHRS[3:0] specifies the time for which resume signaling is to be reflected by host ($T_{L1HubDrvResume2}$) on the USB bus when it detects device initiated resume.

BESLTHRS must not be programmed with a value greater than 1100b in host mode, because this exceeds maximum $T_{L1HubDrvResume2}$.

Thres[3:0] host mode resume signaling time (μs):

0000: 75

0001: 100

0010: 150

0011: 250

0100: 350

0101: 450

0110: 950

All other values: reserved

Bit 7 **L1SEN:** L1 Shallow Sleep enable

Enables suspending the PHY in L1 Sleep mode. For maximum power saving during L1 Sleep mode, this bit should be set to '1' by application SW in all the cases.

Bit 6 **REMWAKE:** bRemoteWake value

Host mode:

The value of remote wake up to be sent in the wIndex field of LPM transaction.

Device mode (read-only):

This field is updated with the received LPM token bRemoteWake bmAttribute when an ACK, NYET, or STALL response is sent to an LPM transaction.

Bits 5:2 **BESL[3:0]**: Best effort service latency

Host mode:

The value of BESL to be sent in an LPM transaction. This value is also used to initiate resume for a duration $T_{L1HubDrvResume1}$ for host initiated resume.

Device mode (read-only):

This field is updated with the received LPM token BESL bmAttribute when an ACK, NYET, or STALL response is sent to an LPM transaction.

BESL[3:0] T_{BESL} (μ s)

0000: 125

0001: 150

0010: 200

0011: 300

0100: 400

0101: 500

0110: 1000

0111: 2000

1000: 3000

1001: 4000

1010: 5000

1011: 6000

1100: 7000

1101: 8000

1110: 9000

1111: 10000

Bit 1 **LPMACK**: LPM token acknowledge enable

Handshake response to LPM token preprogrammed by device application software.

1: ACK

Even though ACK is preprogrammed, the core device responds with ACK only on successful LPM transaction. The LPM transaction is successful if:

- No PID/CRC5 errors in either EXT token or LPM token (else ERROR)
- Valid bLinkState = 0001B (L1) received in LPM transaction (else STALL)
- No data pending in transmit queue (else NYET).

0: NYET

The preprogrammed software bit is over-ridden for response to LPM token when:

- The received bLinkState is not L1 (STALL response), or
- An error is detected in either of the LPM token packets because of corruption (ERROR response).

Note: Accessible only in device mode.

Bit 0 **LPMEN**: LPM support enable

The application uses this bit to control the OTG_FS/OTG_HS core LPM capabilities.

If the core operates as a non-LPM-capable host, it cannot request the connected device or hub to activate LPM mode.

If the core operates as a non-LPM-capable device, it cannot respond to any LPM transactions.

0: LPM capability is not enabled

1: LPM capability is enabled

32.15.15 OTG host periodic transmit FIFO size register (OTG_HPTXFSIZ)

Address offset: 0x100

Reset value: 0x0200 0400

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PTXFSIZ[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
PTXSA[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **PTXFSIZ[15:0]**: Host periodic Tx FIFO depth

This value is in terms of 32-bit words.

Minimum value is 16

Bits 15:0 **PTXSA[15:0]**: Host periodic Tx FIFO start address

This field configures the memory start address for periodic transmit FIFO RAM.

32.15.16 OTG device IN endpoint transmit FIFO size register (OTG_DIEPTXF x) ($x = 1..5[\text{FS}] / 8[\text{HS}]$, where x is the FIFO number)

Address offset: 0x104 + ($x - 1$) * 0x04

Reset value: 0x0200 0200 + ($x * 0x200$)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
INEPTXFD[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
INEPTXSA[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **INEPTXFD[15:0]**: IN endpoint Tx FIFO depth

This value is in terms of 32-bit words.

Minimum value is 16

Bits 15:0 **INEPTXSA[15:0]**: IN endpoint FIFO x transmit RAM start address

This field contains the memory start address for IN endpoint transmit FIFO x . The address must be aligned with a 32-bit memory location.

32.15.17 Host-mode registers

Bit values in the register descriptions are expressed in binary unless otherwise specified.

Host-mode registers affect the operation of the core in the host mode. Host mode registers must not be accessed in device mode, as the results are undefined. Host mode registers can be categorized as follows:

32.15.18 OTG host configuration register (OTG_HCFG)

Address offset: 0x400

Reset value: 0x0000 0000

This register configures the core after power-on. Do not make changes to this register after initializing the host.

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

Bits 31:3 Reserved, must be kept at reset value.

Bit 2 **FSLSS**: FS- and LS-only support

The application uses this bit to control the core's enumeration speed. Using this bit, the application can make the core enumerate as an FS host, even if the connected device supports HS traffic. Do not make changes to this field after initial programming.

1: FS/LS-only, even if the connected device can support HS (read-only).

Bits 1:0 **FSLSPCS[1:0]**: FS/LS PHY clock select

When the core is in FS host mode

01: PHY clock is running at 48 MHz

Others: Reserved

When the core is in LS host mode

00: Reserved

01: Select 48 MHz PHY clock frequency

10: Select 6 MHz PHY clock frequency

11: Reserved

Note: The FSLSPCS must be set on a connection event according to the speed of the connected device (after changing this bit, a software reset must be performed).

32.15.19 OTG host frame interval register (OTG_HFIR)

Address offset: 0x404

Reset value: 0x0000 EA60

This register stores the frame interval information for the current speed to which the OTG_FS/OTG_HS controller has enumerated.

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

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

Bit 16 **RLDCTRL**: Reload control

This bit allows dynamic reloading of the HFIR register during run time.

0: The HFIR can be dynamically reloaded during run time.

1: The HFIR cannot be reloaded dynamically

This bit needs to be programmed during initial configuration and its value must not be changed during run time.

Caution: RLDCTRL = 1 is not recommended.

Bits 15:0 **FRIVL[15:0]**: Frame interval for USB OTG FS

The value that the application programs to this field, specifies the interval between two consecutive SOFs (FS) or Keep-Alive tokens (LS). This field contains the number of PHY clocks that constitute the required frame interval. The application can write a value to this register only after the port enable bit of the host port control and status register (PENA bit in OTG_HPRT) has been set. If no value is programmed, the core calculates the value based on the PHY clock specified in the FS/LS PHY clock select field of the host configuration register (FSLSPCS in OTG_HCFG). Do not change the value of this field after the initial configuration, unless the RLDCTRL bit is set. In such case, the FRIVL is reloaded with each SOF event.

– Frame interval = 1 ms × (FRIVL - 1)

Bits 15:0 **FRIVL[15:0]**: Frame interval for USB OTG HS

The value that the application programs to this field, specifies the interval between two consecutive micro-SOFs (HS) or Keep-Alive tokens (LS). This field contains the number of PHY clocks that constitute the required frame interval. The application can write a value to this register only after the port enable bit of the host port control and status register (PENA bit in OTG_HPRT) has been set. If no value is programmed, the core calculates the value based on the PHY clock specified in the FS/LS PHY clock select field of the host configuration register (FSLSPCS in OTG_HCFG). Do not change the value of this field after the initial configuration, unless the RLDCTRL bit is set. In such case, the FRIVL is reloaded with each SOF event.

– Frame interval = 125 µs × (FRIVL - 1) in high speed operation (PHYSEL = 0)

– Frame interval = 1 ms × (FRIVL - 1) in low/full speed operation (PHYSEL = 1)

32.15.20 OTG host frame number/frame time remaining register (OTG_HFNUM)

Address offset: 0x408

Reset value: 0x0000 3FFF

This register indicates the current frame number. It also indicates the time remaining (in terms of the number of PHY clocks) in the current frame.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FTREM[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
FRNUM[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 **FTREM[15:0]**: Frame time remaining

Indicates the amount of time remaining in the current frame, in terms of PHY clocks. This field decrements on each PHY clock. When it reaches zero, this field is reloaded with the value in the Frame interval register and a new SOF is transmitted on the USB.

Bits 15:0 **FRNUM[15:0]**: Frame number

This field increments when a new SOF is transmitted on the USB, and is cleared to 0 when it reaches 0x3FFF.

32.15.21 OTG_Host periodic transmit FIFO/queue status register (OTG_HPTXSTS)

Address offset: 0x410

Reset value: 0x0008 0100

This read-only register contains the free space information for the periodic Tx FIFO and the periodic transmit request queue.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PTXQTOP[7:0]								PTXQSAV[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
PTXFSAVL[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:24 **PTXQTOP[7:0]**: Top of the periodic transmit request queue

This indicates the entry in the periodic Tx request queue that is currently being processed by the MAC.

This register is used for debugging.

Bit 31: Odd/Even frame

0: send in even frame

1: send in odd frame

Bits 30:27: Channel/endpoint number

Bits 26:25: Type

00: IN/OUT

01: Zero-length packet

11: Disable channel command

Bit 24: Terminate (last entry for the selected channel/endpoint)

Bits 23:16 **PTXQSAV[7:0]**: Periodic transmit request queue space available

Indicates the number of free locations available to be written in the periodic transmit request queue. This queue holds both IN and OUT requests.

00: Periodic transmit request queue is full

01: 1 location available

10: 2 locations available

b_n: n locations available (0 ≤ n ≤ 8)

Others: Reserved

Bits 15:0 **PTXFSAVL[15:0]**: Periodic transmit data FIFO space available

Indicates the number of free locations available to be written to in the periodic Tx FIFO.

Values are in terms of 32-bit words

0000: Periodic Tx FIFO is full

0001: 1 word available

0010: 2 words available

b_n: n words available (where 0 ≤ n ≤ PTXFD)

Others: Reserved

32.15.22 OTG host all channels interrupt register (OTG_HAIT)

Address offset: 0x414

Reset value: 0x0000 0000

When a significant event occurs on a channel, the host all channels interrupt register interrupts the application using the host channels interrupt bit of the core interrupt register (HCINT bit in OTG_GINTSTS). This is shown in [Figure 436](#). There is one interrupt bit per channel, up to a maximum of 16 bits. Bits in this register are set and cleared when the application sets and clears bits in the corresponding host channel-x interrupt register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
HAIN[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

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

Bits 15:0 **HAINT[15:0]**: Channel interrupts

One bit per channel: Bit 0 for Channel 0, bit 15 for Channel 15

32.15.23 OTG host all channels interrupt mask register (OTG_HAINTMSK)

Address offset: 0x418

Reset value: 0x0000 0000

The host all channel interrupt mask register works with the host all channel interrupt register to interrupt the application when an event occurs on a channel. There is one interrupt mask bit per channel, up to a maximum of 16 bits.

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

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

Bits 15:0 **HAINTM[15:0]**: Channel interrupt mask

0: Masked interrupt

1: Unmasked interrupt

One bit per channel: Bit 0 for channel 0, bit 15 for channel 15

32.15.24 OTG host port control and status register (OTG_HPRT)

Address offset: 0x440

Reset value: 0x0000 0000

This register is available only in host mode. Currently, the OTG host supports only one port.

A single register holds USB port-related information such as USB reset, enable, suspend, resume, connect status, and test mode for each port. It is shown in [Figure 436](#). The rc_w1 bits in this register can trigger an interrupt to the application through the host port interrupt bit of the core interrupt register (HPRTINT bit in OTG_GINTSTS). On a port interrupt, the application must read this register and clear the bit that caused the interrupt. For the rc_w1 bits, the application must write a 1 to the bit to clear the interrupt.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PSPD[1:0]	PTCTL [3]	
													r	r	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PTCTL[2:0]		PPWR	PLSTS[1:0]		Res.	PRST	PSUSP	PRES	POC CHNG	POCA	PEN CHNG	PENA	PCDET	PCSTS	
rw	rw	rw	rw	r	r	rw	rs	rw	rc_w1	r	rc_w1	rc_w1	rc_w1	rc_w1	r

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

Bits 18:17 **PSPD[1:0]: Port speed**

Indicates the speed of the device attached to this port.

- 01: Full speed
- 10: Low speed
- 11: Reserved
- 00: High speed

Bits 16:13 **PTCTL[3:0]: Port test control**

The application writes a nonzero value to this field to put the port into a Test mode, and the corresponding pattern is signaled on the port.

- 0000: Test mode disabled
- 0001: Test_J mode
- 0010: Test_K mode
- 0011: Test_SE0_NAK mode
- 0100: Test_Packet mode
- 0101: Test_Force_Enable
- Others: Reserved

Bit 12 **PPWR: Port power**

The application uses this field to control power to this port, and the core clears this bit on an overcurrent condition.

- 0: Power off
- 1: Power on

Bits 11:10 **PLSTS[1:0]: Port line status**

Indicates the current logic level USB data lines

- Bit 10: Logic level of OTG_DP
- Bit 11: Logic level of OTG_DM

Bit 9 Reserved, must be kept at reset value.

Bit 8 PRST: Port reset

When the application sets this bit, a reset sequence is started on this port. The application must time the reset period and clear this bit after the reset sequence is complete.

0: Port not in reset

1: Port in reset

The application must leave this bit set for a minimum duration of at least 10 ms to start a reset on the port. The application can leave it set for another 10 ms in addition to the required minimum duration, before clearing the bit, even though there is no maximum limit set by the USB standard.

High speed: 50 ms

Full speed/Low speed: 10 ms

Bit 7 PSUSP: Port suspend

The application sets this bit to put this port in suspend mode. The core only stops sending SOFs when this is set. To stop the PHY clock, the application must set the port clock stop bit, which asserts the suspend input pin of the PHY.

The read value of this bit reflects the current suspend status of the port. This bit is cleared by the core after a remote wakeup signal is detected or the application sets the port reset bit or port resume bit in this register or the resume/remote wakeup detected interrupt bit or disconnect detected interrupt bit in the core interrupt register (WKUPINT or DISCINT in OTG_GINTSTS, respectively).

0: Port not in suspend mode

1: Port in suspend mode

Bit 6 PRES: Port resume

The application sets this bit to drive resume signaling on the port. The core continues to drive the resume signal until the application clears this bit.

If the core detects a USB remote wakeup sequence, as indicated by the port resume/remote wakeup detected interrupt bit of the core interrupt register (WKUPINT bit in OTG_GINTSTS), the core starts driving resume signaling without application intervention and clears this bit when it detects a disconnect condition. The read value of this bit indicates whether the core is currently driving resume signaling.

0: No resume driven

1: Resume driven

When LPM is enabled and the core is in L1 state, the behavior of this bit is as follow:

1. The application sets this bit to drive resume signaling on the port.
2. The core continues to drive the resume signal until a predetermined time specified in BESLTHRS[3:0] field of OTG_GLMCFG register.
3. If the core detects a USB remote wakeup sequence, as indicated by the port L1Resume/Remote L1Wakeup detected interrupt bit of the core interrupt register (WKUPINT in OTG_GINTSTS), the core starts driving resume signaling without application intervention and clears this bit at the end of resume. This bit can be set or cleared by both the core and the application. This bit is cleared by the core even if there is no device connected to the host.

Bit 5 POCCHNG: Port overcurrent change

The core sets this bit when the status of the port overcurrent active bit (bit 4) in this register changes.

Bit 4 POCA: Port overcurrent active

Indicates the overcurrent condition of the port.

0: No overcurrent condition

1: Overcurrent condition

Bit 3 PENCHNG: Port enable/disable change

The core sets this bit when the status of the port enable bit 2 in this register changes.

Bit 2 **PENA:** Port enable

A port is enabled only by the core after a reset sequence, and is disabled by an overcurrent condition, a disconnect condition, or by the application clearing this bit. The application cannot set this bit by a register write. It can only clear it to disable the port. This bit does not trigger any interrupt to the application.

0: Port disabled

1: Port enabled

Bit 1 **PCDET:** Port connect detected

The core sets this bit when a device connection is detected to trigger an interrupt to the application using the host port interrupt bit in the core interrupt register (HPRTINT bit in OTG_GINTSTS). The application must write a 1 to this bit to clear the interrupt.

Bit 0 **PCSTS:** Port connect status

0: No device is attached to the port

1: A device is attached to the port

32.15.25 OTG host channel x characteristics register (OTG_HCCHARx) ($x = 0..15[\text{HS}] / 11[\text{FS}]$, where $x = \text{Channel number}$)

Address offset: 0x500 + ($x * 0x20$)

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CHENA	CHDIS	ODDFRM	DAD[6:0]				MCNT[1:0]				EPTYP[1:0]		LSDEV	Res.	
rs	rs	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
EPDIR	EPNUM[3:0]				MPSIZ[10:0]										
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **CHENA:** Channel enable

This field is set by the application and cleared by the OTG host.

0: Channel disabled

1: Channel enabled

Bit 30 **CHDIS:** Channel disable

The application sets this bit to stop transmitting/receiving data on a channel, even before the transfer for that channel is complete. The application must wait for the Channel disabled interrupt before treating the channel as disabled.

Bit 29 **ODDFRM:** Odd frame

This field is set (reset) by the application to indicate that the OTG host must perform a transfer in an odd frame. This field is applicable for only periodic (isochronous and interrupt) transactions.

0: Even frame

1: Odd frame

Bits 28:22 **DAD[6:0]:** Device address

This field selects the specific device serving as the data source or sink.

Bits 21:20 **MCNT[1:0]: Multicount**

This field indicates to the host the number of transactions that must be executed per frame for this periodic endpoint. For non-periodic transfers, this field is not used

00: Reserved. This field yields undefined results

01: 1 transaction

10: 2 transactions per frame to be issued for this endpoint

11: 3 transactions per frame to be issued for this endpoint

Note: This field must be set to at least 01.

Bits 19:18 **EPTYP[1:0]: Endpoint type**

Indicates the transfer type selected.

00: Control

01: Isochronous

10: Bulk

11: Interrupt

Bit 17 **LSDEV: Low-speed device**

This field is set by the application to indicate that this channel is communicating to a low-speed device.

Bit 16 Reserved, must be kept at reset value.

Bit 15 **EPDIR: Endpoint direction**

Indicates whether the transaction is IN or OUT.

0: OUT

1: IN

Bits 14:11 **EPNUM[3:0]: Endpoint number**

Indicates the endpoint number on the device serving as the data source or sink.

Bits 10:0 **MPSIZ[10:0]: Maximum packet size**

Indicates the maximum packet size of the associated endpoint.

32.15.26 OTG host channel x split control register (OTG_HCSPLTx) (x = 0..15, where x = Channel number)

Address offset: 0x504 + (x * 0x20)

Reset value: 0x0000 0000

Note: Configuration register applies only to USB OTG HS.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
SPLIT EN	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	COMP LSPLT	
rw																rw	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
XACTPOS[1:0]	HUBADDR[6:0]								PRTADDR[6:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit 31 **SPLITEN**: Split enable

The application sets this bit to indicate that this channel is enabled to perform split transactions.

Bits 30:17 Reserved, must be kept at reset value.

Bit 16 **COMPLSPLT**: Do complete split

The application sets this bit to request the OTG host to perform a complete split transaction.

Bits 15:14 **XACTPOS[1:0]**: Transaction position

This field is used to determine whether to send all, first, middle, or last payloads with each OUT transaction.

11: All. This is the entire data payload of this transaction (which is less than or equal to 188 bytes)

10: Begin. This is the first data payload of this transaction (which is larger than 188 bytes)

00: Mid. This is the middle payload of this transaction (which is larger than 188 bytes)

01: End. This is the last payload of this transaction (which is larger than 188 bytes)

Bits 13:7 **HUBADDR[6:0]**: Hub address

This field holds the device address of the transaction translator's hub.

Bits 6:0 **PRTADDR[6:0]**: Port address

This field is the port number of the recipient transaction translator.

32.15.27 OTG host channel x interrupt register (OTG_HCINT x) ($x = 0..15[\text{HS}] / 11[\text{FS}]$, where $x = \text{Channel number}$)

Address offset: 0x508 + ($x * 0x20$)

Reset value: 0x0000 0000

This register indicates the status of a channel with respect to USB- and AHB-related events. It is shown in [Figure 436](#). The application must read this register when the host channels interrupt bit in the core interrupt register (HCINT bit in OTG_GINTSTS) is set. Before the application can read this register, it must first read the host all channels interrupt (OTG_HAINT) register to get the exact channel number for the host channel- x interrupt register. The application must clear the appropriate bit in this register to clear the corresponding bits in the OTG_HAINT and OTG_GINTSTS registers.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	DTERR	FRM OR	BBERR	TXERR	Res.	ACK	NAK	STALL	Res.	CHH	XFRC
					rc_w1	rc_w1	rc_w1	rc_w1		rc_w1	rc_w1	rc_w1		rc_w1	rc_w1

Note: Configuration register for USB OTG FS.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	DTERR	FRM OR	BBERR	TXERR	NYET	ACK	NAK	STALL	AHB ERR	CHH	XFRC
					rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Note: Configuration register for USB OTG HS.

Bits 31:11 Reserved, must be kept at reset value.

Bit 10 **DTERR**: Data toggle error.

Bit 9 **FRMOR**: Frame overrun.

Bit 8 **BBERR**: Babble error.

Bit 7 **TXERR**: Transaction error.

Indicates one of the following errors occurred on the USB.

CRC check failure

Timeout

Bit stuff error

False EOP

Bit 6 Reserved, must be kept at reset value for USB OTG FS.

Bit 6 **NYET**: Not yet ready response received interrupt for USB OTG HS.

Bit 5 **ACK**: ACK response received/transmitted interrupt.

Bit 4 **NAK**: NAK response received interrupt.

Bit 3 **STALL**: STALL response received interrupt.

Bit 2 Reserved, must be kept at reset value for USB OTG FS.

Bit 2 **AHBERR**: AHB error for USB OTG HS

This error is generated only in Internal DMA mode when an AHB error occurs during an AHB read/write operation. The application can read the corresponding DMA channel address register to get the error address.

Bit 1 **CHH**: Channel halted.

Indicates the transfer completed abnormally either because of any USB transaction error or in response to disable request by the application.

Bit 0 **XFRC**: Transfer completed.

Transfer completed normally without any errors.

32.15.28 OTG host channel x interrupt mask register (OTG_HCINTMSKx) (x = 0..15[HS] / 11[FS], where x = Channel number)

Address offset: 0x50C + (x * 0x20)

Reset value: 0x0000 0000

This register reflects the mask for each channel status described in the previous section.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	DTERRM	FRMORM	BBERRM	TXERRM	Res.	ACKM	NAKM	STALLM	Res.	CHHM	XFRCM
					rw	rw	rw	rw		rw	rw	rw		rw	rw

Note: Configuration register for USB OTG FS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	DTERRM	FRMORM	BBERRM	TXERRM	NYET	ACKM	NAKM	STALLM	AHBERRM	CHHM	XFRCM
					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Note: Configuration register for USB OTG HS

Bits 31:11 Reserved, must be kept at reset value.

Bit 10 **DTERRM**: Data toggle error mask.

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 9 **FRMORM**: Frame overrun mask.

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 8 **BBERRM**: Babble error mask.

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 7 **TXERRM**: Transaction error mask.

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 6 Reserved, must be kept at reset value for USB OTG FS.

Bit 6 **NYET**: response received interrupt mask for USB OTG HS.

- 0: Masked interrupt
- 1: Unmasked interrupt

- Bit 5 **ACKM**: ACK response received/transmitted interrupt mask.
 0: Masked interrupt
 1: Unmasked interrupt
- Bit 4 **NAKM**: NAK response received interrupt mask.
 0: Masked interrupt
 1: Unmasked interrupt
- Bit 3 **STALLM**: STALL response received interrupt mask.
 0: Masked interrupt
 1: Unmasked interrupt
- Bit 2 **AHBERRM**: AHB error for USB OTG HS.
 0: Masked interrupt
 1: Unmasked interrupt
- Bit 2 Reserved, must be kept at reset value for USB OTG FS.
- Bit 1 **CHHM**: Channel halted mask
 0: Masked interrupt
 1: Unmasked interrupt
- Bit 0 **XFRCM**: Transfer completed mask
 0: Masked interrupt
 1: Unmasked interrupt

32.15.29 OTG host channel x transfer size register (OTG_HCTSIZx) (x = 0..15[HS] / 11[FS], where x = Channel number)

Address offset: 0x510 + (x * 0x20)

Reset value: 0x0000 0000

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

Bit 31 Reserved, must be kept at reset value.

Bits 30:29 **DPID[1:0]: Data PID**

The application programs this field with the type of PID to use for the initial transaction. The host maintains this field for the rest of the transfer.

00: DATA0

01: DATA2

10: DATA1

11: SETUP (control) / reserved[FS]MDATA[HS] (non-control)

Bits 28:19 **PKTCNT[9:0]: Packet count**

This field is programmed by the application with the expected number of packets to be transmitted (OUT) or received (IN).

The host decrements this count on every successful transmission or reception of an OUT/IN packet. Once this count reaches zero, the application is interrupted to indicate normal completion.

Bits 18:0 **XFRSIZ[18:0]: Transfer size**

For an OUT, this field is the number of data bytes the host sends during the transfer.

For an IN, this field is the buffer size that the application has reserved for the transfer. The application is expected to program this field as an integer multiple of the maximum packet size for IN transactions (periodic and non-periodic).

32.15.30 OTG host channel x DMA address register (OTG_HCDMAx) ($x = 0..15$, where x = Channel number)

Address offset: 0x514 + ($x * 0x20$)

Reset value: 0x0000 0000

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

Bits 31:0 **DMAADDR[31:0]: DMA address**

This field holds the start address in the external memory from which the data for the endpoint must be fetched or to which it must be stored. This register is incremented on every AHB transaction.

32.15.31 Device-mode registers

These registers must be programmed every time the core changes to device mode

32.15.32 OTG device configuration register (OTG_DCFG)

Address offset: 0x800

Reset value: 0x0220 0000

This register configures the core in device mode after power-on or after certain control commands or enumeration. Do not make changes to this register after initial programming.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERRATIM	Res.	Res.	PFIVL[1:0]		DAD[6:0]								Res.	NZLSO HSK	DSPD[1:0]
rw			rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw

Note: Configuration register for USB OTG FS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	PERSCHIVL[1:0]	Res.	Res.							
						rw	rw								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERRATIM	XCVR DLY	Res.	PFIVL[1:0]		DAD[6:0]								Res.	NZLSO HSK	DSPD[1:0]
rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw

Note: Configuration register for USB OTG HS

Bits 31:16 Reserved, must be kept at reset value for USB OTG FS.

Bits 31:26 Reserved, must be kept at reset value for USB OTG HS.

Bits 25:24 PERSCHIVL[1:0]: Periodic schedule interval for USB OTG HS

This field specifies the amount of time the Internal DMA engine must allocate for fetching periodic IN endpoint data. Based on the number of periodic endpoints, this value must be specified as 25, 50 or 75% of the (micro) frame.

- When any periodic endpoints are active, the internal DMA engine allocates the specified amount of time in fetching periodic IN endpoint data
- When no periodic endpoint is active, then the internal DMA engine services nonperiodic endpoints, ignoring this field
- After the specified time within a (micro) frame, the DMA switches to fetching nonperiodic endpoints

00: 25% of (micro)frame

01: 50% of (micro)frame

10: 75% of (micro)frame

11: Reserved

Bits 23:16 Reserved, must be kept at reset value for USB OTG HS.

Bit 15 ERRATIM: Erratic error interrupt mask

1: Mask early suspend interrupt on erratic error

0: Early suspend interrupt is generated on erratic error

Bit 14 **XCVRDLY:** Transceiver delay

Enables or disables delay in ULPI timing during device chirp.

0: Disable delay (use default timing)

1: Enable delay to default timing, necessary for some ULPI PHYs

Bit 13 Reserved, must be kept at reset value.

Bits 12:11 **PFIVL[1:0]:** Periodic frame interval

Indicates the time within a frame at which the application must be notified using the end of periodic frame interrupt. This can be used to determine if all the isochronous traffic for that frame is complete.

00: 80% of the frame interval

01: 85% of the frame interval

10: 90% of the frame interval

11: 95% of the frame interval

Bits 10:4 **DAD[6:0]:** Device address

The application must program this field after every SetAddress control command.

Bit 3 Reserved, must be kept at reset value.

Bit 2 **NZLSOHSK:** Non-zero-length status OUT handshake

The application can use this field to select the handshake the core sends on receiving a nonzero-length data packet during the OUT transaction of a control transfer's status stage.

1:Send a STALL handshake on a nonzero-length status OUT transaction and do not send the received OUT packet to the application.

0:Send the received OUT packet to the application (zero-length or nonzero-length) and send a handshake based on the NAK and STALL bits for the endpoint in the device endpoint control register.

Bits 1:0 **DSPD[1:0]:** Device speed

Indicates the speed at which the application requires the core to enumerate, or the maximum speed the application can support. However, the actual bus speed is determined only after the chirp sequence is completed, and is based on the speed of the USB host to which the core is connected.

00: Reserved

01: Reserved

10: Reserved

11: Full speed (USB 1.1 transceiver clock is 48 MHz)

Bits 1:0 **DSPD[1:0]:** Device speed

Indicates the speed at which the application requires the core to enumerate, or the maximum speed the application can support. However, the actual bus speed is determined only after the chirp sequence is completed, and is based on the speed of the USB host to which the core is connected.

00: High speed

01: Full speed using HS

10: Reserved

11: Full speed using internal FS PHY

32.15.33 OTG device control register (OTG_DCTL)

Address offset: 0x804

Reset value: 0x0000 0002

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DS BESL RJCT	Res.	Res.
													rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	PO PRGDNE	CGO NAK	SGO NAK	CGI NAK	SGI NAK	TCTL[2:0]	GON STS	GIN STS	SDIS	RWU SIG		
				rw	w	w	w	w	rw	rw	rw	r	rw	rw	rw

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

Bit 18 **DSBESLRJCT**: Deep sleep BESL reject

Core rejects LPM request with BESL value greater than BESL threshold programmed.
NYET response is sent for LPM tokens with BESL value greater than BESL threshold. By default, the deep sleep BESL reject feature is disabled.

Bits 17:12 Reserved, must be kept at reset value.

Bit 11 **POPRGDNE**: Power-on programming done

The application uses this bit to indicate that register programming is completed after a wakeup from power down mode.

Bit 10 **CGONAK**: Clear global OUT NAK

Writing 1 to this field clears the Global OUT NAK.

Bit 9 **SGONAK**: Set global OUT NAK

Writing 1 to this field sets the Global OUT NAK.
The application uses this bit to send a NAK handshake on all OUT endpoints.
The application must set this bit only after making sure that the Global OUT NAK effective bit in the core interrupt register (GONAKEFF bit in OTG_GINTSTS) is cleared.

Bit 8 **CGINAK**: Clear global IN NAK

Writing 1 to this field clears the Global IN NAK.

Bit 7 **SGINAK**: Set global IN NAK

Writing 1 to this field sets the Global non-periodic IN NAK. The application uses this bit to send a NAK handshake on all non-periodic IN endpoints.
The application must set this bit only after making sure that the Global IN NAK effective bit in the core interrupt register (GINAKEFF bit in OTG_GINTSTS) is cleared.

Bits 6:4 **TCTL[2:0]**: Test control

- 000: Test mode disabled
- 001: Test_J mode
- 010: Test_K mode
- 011: Test_SE0_NAK mode
- 100: Test_Packet mode
- 101: Test_Force_Enable
- Others: Reserved

Bit 3 **GONSTS**: Global OUT NAK status

0:A handshake is sent based on the FIFO status and the NAK and STALL bit settings.
 1:No data is written to the Rx FIFO, irrespective of space availability. Sends a NAK handshake on all packets, except on SETUP transactions. All isochronous OUT packets are dropped.

Bit 2 **GINSTS**: Global IN NAK status

0:A handshake is sent out based on the data availability in the transmit FIFO.
 1:A NAK handshake is sent out on all non-periodic IN endpoints, irrespective of the data availability in the transmit FIFO.

Bit 1 **SDIS**: Soft disconnect

The application uses this bit to signal the USB OTG core to perform a soft disconnect. As long as this bit is set, the host does not see that the device is connected, and the device does not receive signals on the USB. The core stays in the disconnected state until the application clears this bit.

0:Normal operation. When this bit is cleared after a soft disconnect, the core generates a device connect event to the USB host. When the device is reconnected, the USB host restarts device enumeration.

1:The core generates a device disconnect event to the USB host.

Bit 0 **RWUSIG**: Remote wakeup signaling

When the application sets this bit, the core initiates remote signaling to wake up the USB host. The application must set this bit to instruct the core to exit the suspend state. As specified in the USB 2.0 specification, the application must clear this bit 1 ms to 15 ms after setting it.

If LPM is enabled and the core is in the L1 (sleep) state, when the application sets this bit, the core initiates L1 remote signaling to wake up the USB host. The application must set this bit to instruct the core to exit the sleep state. As specified in the LPM specification, the hardware automatically clears this bit 50 μ s ($T_{L1DevDrvResume}$) after being set by the application. The application must not set this bit when bRemoteWake from the previous LPM transaction is zero (refer to REMWAKE bit in GLPMCFG register).

Table 234 contains the minimum duration (according to device state) for which the Soft disconnect (SDIS) bit must be set for the USB host to detect a device disconnect. To accommodate clock jitter, it is recommended that the application add some extra delay to the specified minimum duration.

Table 234. Minimum duration for soft disconnect

Operating speed	Device state	Minimum duration
Full speed	Suspended	1 ms + 2.5 μ s
Full speed	Idle	2.5 μ s
Full speed	Not Idle or suspended (Performing transactions)	2.5 μ s
High speed	Not Idle or suspended (Performing transactions)	125 μ s

32.15.34 OTG device status register (OTG_DSTS)

Address offset: 0x808

Reset value: 0x0000 0010

This register indicates the status of the core with respect to USB-related events. It must be read on interrupts from the device all interrupts (OTG_DAINT) register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DEVLNSTS[1:0]		FNSOF[13:8]					
								r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FNSOF[7:0]								Res.	Res.	Res.	Res.	EERR	ENUMSPD[1:0]	SUSP STS	
r	r	r	r	r	r	r	r					r	r	r	r

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

Bits 23:22 **DEVLNSTS[1:0]: Device line status**

Indicates the current logic level USB data lines.

Bit [23]: Logic level of D+

Bit [22]: Logic level of D-

Bits 21:8 **FNSOF[13:0]: Frame number of the received SOF**

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

Bit 3 **EERR: Erratic error**

The core sets this bit to report any erratic errors.

Due to erratic errors, the OTG_FS/OTG_HS controller goes into suspended state and an interrupt is generated to the application with Early suspend bit of the OTG_GINTSTS register (ESUSP bit in OTG_GINTSTS). If the early suspend is asserted due to an erratic error, the application can only perform a soft disconnect recover.

Bits 2:1 **ENUMSPD[1:0]: Enumerated speed**

Indicates the speed at which the OTG_FS/OTG_HS controller has come up after speed detection through a chirp sequence.

01: Reserved

10: Reserved

11: Full speed (PHY clock is running at 48 MHz)

Others: reserved

Bit 0 **SUSPSTS: Suspend status**

In device mode, this bit is set as long as a suspend condition is detected on the USB. The core enters the suspended state when there is no activity on the USB data lines for a period of 3 ms. The core comes out of the suspend:

- When there is an activity on the USB data lines
- When the application writes to the remote wakeup signaling bit in the OTG_DCTL register (RWUSIG bit in OTG_DCTL).

32.15.35 OTG device IN endpoint common interrupt mask register (OTG_DIEPMSK)

Address offset: 0x810

Reset value: 0x0000 0000

This register works with each of the OTG_DIEPINTx registers for all endpoints to generate an interrupt per IN endpoint. The IN endpoint interrupt for a specific status in the OTG_DIEPINTx register can be masked by writing to the corresponding bit in this register. Status bits are masked by default.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	NAKM	Res.	Res.	Res.	Res.	TXFURM	Res.	INEPNEM	INEPNMM	ITTXFE MSK	TOM	Res.	EPDM	XFRCM
		rw					rw		rw	rw	rw	rw		rw	rw

Note: Configuration register for USB OTG FS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	NAKM	Res.	Res.	Res.	Res.	TXFURM	Res.	INEPNEM	INEPNMM	ITTXFE MSK	TOM	AHBERRM	EPDM	XFRCM
		rw					rw		rw	rw	rw	rw	rw	rw	rw

Note: Configuration register for USB OTG HS

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

Bit 13 **NAKM:** NAK interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bits 12:10 Reserved, must be kept at reset value.

Bit 9 Reserved, must be kept at reset value.

Bit 8 **TXFURM:** FIFO underrun mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 7 Reserved, must be kept at reset value.

Bit 6 **INEPNEM:** IN endpoint NAK effective mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 5 **INEPNMM:** IN token received with EP mismatch mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 4 **ITTXFEMSK**: IN token received when Tx FIFO empty mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 3 **TOM**: Timeout condition mask (Non-isochronous endpoints)

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 2 **AHBERRM**: AHB error mask for USB OTG HS

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 1 **EPDM**: Endpoint disabled interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 0 **XFRCM**: Transfer completed interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

32.15.36 OTG device OUT endpoint common interrupt mask register (OTG_DOEPMASK)

Address offset: 0x814

Reset value: 0x0000 0000

This register works with each of the OTG_DOEPINTx registers for all endpoints to generate an interrupt per OUT endpoint. The OUT endpoint interrupt for a specific status in the OTG_DOEPINTx register can be masked by writing into the corresponding bit in this register. Status bits are masked by default.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	NYET MSK	NAK MSK	BERR M	Res.	Res.	Res.	OUT PKT ERRM	Res.	Res.	STS PHSR XM	OTEPD M	STUPM	Res.	EPDM	XFRCM
rw	rw	rw	rw				rw			rw	rw	rw		rw	rw

Note: Configuration register for USB OTG FS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	NYET MSK	NAK MSK	BERR M	Res.	Res.	Res.	OUT PKT ERRM	Res.	B2B STUPM	STS PHSR XM	OTEPD M	STUPM	AHB ERRM	EPDM	XFRCM
rw	rw	rw	rw				rw		rw	rw	rw	rw	rw	rw	rw

Note: Configuration register for USB OTG HS

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

Bit 14 **NYETMSK**: NYET interrupt mask for USB OTG HS

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 13 **NAKMSK**: NAK interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 12 **BERRM**: Babble error interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bits 11:10 Reserved, must be kept at reset value.

Bit 9 Reserved, must be kept at reset value.

Bit 8 **OUTPKTERRM**: Out packet error mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 7 Reserved, must be kept at reset value.

Bit 6 **B2BSTUPM**: Back-to-back SETUP packets received mask for USB OTG HS

- Applies to control OUT endpoints only.
- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 5 **STSPHSRXM**: Status phase received for control write mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 4 **OTEPDM**: OUT token received when endpoint disabled mask. Applies to control OUT endpoints only.

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 3 **STUPM**: STUPM: SETUP phase done mask. Applies to control endpoints only.

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 2 **AHBERRM**: AHB error mask for USB OTG HS

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 1 **EPDM**: Endpoint disabled interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 0 **XFRCM**: Transfer completed interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

32.15.37 OTG device all endpoints interrupt register (OTG_DAINT)

Address offset: 0x818

Reset value: 0x0000 0000

When a significant event occurs on an endpoint, a OTG_DAINT register interrupts the application using the device OUT endpoints interrupt bit or device IN endpoints interrupt bit of the OTG_GINTSTS register (OEPINT or IEPINT in OTG_GINTSTS, respectively). There is one interrupt bit per endpoint, up to a maximum of 16 bits for OUT endpoints and 16 bits for IN endpoints. For a bidirectional endpoint, the corresponding IN and OUT interrupt bits are used. Bits in this register are set and cleared when the application sets and clears bits in the corresponding device endpoint-x interrupt register (OTG_DIEPINTx/OTG_DOEPINTx).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OEPINT[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
IEPINT[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 **OEPINT[15:0]**: OUT endpoint interrupt bits

One bit per OUT endpoint:

Bit 16 for OUT endpoint 0, bit 19 for OUT endpoint 3.

Bits 15:0 **IEPINT[15:0]**: IN endpoint interrupt bits

One bit per IN endpoint:

Bit 0 for IN endpoint 0, bit 3 for endpoint 3.

32.15.38 OTG all endpoints interrupt mask register (OTG_DAINTMSK)

Address offset: 0x81C

Reset value: 0x0000 0000

The OTG_DAINTMSK register works with the device endpoint interrupt register to interrupt the application when an event occurs on a device endpoint. However, the OTG_DAINT register bit corresponding to that interrupt is still set.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OEPM[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
IEPM[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **OEPM[15:0]**: OUT EP interrupt mask bits

One per OUT endpoint:

Bit 16 for OUT EP 0, bit 19 for OUT EP 3

0: Masked interrupt

1: Unmasked interrupt

Bits 15:0 **IEPM[15:0]**: IN EP interrupt mask bits

One bit per IN endpoint:

Bit 0 for IN EP 0, bit 3 for IN EP 3

0: Masked interrupt

1: Unmasked interrupt

32.15.39 OTG device V_{BUS} discharge time register (OTG_DVBUSDIS)

Address offset: 0x0828

Reset value: 0x0000 17D7

This register specifies the V_{BUS} discharge time after V_{BUS} pulsing during SRP.

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

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

Bits 15:0 **VBUSDT[15:0]**: Device V_{BUS} discharge time

Specifies the V_{BUS} discharge time after V_{BUS} pulsing during SRP. This value equals:

V_{BUS} discharge time in PHY clocks / 1 024

Depending on your V_{BUS} load, this value may need adjusting.

32.15.40 OTG device V_{BUS} pulsing time register (OTG_DVBUSPULSE)

Address offset: 0x082C

Reset value: 0x0000 05B8

This register specifies the V_{BUS} pulsing time during SRP.

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

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

Bits 15:0 **DVBUSP[15:0]**: Device V_{BUS} pulsing time. This feature is only relevant to OTG1.3.

Specifies the V_{BUS} pulsing time during SRP. This value equals:
V_{BUS} pulsing time in PHY clocks / 1 024

32.15.41 OTG device threshold control register (OTG_DTHRCTL)

Address offset: 0x0830

Reset value: 0x0000 0000

Note: Configuration register applies only to USB OTG HS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16			
Res.	Res.	Res.	Res.	ARPEN	Res.	RXTHRLEN[8:0]												RXTHREN
				rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
Res.	Res.	Res.	Res.	Res.	TXTHRLEN[8:0]												ISOTHREN	NONISOTHREN
					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bits 31:28 Reserved, must be kept at reset value.

Bit 27 **ARPEN**: Arbiter parking enable

This bit controls internal DMA arbiter parking for IN endpoints. When thresholding is enabled and this bit is set to one, then the arbiter parks on the IN endpoint for which there is a token received on the USB. This is done to avoid getting into underrun conditions. By default parking is enabled.

Bit 26 Reserved, must be kept at reset value.

Bits 25:17 **RXTHRLEN[8:0]**: Receive threshold length

This field specifies the receive thresholding size in 32-bit words. This field also specifies the amount of data received on the USB before the core can start transmitting on the AHB. The threshold length has to be at least eight 32-bit words. The recommended value for RXTHRLEN is to be the same as the programmed AHB burst length (HBSTLEN bit in OTG_GAHBCFG).

Bit 16 **RXTHREN**: Receive threshold enable

When this bit is set, the core enables thresholding in the receive direction.

Bits 15:11 Reserved, must be kept at reset value.

Bits 10:2 **TXTHRLEN[8:0]**: Transmit threshold length

This field specifies the transmit thresholding size in 32-bit words. This field specifies the amount of data in bytes to be in the corresponding endpoint transmit FIFO, before the core can start transmitting on the USB. The threshold length has to be at least eight 32-bit words. This field controls both isochronous and nonisochronous IN endpoint thresholds. The recommended value for TXTHRLEN is to be the same as the programmed AHB burst length (HBSTLEN bit in OTG_GAHBCFG).

Bit 1 **ISOTHREN**: ISO IN endpoint threshold enable

When this bit is set, the core enables thresholding for isochronous IN endpoints.

Bit 0 **NONISOTHREN**: Nonisochronous IN endpoints threshold enable

When this bit is set, the core enables thresholding for nonisochronous IN endpoints.

32.15.42 OTG device IN endpoint FIFO empty interrupt mask register (OTG_DIEPEMPMSK)

Address offset: 0x834

Reset value: 0x0000 0000

This register is used to control the IN endpoint FIFO empty interrupt generation (TXFE_OTG_DIEPINTx).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
INEPTXFEM[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 15:0 **INEPTXFEM[15:0]**: IN EP Tx FIFO empty interrupt mask bits

These bits act as mask bits for OTG_DIEPINTx.

TXFE interrupt one bit per IN endpoint:

Bit 0 for IN endpoint 0, bit 3 for IN endpoint 3

0: Masked interrupt

1: Unmasked interrupt

32.15.43 OTG device each endpoint interrupt register (OTG_DEACHINT)

Address offset: 0x0838

Reset value: 0x0000 0000

Note: Configuration register applies only to USB OTG HS.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	OEP1 INT	Res.													
														r	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	IEP1 INT	Res.													
														r	

Bits 31:18 Reserved, must be kept at reset value.

Bit 17 **OEP1INT:** OUT endpoint 1 interrupt bit

Bits 16:2 Reserved, must be kept at reset value.

Bit 1 **IEP1INT:** IN endpoint 1 interrupt bit

Bit 0 Reserved, must be kept at reset value.

32.15.44 OTG device each endpoint interrupt mask register (OTG_DEACHINTMSK)

Address offset: 0x083C

Reset value: 0x0000 0000

There is one interrupt bit for endpoint 1 IN and one interrupt bit for endpoint 1 OUT.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	OEP1 INTM	Res.													
														rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	IEP1 INTM	Res.													
														rw	

Bits 31:18 Reserved, must be kept at reset value.

Bit 17 **OEP1INTM:** OUT endpoint 1 interrupt mask bit

Bits 16:2 Reserved, must be kept at reset value.

Bit 1 **IEP1INTM:** IN endpoint 1 interrupt mask bit

Bit 0 Reserved, must be kept at reset value.

Note: Configuration register applies only to USB OTG HS

32.15.45 OTG device each IN endpoint-1 interrupt mask register (OTG_HS_DIEPEACHMSK1)

Address offset: 0x844

Reset value: 0x0000 0000

This register works with the OTG_DIEPINT1 register to generate a dedicated interrupt OTG_HS_EP1_IN for endpoint #1. The IN endpoint interrupt for a specific status in the OTG_DOEPINT1 register can be masked by writing into the corresponding bit in this register. Status bits are masked by default.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	NAKM	Res.	Res.	Res.	Res.	TXFURM	Res.	INEPNEM	Res.	ITTXFEMSK	TOM	AHBERRM	EPDM	XFRCM
		rw					rw		rw		rw	rw	rw	rw	rw

Note: Configuration register applies only to USB OTG HS

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

Bit 13 **NAKM**: NAK interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bits 12:10 Reserved, must be kept at reset value.

Bit 9 Reserved, must be kept at reset value.

Bit 8 **TXFURM**: FIFO underrun mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 7 Reserved, must be kept at reset value.

Bit 6 **INEPNEM**: IN endpoint NAK effective mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 5 Reserved, must be kept at reset value.

Bit 4 **ITTXFEMSK**: IN token received when Tx FIFO empty mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 3 **TOM**: Timeout condition mask (Non-isochronous endpoints)

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 2 **AHBERRM:** AHB error mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 1 **EPDM:** Endpoint disabled interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 0 **XFRCM:** Transfer completed interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

32.15.46 OTG device each OUT endpoint-1 interrupt mask register (OTG_HS_DOEPEACHMSK1)

Address offset: 0x884

Reset value: 0x0000 0000

This register works with the OTG_DOEPINT1 register to generate a dedicated interrupt OTG_HS_EP1_OUT for endpoint #1. The OUT endpoint interrupt for a specific status in the OTG_DOEPINT1 register can be masked by writing into the corresponding bit in this register. Status bits are masked by default.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	NYET MSK	NAK MSK	BERR M	Res.	Res.	Res.	OUT PKT ERRM	Res.	B2B STUPM	Res.	OTEPD M	STUPM	AHB ERRM	EPDM	XFRCM
rw	rw	rw					rw		rw		rw	rw	rw	rw	rw

Note: Configuration register applies only to USB OTG HS

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

Bit 14 **NYETMSK:** NYET interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 13 **NAKMSK:** NAK interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 12 **BERRM:** Babble error interrupt mask

- 0: Masked interrupt
- 1: Unmasked interrupt

Bits 11:10 Reserved, must be kept at reset value.

Bit 9 Reserved, must be kept at reset value.

Bit 8 **OUTPKTERRM:** Out packet error mask

- 0: Masked interrupt
- 1: Unmasked interrupt

- Bit 7 Reserved, must be kept at reset value.
- Bit 6 **B2BSTUPM:** Back-to-back SETUP packets received mask
Applies to control OUT endpoints only.
0: Masked interrupt
1: Unmasked interrupt
- Bit 5 Reserved, must be kept at reset value.
- Bit 4 **OTEPDM:** OUT token received when endpoint disabled mask
Applies to control OUT endpoints only.
0: Masked interrupt
1: Unmasked interrupt
- Bit 3 **STUPM:** STUPM: SETUP phase done mask
Applies to control endpoints only.
0: Masked interrupt
1: Unmasked interrupt
- Bit 2 **AHBERRM:** AHB error mask
0: Masked interrupt
1: Unmasked interrupt
- Bit 1 **EPDM:** Endpoint disabled interrupt mask
0: Masked interrupt
1: Unmasked interrupt
- Bit 0 **XFRCM:** Transfer completed interrupt mask
0: Masked interrupt
1: Unmasked interrupt

32.15.47 OTG device control IN endpoint 0 control register (OTG_DIEPCTL0)

Address offset: 0x900

Reset value: 0x0000 0000

This section describes the OTG_DIEPCTL0 register for USB_OTG FS. Nonzero control endpoints use registers for endpoints 1–3.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EPENA	EPDIS	Res.	Res.	SNAK	CNAK	TXFNUM[3:0]				STALL	Res.	EPTYP		NAK STS	Res.
rs	rs			w	w	rw	rw	rw	rw	rs		r	r	r	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USBA EP	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	MPSIZ[1:0]	
r														rw	rw

Bit 31 EPENA: Endpoint enable

The application sets this bit to start transmitting data on the endpoint 0.

The core clears this bit before setting any of the following interrupts on this endpoint:

- Endpoint disabled
- Transfer completed

Bit 30 EPDIS: Endpoint disable

The application sets this bit to stop transmitting data on an endpoint, even before the transfer for that endpoint is complete. The application must wait for the endpoint disabled interrupt before treating the endpoint as disabled. The core clears this bit before setting the endpoint disabled interrupt. The application must set this bit only if endpoint enable is already set for this endpoint.

Bits 29:28 Reserved, must be kept at reset value.

Bit 27 SNAK: Set NAK

A write to this bit sets the NAK bit for the endpoint.

Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The core can also set this bit for an endpoint after a SETUP packet is received on that endpoint.

Bit 26 CNAK: Clear NAK

A write to this bit clears the NAK bit for the endpoint.

Bits 25:22 TXFNUM[3:0]: Tx FIFO number

This value is set to the FIFO number that is assigned to IN endpoint 0.

Bit 21 STALL: STALL handshake

The application can only set this bit, and the core clears it when a SETUP token is received for this endpoint. If a NAK bit, a Global IN NAK or Global OUT NAK is set along with this bit, the STALL bit takes priority.

Bit 20 Reserved, must be kept at reset value.

Bits 19:18 EPTYP: Endpoint type

Hardcoded to '00' for control.

Bit 17 NAKSTS: NAK status

Indicates the following:

0: The core is transmitting non-NAK handshakes based on the FIFO status

1: The core is transmitting NAK handshakes on this endpoint.

When this bit is set, either by the application or core, the core stops transmitting data, even if there are data available in the Tx FIFO. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.

Bit 16 Reserved, must be kept at reset value.

Bit 15 **USBAEP**: USB active endpoint

This bit is always set to 1, indicating that control endpoint 0 is always active in all configurations and interfaces.

Bits 14:2 Reserved, must be kept at reset value.

Bits 1:0 **MPSIZ[1:0]**: Maximum packet size

The application must program this field with the maximum packet size for the current logical endpoint.

00: 64 bytes

01: 32 bytes

10: 16 bytes

11: 8 bytes

Note: Configuration register applies only to USB OTG FS

32.15.48 OTG device IN endpoint x control register (OTG_DIEPCTLx) (x = 1..5[FS] / 0..8[HS], where x = endpoint number)

Address offset: 0x900 + (x * 0x20)

Reset value: 0x0000 0000

The application uses this register to control the behavior of each logical endpoint other than endpoint 0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EPENA	EPDIS	SODDFRM	SD0 PID/ SEVN FIRM	SNAK	CNAK	TXFNUM[3:0]				STALL	Res.	EPTYP[1:0]		NAK STS	EO NUM/ DPID
rs	rs	w	w	w	w	rw	rw	rw	rw	rw/rs		rw	rw	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USBAEP	Res.	Res.	Res.	Res.	MPSIZ[10:0]										
rw					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **EPENA**: Endpoint enable

The application sets this bit to start transmitting data on an endpoint.

The core clears this bit before setting any of the following interrupts on this endpoint:

- SETUP phase done
- Endpoint disabled
- Transfer completed

Bit 30 **EPDIS**: Endpoint disable

The application sets this bit to stop transmitting/receiving data on an endpoint, even before the transfer for that endpoint is complete. The application must wait for the endpoint disabled interrupt before treating the endpoint as disabled. The core clears this bit before setting the endpoint disabled interrupt. The application must set this bit only if endpoint enable is already set for this endpoint.

Bit 29 **SODDFRM**: Set odd frame

Applies to isochronous IN and OUT endpoints only.

Writing to this field sets the Even/Odd frame (EONUM) field to odd frame.

Bit 28 **SD0PID:** Set DATA0 PID

Applies to interrupt/bulk IN endpoints only.

Writing to this field sets the endpoint data PID (DPID) field in this register to DATA0.

SEVNFRM: Set even frame

Applies to isochronous IN endpoints only.

Writing to this field sets the Even/Odd frame (EONUM) field to even frame.

Bit 27 **SNAK:** Set NAK

A write to this bit sets the NAK bit for the endpoint.

Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The core can also set this bit for OUT endpoints on a transfer completed interrupt, or after a SETUP is received on the endpoint.

Bit 26 **CNAK:** Clear NAK

A write to this bit clears the NAK bit for the endpoint.

Bits 25:22 **TXFNUM:** Tx FIFO number

These bits specify the FIFO number associated with this endpoint. Each active IN endpoint must be programmed to a separate FIFO number.

This field is valid only for IN endpoints.

Bit 21 **STALL:** STALL handshake

Applies to non-control, non-isochronous IN endpoints only (access type is rw).

The application sets this bit to stall all tokens from the USB host to this endpoint. If a NAK bit, Global IN NAK, or Global OUT NAK is set along with this bit, the STALL bit takes priority. Only the application can clear this bit, never the core.

Applies to control endpoints only (access type is rs).

The application can only set this bit, and the core clears it, when a SETUP token is received for this endpoint. If a NAK bit, Global IN NAK, or Global OUT NAK is set along with this bit, the STALL bit takes priority. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.

Bit 20 Reserved, must be kept at reset value.

Bits 19:18 **EPTYP[1:0]:** Endpoint type

This is the transfer type supported by this logical endpoint.

00: Control

01: Isochronous

10: Bulk

11: Interrupt

Bit 17 **NAKSTS:** NAK status

It indicates the following:

0: The core is transmitting non-NAK handshakes based on the FIFO status.

1: The core is transmitting NAK handshakes on this endpoint.

When either the application or the core sets this bit:

For non-isochronous IN endpoints: The core stops transmitting any data on an IN endpoint, even if there are data available in the Tx FIFO.

For isochronous IN endpoints: The core sends out a zero-length data packet, even if there are data available in the Tx FIFO.

Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.

Bit 16 **EONUM:** Even/odd frame

Applies to isochronous IN endpoints only.

Indicates the frame number in which the core transmits/receives isochronous data for this endpoint. The application must program the even/odd frame number in which it intends to transmit/receive isochronous data for this endpoint using the SEVNFRM and SODDFRM fields in this register.

0: Even frame

1: Odd frame

DPID: Endpoint data PID

Applies to interrupt/bulk IN endpoints only.

Contains the PID of the packet to be received or transmitted on this endpoint. The application must program the PID of the first packet to be received or transmitted on this endpoint, after the endpoint is activated. The application uses the SD0PID register field to program either DATA0 or DATA1 PID.

0: DATA0

1: DATA1

Bit 15 **USBAEP:** USB active endpoint

Indicates whether this endpoint is active in the current configuration and interface. The core clears this bit for all endpoints (other than EP 0) after detecting a USB reset. After receiving the SetConfiguration and SetInterface commands, the application must program endpoint registers accordingly and set this bit.

Bits 14:11 Reserved, must be kept at reset value.

Bits 10:0 **MPSIZ[10:0]:** Maximum packet size

The application must program this field with the maximum packet size for the current logical endpoint. This value is in bytes.

32.15.49 OTG device IN endpoint x interrupt register (OTG_DIEPINTx) (x = 0..5[FS] /8[HS], where x = Endpoint number)

Address offset: 0x908 + (x * 0x20)

Reset value: 0x0000 0080

This register indicates the status of an endpoint with respect to USB- and AHB-related events. It is shown in [Figure 436](#). The application must read this register when the IN endpoints interrupt bit of the core interrupt register (IEPINT in OTG_GINTSTS) is set. Before the application can read this register, it must first read the device all endpoints interrupt (OTG_DAINT) register to get the exact endpoint number for the device endpoint-x interrupt register. The application must clear the appropriate bit in this register to clear the corresponding bits in the OTG_DAINT and OTG_GINTSTS registers.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	NAK	Res.	PKTD RPSTS	Res.	Res.	TXFIF OUD RN	TXFE	IN EPNE	IN EPNM	ITTXFE	TOC	Res.	EP DISD	XFRFC
		rc_w1		rc_w1			rc_w1	r	r	rc_w1	rc_w1	rc_w1		rc_w1	rc_w1

Note: Configuration register for USB OTG FS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	NAK	Res.	PKTD RPSTS	Res.	Res.	TXFIF OUD RN	TXFE	IN EPNE	IN EPNM	ITTXFE	TOC	AHB ERR	EP DISD	XFRC
		rc_w1		rc_w1			rc_w1	r	r	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Note: Configuration register for USB OTG HS

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

Bit 13 **NAK:** NAK input

The core generates this interrupt when a NAK is transmitted or received by the device. In case of isochronous IN endpoints the interrupt gets generated when a zero length packet is transmitted due to unavailability of data in the Tx FIFO.

Bit 12 Reserved, must be kept at reset value.

Bit 11 **PKTDRPSTS:** Packet dropped status

This bit indicates to the application that an ISOC OUT packet has been dropped. This bit does not have an associated mask bit and does not generate an interrupt.

Bit 10 Reserved, must be kept at reset value.

Bit 9 Reserved, must be kept at reset value.

Bit 8 **TXFIFOUDRN:** Transmit Fifo Underrun (TxfifoUndrn)

The core generates this interrupt when it detects a transmit FIFO underrun condition for this endpoint. Dependency: This interrupt is valid only when Thresholding is enabled

Bit 7 **TXFE:** Transmit FIFO empty

This interrupt is asserted when the Tx FIFO for this endpoint is either half or completely empty. The half or completely empty status is determined by the Tx FIFO Empty Level bit in the OTG_GAHBCFG register (TXFELVL bit in OTG_GAHBCFG).

Bit 6 **INEPNE:** IN endpoint NAK effective

This bit can be cleared when the application clears the IN endpoint NAK by writing to the CNAK bit in OTG_DIEPCTLx.

This interrupt indicates that the core has sampled the NAK bit set (either by the application or by the core). The interrupt indicates that the IN endpoint NAK bit set by the application has taken effect in the core.

This interrupt does not guarantee that a NAK handshake is sent on the USB. A STALL bit takes priority over a NAK bit.

Bit 5 **INEPNM:** IN token received with EP mismatch

Indicates that the data in the top of the non-periodic TxFIFO belongs to an endpoint other than the one for which the IN token was received. This interrupt is asserted on the endpoint for which the IN token was received.

Bit 4 **ITTXFE:** IN token received when Tx FIFO is empty

Indicates that an IN token was received when the associated Tx FIFO (periodic/non-periodic) was empty. This interrupt is asserted on the endpoint for which the IN token was received.

Bit 3 **TOC**: Timeout condition

Indicates that the core has detected a timeout condition on the USB for the last IN token on this endpoint.

Bit 2 **AHBERR**: AHB error for USB OTG HS

This is generated only in internal DMA mode when there is an AHB error during an AHB read/write. The application can read the corresponding endpoint DMA address register to get the error address.

Bit 1 **EPDISD**: Endpoint disabled interrupt

This bit indicates that the endpoint is disabled per the application's request.

Bit 0 **XFRS**: Transfer completed interrupt

This field indicates that the programmed transfer is complete on the AHB as well as on the USB, for this endpoint.

32.15.50 OTG device IN endpoint 0 transfer size register (OTG_DIEPTSIZE0)

Address offset: 0x910

Reset value: 0x0000 0000

The application must modify this register before enabling endpoint 0. Once endpoint 0 is enabled using the endpoint enable bit in the device control endpoint 0 control registers (EPENA in OTG_DIEPCTL0), the core modifies this register. The application can only read this register once the core has cleared the endpoint enable bit.

Nonzero endpoints use the registers for endpoints 1–3.

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

Bits 31:21 Reserved, must be kept at reset value.

Bits 20:19 **PKTCNT[1:0]**: Packet count

Indicates the total number of USB packets that constitute the transfer size amount of data for endpoint 0.

This field is decremented every time a packet (maximum size or short packet) is read from the Tx FIFO.

Bits 18:7 Reserved, must be kept at reset value.

Bits 6:0 **XFRSIZ[6:0]**: Transfer size

Indicates the transfer size in bytes for endpoint 0. The core interrupts the application only after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet.

The core decrements this field every time a packet from the external memory is written to the Tx FIFO.

32.15.51 OTG device IN endpoint x DMA address register (OTG_DIEPDMA x) ($x = 0..8$, where $x = \text{endpoint number}$)

Address offset: 0x914 + ($x * 0x20$)

Reset value: 0x0000 0000

Note: Configuration register applies only to USB OTG HS

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

Bits 31:0 **DMAADDR[31:0]: DMA Address**

This field holds the start address in the external memory from which the data for the endpoint must be fetched. This register is incremented on every AHB transaction.

32.15.52 OTG device IN endpoint transmit FIFO status register (OTG_DTXFSTS x) ($x = 0..5[\text{FS}] / 8[\text{HS}]$, where $x = \text{endpoint number}$)

Address offset for IN endpoints: 0x918 + ($x * 0x20$) This read-only register contains the free space information for the device IN endpoint Tx FIFO.

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

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

Bits 15:0 **INEPTFSAV[15:0]: IN endpoint Tx FIFO space available**

Indicates the amount of free space available in the endpoint Tx FIFO.

Values are in terms of 32-bit words:

0x0: Endpoint Tx FIFO is full

0x1: 1 word available

0x2: 2 words available

0xn: n words available

Others: Reserved

32.15.53 OTG device IN endpoint x transfer size register (OTG_DIEPTSI x) ($x = 1..5[FS] /8[HS]$, where $x = \text{endpoint number}$)

Address offset: 0x910 + ($x * 0x20$)

Reset value: 0x0000 0000

The application must modify this register before enabling the endpoint. Once the endpoint is enabled using the endpoint enable bit in the OTG_DIEPCTL x registers (EPENA bit in OTG_DIEPCTL x), the core modifies this register. The application can only read this register once the core has cleared the endpoint enable bit.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	MCNT[1:0]		PKTCNT[9:0]												XFRSIZ[18:16]
	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
XFRSIZ[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 Reserved, must be kept at reset value.

Bits 30:29 **MCNT[1:0]: Multi count**

For periodic IN endpoints, this field indicates the number of packets that must be transmitted per frame on the USB. The core uses this field to calculate the data PID for isochronous IN endpoints.

- 01: 1 packet
- 10: 2 packets
- 11: 3 packets

Bits 28:19 **PKTCNT[9:0]: Packet count**

Indicates the total number of USB packets that constitute the transfer size amount of data for this endpoint.

This field is decremented every time a packet (maximum size or short packet) is read from the Tx FIFO.

Bits 18:0 **XFRSIZ[18:0]: Transfer size**

This field contains the transfer size in bytes for the current endpoint. The core only interrupts the application after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet.

The core decrements this field every time a packet from the external memory is written to the Tx FIFO.

32.15.54 OTG device control OUT endpoint 0 control register (OTG_DOEPCTL0)

Address offset: 0xB00

Reset value: 0x0000 8000

This section describes the OTG_DOEPCTL0 register. Nonzero control endpoints use registers for endpoints 1–3.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EPENA	EPDIS	Res.	Res.	SNAK	CNAK	Res.	Res.	Res.	Res.	STALL	SNPM	EPTYP[1:0]	NAK STS	Res.	
w	r			w	w					rs	rw	r	r	r	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USBA EP	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	MPSIZ[1:0]	
r														r	r

Bit 31 **EPENA:** Endpoint enable

The application sets this bit to start transmitting data on endpoint 0.

The core clears this bit before setting any of the following interrupts on this endpoint:

- SETUP phase done
- Endpoint disabled
- Transfer completed

Bit 30 **EPDIS:** Endpoint disable

The application cannot disable control OUT endpoint 0.

Bits 29:28 Reserved, must be kept at reset value.

Bit 27 **SNAK:** Set NAK

A write to this bit sets the NAK bit for the endpoint.

Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The core can also set this bit on a transfer completed interrupt, or after a SETUP is received on the endpoint.

Bit 26 **CNAK:** Clear NAK

A write to this bit clears the NAK bit for the endpoint.

Bits 25:22 Reserved, must be kept at reset value.

Bit 21 **STALL:** STALL handshake

The application can only set this bit, and the core clears it, when a SETUP token is received for this endpoint. If a NAK bit or Global OUT NAK is set along with this bit, the STALL bit takes priority. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.

Bit 20 **SNPM:** Snoop mode

This bit configures the endpoint to Snoop mode. In Snoop mode, the core does not check the correctness of OUT packets before transferring them to application memory.

Bits 19:18 **EPTYP[1:0]:** Endpoint type

Hardcoded to 2'b00 for control.

Bit 17 **NAKSTS:** NAK status

Indicates the following:

0: The core is transmitting non-NAK handshakes based on the FIFO status.

1: The core is transmitting NAK handshakes on this endpoint.

When either the application or the core sets this bit, the core stops receiving data, even if there is space in the Rx FIFO to accommodate the incoming packet. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.

Bit 16 Reserved, must be kept at reset value.

Bit 15 **USBAEP:** USB active endpoint

This bit is always set to 1, indicating that a control endpoint 0 is always active in all configurations and interfaces.

Bits 14:2 Reserved, must be kept at reset value.

Bits 1:0 **MPSIZ[1:0]:** Maximum packet size

The maximum packet size for control OUT endpoint 0 is the same as what is programmed in control IN endpoint 0.

00: 64 bytes

01: 32 bytes

10: 16 bytes

11: 8 bytes

32.15.55 OTG device OUT endpoint x interrupt register (OTG_DOEPINTx) (x = 0..5[FS] /8[HS], where x = Endpoint number)

Address offset: 0xB08 + (x * 0x20)

Reset value: 0x0000 0080

This register indicates the status of an endpoint with respect to USB- and AHB-related events. It is shown in [Figure 436](#). The application must read this register when the OUT endpoints interrupt bit of the OTG_GINTSTS register (OEPINT bit in OTG_GINTSTS) is set. Before the application can read this register, it must first read the OTG_DAINT register to get the exact endpoint number for the OTG_DOEPINTx register. The application must clear the appropriate bit in this register to clear the corresponding bits in the OTG_DAINT and OTG_GINTSTS registers.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	NYET	NAK	BERR	Res.	Res.	Res.	OUT PKT ERR	Res.	Res.	STSPH SRX	OTEPE DIS	STUP	Res.	EP DISD	XFRC
	rc_w1	rc_w1	rc_w1				rc_w1			rc_w1	rc_w1	rc_w1		rc_w1	rc_w1

Note: Configuration register for USB OTG FS.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STPK TRX	NYET	NAK	BERR	Res.	Res.	Res.	OUT PKT ERR	Res.	B2B STUP	STSPH SRX	OTEPE DIS	STUP	AHB ERR	EP DISD	XFRC
rc_w1	rc_w1	rc_w1	rc_w1				rc_w1		rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Note: Configuration register for USB OTG HS.

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

Bit 15 **STPKTRX**: Setup packet received.

Applicable for control OUT endpoints in only in the Buffer DMA Mode. Set by the OTG_HS, this bit indicates that this buffer holds 8 bytes of setup data. There is only one setup packet per buffer. On receiving a setup packet, the OTG_HS closes the buffer and disables the corresponding endpoint after SETUP_COMPLETE status is seen in the Rx FIFO. OTG_HS puts a SETUP_COMPLETE status into the Rx FIFO when it sees the first IN or OUT token after the SETUP packet for that particular endpoint. The application must then re-enable the endpoint to receive any OUT data for the control transfer and reprogram the buffer start address. Because of the above behavior, OTG_HS can receive any number of back to back setup packets and one buffer for every setup packet is used.

Bit 14 **NYET**: NYET interrupt

This interrupt is generated when a NYET response is transmitted for a non isochronous OUT endpoint.

Bit 13 **NAK**: NAK input

The core generates this interrupt when a NAK is transmitted or received by the device. In case of isochronous IN endpoints the interrupt gets generated when a zero length packet is transmitted due to unavailability of data in the Tx FIFO.

Bit 12 **BERR**: Babble error interrupt

The core generates this interrupt when babble is received for the endpoint.

Bits 11:10 Reserved, must be kept at reset value.

Bit 9 Reserved, must be kept at reset value.

Bit 8 **OUTPKTERR**: OUT packet error

This interrupt is asserted when the core detects an overflow or a CRC error for an OUT packet. This interrupt is valid only when thresholding is enabled.

Bit 7 Reserved, must be kept at reset value.

Bit 6 **B2BSTUP**: Back-to-back SETUP packets received for USB OTG HS

Applies to control OUT endpoint only.

This bit indicates that the core has received more than three back-to-back SETUP packets for this particular endpoint.

Bit 5 **STSPHSRX**: Status phase received for control write

This interrupt is valid only for control OUT endpoints. This interrupt is generated only after OTG_FS/OTG_HS has transferred all the data that the host has sent during the data phase of a control write transfer, to the system memory buffer. The interrupt indicates to the application that the host has switched from data phase to the status phase of a control write transfer. The application can use this interrupt to ACK or STALL the status phase, after it has decoded the data phase.

Bit 4 **Otepdis**: OUT token received when endpoint disabled

Applies only to control OUT endpoints.

Indicates that an OUT token was received when the endpoint was not yet enabled. This interrupt is asserted on the endpoint for which the OUT token was received.

Bit 3 **Stup**: SETUP phase done

Applies to control OUT endpoint only.

Indicates that the SETUP phase for the control endpoint is complete and no more back-to-back SETUP packets were received for the current control transfer. On this interrupt, the application can decode the received SETUP data packet.

Bit 2 **Ahberr**: AHB error for USB OTG HS

This is generated only in internal DMA mode when there is an AHB error during an AHB read/write. The application can read the corresponding endpoint DMA address register to get the error address.

Bit 1 **Epdisd**: Endpoint disabled interrupt

This bit indicates that the endpoint is disabled per the application's request.

Bit 0 **Xfrc**: Transfer completed interrupt

This field indicates that the programmed transfer is complete on the AHB as well as on the USB, for this endpoint.

32.15.56 OTG device OUT endpoint 0 transfer size register (OTG_DOEPTSIZ0)

Address offset: 0xB10

Reset value: 0x0000 0000

The application must modify this register before enabling endpoint 0. Once endpoint 0 is enabled using the endpoint enable bit in the OTG_DOEPCTL0 registers (EPENA bit in OTG_DOEPCTL0), the core modifies this register. The application can only read this register once the core has cleared the endpoint enable bit.

Nonzero endpoints use the registers for endpoints 1–5[FS] /8[HS].

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

Bit 31 Reserved, must be kept at reset value.

Bits 30:29 **STUPCNT[1:0]:** SETUP packet count

This field specifies the number of back-to-back SETUP data packets the endpoint can receive.

- 01: 1 packet
- 10: 2 packets
- 11: 3 packets

Bits 28:20 Reserved, must be kept at reset value.

Bit 19 **PKTCNT:** Packet count

This field is decremented to zero after a packet is written into the Rx FIFO.

Bits 18:7 Reserved, must be kept at reset value.

Bits 6:0 **XFRSIZ[6:0]:** Transfer size

Indicates the transfer size in bytes for endpoint 0. The core interrupts the application only after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet.

The core decrements this field every time a packet is read from the Rx FIFO and written to the external memory.

32.15.57 OTG device OUT endpoint x DMA address register (OTG_DOEPDMA x) ($x = 0..8$, where $x = \text{endpoint number}$)

Address offset: 0xB14 + ($x * 0x20$)

Reset value: 0x0000 0000

Note: Configuration register applies only to USB OTG HS

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

Bits 31:0 **DMAADDR[31:0]:** DMA Address

This field holds the start address in the external memory from which the data for the endpoint must be fetched. This register is incremented on every AHB transaction.

32.15.58 OTG device OUT endpoint x control register (OTG_DOEPCTLx) ($x = 1..5[\text{FS}] / 8[\text{HS}]$, where $x = \text{endpoint number}$)

Address offset for OUT endpoints: 0xB00 + ($x * 0x20$)

Reset value: 0x0000 0000

The application uses this register to control the behavior of each logical endpoint other than endpoint 0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EPENA	EPDIS	SD1 PID/ SODD FRM	SD0 PID/ SEVN FRM	SNAK	CNAK	Res.	Res.	Res.	Res.	STALL	SNPM	EPTYP[1:0]	NAK STS	EO NUM/ DPID	
rs	rs	w	w	w	w					rw/rs	rw	rw	rw	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USBA EP	Res.	Res.	Res.	Res.								MPSIZ[10:0]			
rw					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **EPENA:** Endpoint enable

Applies to IN and OUT endpoints.

The application sets this bit to start transmitting data on an endpoint.

The core clears this bit before setting any of the following interrupts on this endpoint:

- SETUP phase done
- Endpoint disabled
- Transfer completed

Bit 30 **EPDIS:** Endpoint disable

The application sets this bit to stop transmitting/receiving data on an endpoint, even before the transfer for that endpoint is complete. The application must wait for the endpoint disabled interrupt before treating the endpoint as disabled. The core clears this bit before setting the endpoint disabled interrupt. The application must set this bit only if endpoint enable is already set for this endpoint.

Bit 29 **SD1PID:** Set DATA1 PID

Applies to interrupt/bulk IN and OUT endpoints only. Writing to this field sets the endpoint data PID (DPID) field in this register to DATA1.

SODDFRM: Set odd frame

Applies to isochronous IN and OUT endpoints only. Writing to this field sets the Even/Odd frame (EONUM) field to odd frame.

Bit 28 **SD0PID:** Set DATA0 PID

Applies to interrupt/bulk OUT endpoints only.

Writing to this field sets the endpoint data PID (DPID) field in this register to DATA0.

SEVNFRM: Set even frame

Applies to isochronous OUT endpoints only.

Writing to this field sets the Even/Odd frame (EONUM) field to even frame.

Bit 27 **SNAK:** Set NAK

A write to this bit sets the NAK bit for the endpoint.

Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The core can also set this bit for OUT endpoints on a transfer completed interrupt, or after a SETUP is received on the endpoint.

Bit 26 CNAK: Clear NAK

A write to this bit clears the NAK bit for the endpoint.

Bits 25:22 Reserved, must be kept at reset value.

Bit 21 STALL: STALL handshake

Applies to non-control, non-isochronous OUT endpoints only (access type is rw).

The application sets this bit to stall all tokens from the USB host to this endpoint. If a NAK bit, Global IN NAK, or Global OUT NAK is set along with this bit, the STALL bit takes priority. Only the application can clear this bit, never the core.

Applies to control endpoints only (access type is rs).

The application can only set this bit, and the core clears it, when a SETUP token is received for this endpoint. If a NAK bit, Global IN NAK, or Global OUT NAK is set along with this bit, the STALL bit takes priority. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.

Bit 20 SNP: Snoop mode

This bit configures the endpoint to Snoop mode. In Snoop mode, the core does not check the correctness of OUT packets before transferring them to application memory.

Bits 19:18 EPTYP[1:0]: Endpoint type

This is the transfer type supported by this logical endpoint.

00: Control

01: Isochronous

10: Bulk

11: Interrupt

Bit 17 NAKSTS: NAK status

Indicates the following:

0: The core is transmitting non-NAK handshakes based on the FIFO status.

1: The core is transmitting NAK handshakes on this endpoint.

When either the application or the core sets this bit:

The core stops receiving any data on an OUT endpoint, even if there is space in the Rx FIFO to accommodate the incoming packet.

Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.

Bit 16 EONUM: Even/odd frame

Applies to isochronous IN and OUT endpoints only.

Indicates the frame number in which the core transmits/receives isochronous data for this endpoint. The application must program the even/odd frame number in which it intends to transmit/receive isochronous data for this endpoint using the SEVNFRM and SODDFRM fields in this register.

0: Even frame

1: Odd frame

DPID: Endpoint data PID

Applies to interrupt/bulk OUT endpoints only.

Contains the PID of the packet to be received or transmitted on this endpoint. The application must program the PID of the first packet to be received or transmitted on this endpoint, after the endpoint is activated. The application uses the SD0PID register field to program either DATA0 or DATA1 PID.

0: DATA0

1: DATA1

Bit 15 **USBAEP: USB active endpoint**

Indicates whether this endpoint is active in the current configuration and interface. The core clears this bit for all endpoints (other than EP 0) after detecting a USB reset. After receiving the SetConfiguration and SetInterface commands, the application must program endpoint registers accordingly and set this bit.

Bits 14:11 Reserved, must be kept at reset value.

Bits 10:0 **MPSIZ[10:0]: Maximum packet size**

The application must program this field with the maximum packet size for the current logical endpoint. This value is in bytes.

32.15.59 OTG device OUT endpoint x transfer size register (OTG_DOEPTSIZx) (x = 1..5[FS] /8[HS]), where x = Endpoint number)

Address offset: 0xB10 + (x * 0x20)

Reset value: 0x0000 0000

The application must modify this register before enabling the endpoint. Once the endpoint is enabled using endpoint enable bit of the OTG_DOEPCTLx registers (EPENA bit in OTG_DOEPCTLx), the core modifies this register. The application can only read this register once the core has cleared the endpoint enable bit.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	RXDPID/ STUPCNT[1:0]		PKTCNT[9:0]												XFRSIZ
15	r/rw	r/rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
XFRSIZ															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 Reserved, must be kept at reset value.

Bits 30:29 **RXDPID[1:0]: Received data PID**

Applies to isochronous OUT endpoints only.

This is the data PID received in the last packet for this endpoint.

00: DATA0

01: DATA2

10: DATA1

11: MDATA

STUPCNT[1:0]: SETUP packet count

Applies to control OUT endpoints only.

This field specifies the number of back-to-back SETUP data packets the endpoint can receive.

01: 1 packet

10: 2 packets

11: 3 packets

Bits 28:19 PKTCNT[9:0]: Packet count

Indicates the total number of USB packets that constitute the transfer size amount of data for this endpoint.

This field is decremented every time a packet (maximum size or short packet) is written to the Rx FIFO.

Bits 18:0 XFRSIZ: Transfer size

This field contains the transfer size in bytes for the current endpoint. The core only interrupts the application after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet.

The core decrements this field every time a packet is read from the Rx FIFO and written to the external memory.

32.15.60 OTG power and clock gating control register (OTG_PCGCCTL)

Address offset: 0xE00

Reset value: 0x200B 8000

This register is available in host and device modes.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SUSP	PHY SLEEP	ENL1 GTG	PHY SUSP	Res.	Res.	GATE HCLK	STPP CLK							
							r	r	rw	r			rw	rw	

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

Bit 7 SUSP: Deep Sleep

This bit indicates that the PHY is in Deep Sleep when in L1 state.

Bit 6 PHYSLEEP: PHY in Sleep

This bit indicates that the PHY is in the Sleep state.

Bit 5 ENL1GTG: Enable sleep clock gating

When this bit is set, core internal clock gating is enabled in Sleep state if the core cannot assert utmi_l1_suspend_n. When this bit is not set, the PHY clock is not gated in Sleep state.

Bit 4 PHYSUSP: PHY suspended

Indicates that the PHY has been suspended. This bit is updated once the PHY is suspended after the application has set the STPPCLK bit.

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

Bit 1 **GATEHCLK**: Gate HCLK

The application sets this bit to gate HCLK to modules other than the AHB Slave and Master and wakeup logic when the USB is suspended or the session is not valid. The application clears this bit when the USB is resumed or a new session starts.

Bit 0 **STPPCLK**: Stop PHY clock

The application sets this bit to stop the PHY clock when the USB is suspended, the session is not valid, or the device is disconnected. The application clears this bit when the USB is resumed or a new session starts.

32.15.61 OTG_FS/OTG_HS register map

The table below gives the USB OTG register map and reset values.

Table 235. OTG_FS/OTG_HS register map and reset values

Table 235. OTG_FS/OTG_HS register map and reset values (continued)

Table 235. OTG_FS/OTG_HS register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
0x01C	OTG_GRXSTSR (host mode)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PKTSTS	DPID	BCNT	CHNUM																							
	Reset value									0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0				
	OTG_GRXSTSR (Device mode)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	FRMNUM	PKTSTS	DPID	BCNT	EPNUM																						
	Reset value					0	STSPHST																													
0x020	OTG_GRXSTSP (host mode)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PKTSTS	DPID	BCNT	CHNUM																							
	Reset value									0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0			
	OTG_GRXSTSP (Device mode)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	FRMNUM	PKTSTS	DPID	BCNT	EPNUM																						
	Reset value					0	STSPHST																													
0x024	OTG_GRXFSIZ	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.				RXFD																							
	Reset value														0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
0x028	OTG_HNPTXFSIZ/ OTG_DIEPTXF0									NPTXFD/TX0FD			NPTXFSA/TX0FSA																							
	Reset value	0	0 0 0 0	0 0 0 1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0						0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0			
0x02C	OTG_HNPTXSTS	Res.								NPTXQTOP			NPTQXSADV																							
	Reset value		0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0						0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0			
0x038	OTG_GCCFG	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	VBDEN	SDEN	PDEN	DCDEN	BCDEN	PWRDWN	Res.	X X X X	PS2DET	SDET	PDET	DCDET															
	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																					
0x03C	OTG_CID									PRODUCT_ID (for USB OTG FS)																										
	Reset value	0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0						0 0 0 0	0 0 0 0	0 0 0 0	1 1 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0		
0x03C	OTG_CID									PRODUCT_ID (for USB OTG HS)																										
	Reset value	0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0						0 0 0 1	1 1 0 0	0 0 0 0	1 0 0 0	0 0 0 0	1 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	
0x054	OTG_GLPMCFG	Res.	Res.	Res.	ENBESL	LPMR CNTSTS	SNDLPM	LPM RCNT	LPMCHIDX	L1RSMOK	SLPSTS	LPM RSP	L1DSEN	BESLTHRS	L1SSEN REM/WAKE	BESL																				
	Reset value			0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0			
0x100	OTG_HPTXFSIZ									PTXFSIZ																										
	Reset value	0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0						0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
0x104	OTG_DIEPTXF1									INEPTXFD																										
	Reset value	0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0						0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

Table 235. OTG_FS/OTG_HS register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x108	OTG_DIEPTXF2																																	
		Reset value	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	
0x114	OTG_DIEPTXF5																																	
		Reset value	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	
0x120	OTG_DIEPTXF7																																	
		Reset value	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	
0x400	OTG_HCFG	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	FSLSS	PCS	0	0	0	
		Reset value																																
0x404	OTG_HFIR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	FRIVL				
		Reset value																																
0x408	OTG_HFNUM																																	
		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	1	1	1	1	1	1	1	1	1
0x410	OTG_HPTXSTS																																	
		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	1	0	1	0	1	1	0	0	0
0x414	OTG_HAINT	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	HAINT					
		Reset value																																
0x418	OTG_HAINTMSK	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	HAINTM					
		Reset value																																
0x440	OTG_HPRT	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PSPD	PTCTL	PPWRR	PLSTS	Res.	PRST	FSUSP	PRES	POCCHNG	POCA	PENCHNG	PENA	FCDET	PCSTS				
		Reset value															0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x500	OTG_HCCHAR0	CHENA	CHDIS	ODDFRM	DAD	MCNT	EP1TYP	LSDEV	EPDIR	EPNUM																			MPSIZ					
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 235. OTG_FS/OTG_HS register map and reset values (continued)

Table 235. OTG_FS/OTG_HS register map and reset values (continued)

Table 235. OTG_FS/OTG_HS register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x6F0	OTG_HCTSIZ15	Res	DPID	PKTCNT					XFRSIZ																								
	Reset value		0 0																														
0x6F4	OTG_HCDMA15		DMAADDR																														
	Reset value		0 0																														
0x800	OTG_DCFG	Res	DEV LN STS														FNSOF																
	Reset value																																
0x804	OTG_DCTL	Res	OEPINT														IEPINT																
	Reset value																																
0x808	OTG_DSTS	Res	OEPM														IEPM																
	Reset value																																
0x810	OTG_DIEPMSK	Res	OETMR														IETMR																
	Reset value																																
0x814	OTG_DOEPMSK	Res	OEPNEM														IEPNE																
	Reset value																																
0x818	OTG_DAINT		OEPINT														IEPINT																
	Reset value	0 0																															
0x81C	OTG_DAINTMSK		OEPM														IEPM																
	Reset value	0 0																															
0x828	OTG_DVBUSSDIS	Res	VBUSDT																														
	Reset value																																
0x82C	OTG_DVB_USPULSE	Res	DVBUSP																														
	Reset value																																

Table 235. OTG_FS/OTG_HS register map and reset values (continued)

Table 235. OTG_FS/OTG_HS register map and reset values (continued)

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x910	OTG_DIEPTSIZ0	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	XFRSIZ			
	Reset value																																
0x914	OTG_DIEPDMA																																
	Reset value	0	0	0	0	0	SODDFRM/SD1IPID	Res																									
0x918	OTG_DTXFSTS0	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	INEPTFSAV			
	Reset value																																
0x920	OTG_DIEPCTL1	EPENA	EPDIS	SODDFRM/SD1IPID	SD0PID/SEVNFRM	SD0PID/SEVNFRM	TXFNUM	CNAK	STALL	EPDIS	MPSIZ																						
	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				
0x928	OTG_DIEPINT1	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	INEPTFSAV			
	Reset value																																
0x930	OTG_DIEPTSIZ1	Res	MCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	PKTCNT	XFRSIZ		
	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				
0x938	OTG_DTXFSTS1	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	INEPTFSAV			
	Reset value																																
0x940	OTG_DIEPCTL2	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	MPSIZ			
	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				
0x9A0	OTG_DIEPCTL5	EPENA	EPDIS	SODDFRM/SD1IPID	SD0PID/SEVNFRM	SD0PID/SEVNFRM	TXFNUM	NAKSTS	MPSIZ																								
	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				

Table 235. OTG_FS/OTG_HS register map and reset values (continued)

Table 235. OTG_FS/OTG_HS register map and reset values (continued)

Table 235. OTG_FS/OTG_HS register map and reset values (continued)

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

32.16 OTG_FS/OTG_HS programming model

32.16.1 Core initialization

The application must perform the core initialization sequence. If the cable is connected during power-up, the current mode of operation bit in the OTG_GINTSTS (CMOD bit in OTG_GINTSTS) reflects the mode. The OTG_FS/OTG_HS controller enters host mode when an “A” plug is connected or device mode when a “B” plug is connected.

This section explains the initialization of the OTG_FS/OTG_HS controller after power-on. The application must follow the initialization sequence irrespective of host or device mode operation. All core global registers are initialized according to the core’s configuration:

1. Program the following fields in the OTG_GAHBCFG register:
 - Global interrupt mask bit GINTMSK = 1
 - Rx FIFO non-empty (RXFLVL bit in OTG_GINTSTS)
 - Periodic Tx FIFO empty level
2. Program the following fields in the OTG_GUSBCFG register:
 - HNP capable bit
 - SRP capable bit
 - OTG_FS/OTG_HS timeout calibration field
 - USB turnaround time field
3. The software must unmask the following bits in the OTG_GINTMSK register:
 - OTG interrupt mask
 - Mode mismatch interrupt mask
4. The software can read the CMOD bit in OTG_GINTSTS to determine whether the OTG_FS/OTG_HS controller is operating in host or device mode.

32.16.2 Host initialization

To initialize the core as host, the application must perform the following steps:

1. Program the HPRTINT in the OTG_GINTMSK register to unmask
2. Program the OTG_HCFG register to select full-speed host
3. Program the PPWR bit in OTG_HPRT to 1. This drives V_{BUS} on the USB.
4. Wait for the PCDET interrupt in OTG_HPRT0. This indicates that a device is connecting to the port.
5. Program the PRST bit in OTG_HPRT to 1. This starts the reset process.
6. Wait at least 10 ms for the reset process to complete.
7. Program the PRST bit in OTG_HPRT to 0.
8. Wait for the PENCHNG interrupt in OTG_HPRT.
9. Read the PSPD bit in OTG_HPRT to get the enumerated speed.
10. Program the HFIR register with a value corresponding to the selected PHY clock 1
11. Program the FSLSPCS field in the OTG_HCFG register following the speed of the device detected in step 9. If FSLSPCS has been changed a port reset must be performed.
12. Program the OTG_GRXFSIZ register to select the size of the receive FIFO.
13. Program the OTG_HNPTXFSIZ register to select the size and the start address of the Non-periodic transmit FIFO for non-periodic transactions.
14. Program the OTG_HPTXFSIZ register to select the size and start address of the periodic transmit FIFO for periodic transactions.

To communicate with devices, the system software must initialize and enable at least one channel.

32.16.3 Device initialization

The application must perform the following steps to initialize the core as a device on power-up or after a mode change from host to device.

1. Program the following fields in the OTG_DCFG register:
 - Device speed
 - Non-zero-length status OUT handshake
2. Program the OTG_GINTMSK register to unmask the following interrupts:
 - USB reset
 - Enumeration done
 - Early suspend
 - USB suspend
 - SOF
3. Wait for the USBRST interrupt in OTG_GINTSTS. It indicates that a reset has been detected on the USB that lasts for about 10 ms on receiving this interrupt.

Wait for the ENUMDNE interrupt in OTG_GINTSTS. This interrupt indicates the end of reset on the USB. On receiving this interrupt, the application must read the OTG_DSTS register to determine the enumeration speed and perform the steps listed in [Endpoint initialization on enumeration completion on page 1332](#).

At this point, the device is ready to accept SOF packets and perform control transfers on control endpoint 0.

32.16.4 DMA mode

The OTG host uses the AHB master interface to fetch the transmit packet data (AHB to USB) and receive the data update (USB to AHB). The AHB master uses the programmed DMA address (OTG_HCDMAx register in host mode and OTG_DIEPDMAx/OTG_DOEPDMAx register in peripheral mode) to access the data buffers.

32.16.5 Host programming model

Channel initialization

The application must initialize one or more channels before it can communicate with connected devices. To initialize and enable a channel, the application must perform the following steps:

1. Program the OTG_GINTMSK register to unmask the following:
 2. Channel interrupt
 - Non-periodic transmit FIFO empty for OUT transactions (applicable when operating in pipelined transaction-level with the packet count field programmed with more than one).
 - Non-periodic transmit FIFO half-empty for OUT transactions (applicable when operating in pipelined transaction-level with the packet count field programmed with more than one).
 3. Program the OTG_HAINTMSK register to unmask the selected channels' interrupts.
 4. Program the OTG_HCINTMSK register to unmask the transaction-related interrupts of interest given in the host channel interrupt register.
 5. Program the selected channel's OTG_HCTSIZx register with the total transfer size, in bytes, and the expected number of packets, including short packets. The application must program the PID field with the initial data PID (to be used on the first OUT transaction or to be expected from the first IN transaction).
 6. Program the OTG_HCCHARx register of the selected channel with the device's endpoint characteristics, such as type, speed, direction, and so forth. (The channel can be enabled by setting the channel enable bit to 1 only when the application is ready to transmit or receive any packet).
 7. Program the selected channels in the OTG_HCSPLTx register(s) with the hub and port addresses (split transactions only).
 8. Program the selected channels in the OTG_HCDMAx register(s) with the buffer start address (DMA transactions only).

Halting a channel

The application can disable any channel by programming the OTG_HCCHARx register with the CHDIS and CHENA bits set to 1. This enables the OTG_FS/OTG_HS host to flush the posted requests (if any) and generates a channel halted interrupt. The application must wait for the CHH interrupt in OTG_HCINTx before reallocating the channel for other transactions. The OTG_FS/OTG_HS host does not interrupt the transaction that has already been started on the USB.

To disable a channel in DMA mode operation, the application does not need to check for space in the request queue. The OTG_HS host checks for space to write the disable

request on the disabled channel's turn during arbitration. Meanwhile, all posted requests are dropped from the request queue when the CHDIS bit in OTG_HCCHARx is set to 1.

Before disabling a channel, the application must ensure that there is at least one free space available in the non-periodic request queue (when disabling a non-periodic channel) or the periodic request queue (when disabling a periodic channel). The application can simply flush the posted requests when the request queue is full (before disabling the channel), by programming the OTG_HCCHARx register with the CHDIS bit set to 1, and the CHENA bit cleared to 0.

The application is expected to disable a channel on any of the following conditions:

1. When an STALL, TXERR, BBERR or DTERR interrupt in OTG_HCINTx is received for an IN or OUT channel. The application must be able to receive other interrupts (DTERR, Nak, data, TXERR) for the same channel before receiving the halt.
2. When an XFRC interrupt in OTG_HCINTx is received during a non periodic IN transfer or high-bandwidth interrupt IN transfer
3. When a DISCINT (disconnect device) interrupt in OTG_GINTSTS is received. (The application is expected to disable all enabled channels).
4. When the application aborts a transfer before normal completion.

Ping protocol

When the OTG_HS host operates in high speed, the application must initiate the ping protocol when communicating with high-speed bulk or control (data and status stage) OUT endpoints. The application must initiate the ping protocol when it receives a NAK/NYET/TXERR interrupt. When the OTG_HS host receives one of the above responses, it does not continue any transaction for a specific endpoint, drops all posted or fetched OUT requests (from the request queue), and flushes the corresponding data (from the transmit FIFO). This is valid in slave mode only. In Slave mode, the application can send a ping token either by setting the DOPING bit in OTG_HCTSIZx before enabling the channel or by just writing the OTG_HCTSIZx register with the DOPING bit set when the channel is already enabled. This enables the OTG_HS host to write a ping request entry to the request queue. The application must wait for the response to the ping token (a NAK, ACK, or TXERR interrupt) before continuing the transaction or sending another ping token. The application can continue the data transaction only after receiving an ACK from the OUT endpoint for the requested ping. In DMA mode operation, the application does not need to set the DOPING bit in OTG_HCTSIZx for a NAK/NYET response in case of bulk/control OUT. The OTG_HS host automatically sets the DOPING bit in OTG_HCTSIZx, and issues the ping tokens for bulk/control OUT. The OTG_HS host continues sending ping tokens until it receives an ACK, and then switches automatically to the data transaction.

Operational model

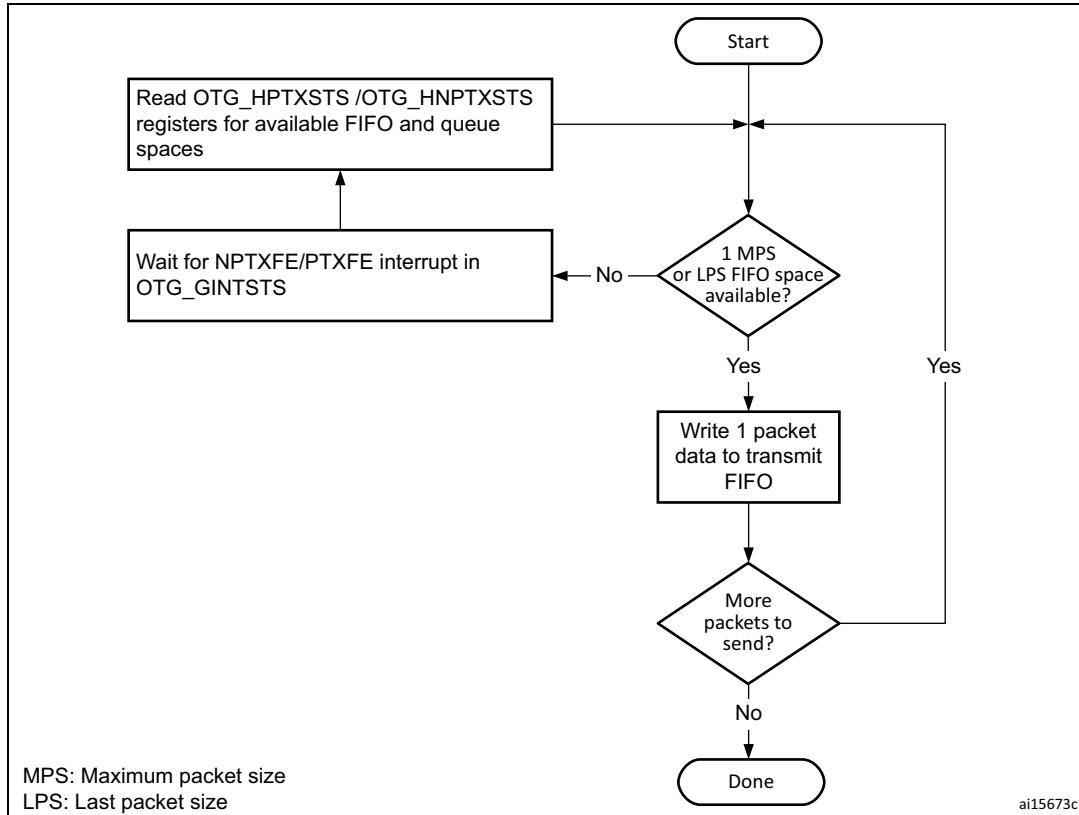
The application must initialize a channel before communicating to the connected device. This section explains the sequence of operation to be performed for different types of USB transactions.

- **Writing the transmit FIFO**

The OTG_FS/OTG_HS host automatically writes an entry (OUT request) to the periodic/non-periodic request queue, along with the last 32-bit word write of a packet. The application must ensure that at least one free space is available in the periodic/non-periodic request queue before starting to write to the transmit FIFO. The application must always write to the transmit FIFO in 32-bit words. If the packet size is non-32-bit word aligned, the application must use padding. The OTG_FS/OTG_HS

host determines the actual packet size based on the programmed maximum packet size and transfer size.

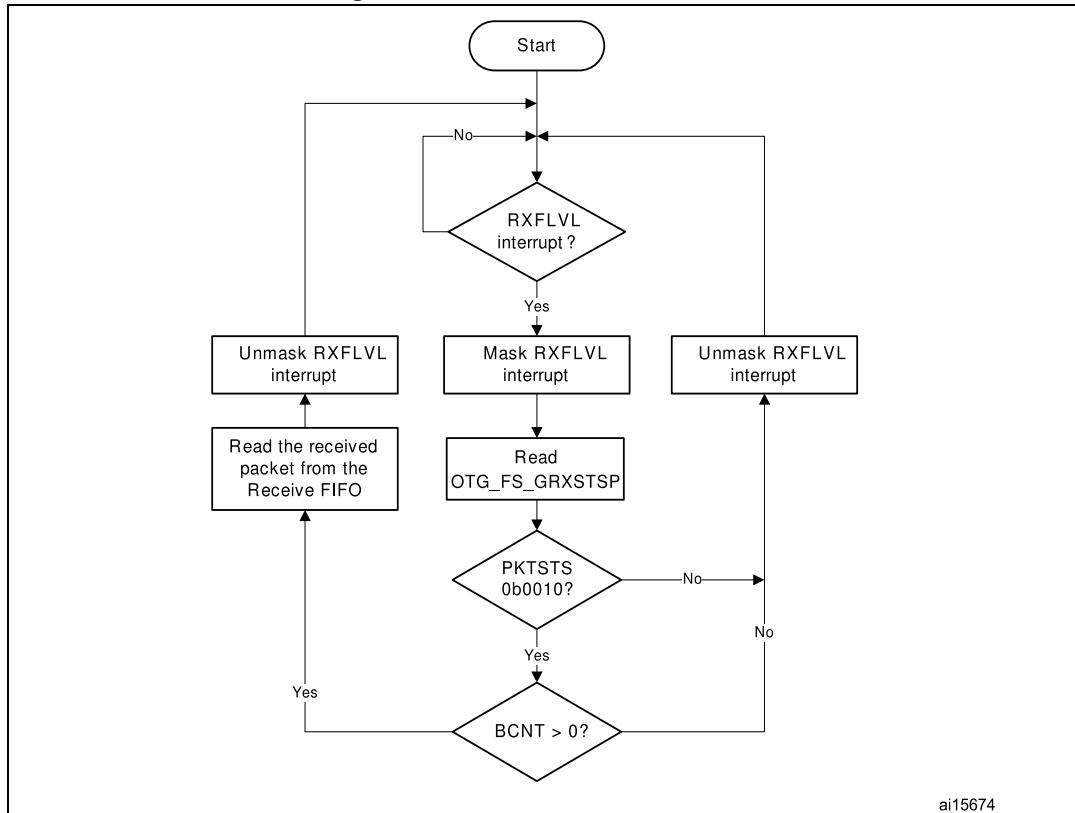
Figure 437. Transmit FIFO write task



- **Reading the receive FIFO**

The application must ignore all packet statuses other than IN data packet (bx0010).

Figure 438. Receive FIFO read task



- **Bulk and control OUT/SETUP transactions**

A typical bulk or control OUT/SETUP pipelined transaction-level operation is shown in [Figure 439](#). See channel 1 (ch_1). Two bulk OUT packets are transmitted. A control SETUP transaction operates in the same way but has only one packet. The assumptions are:

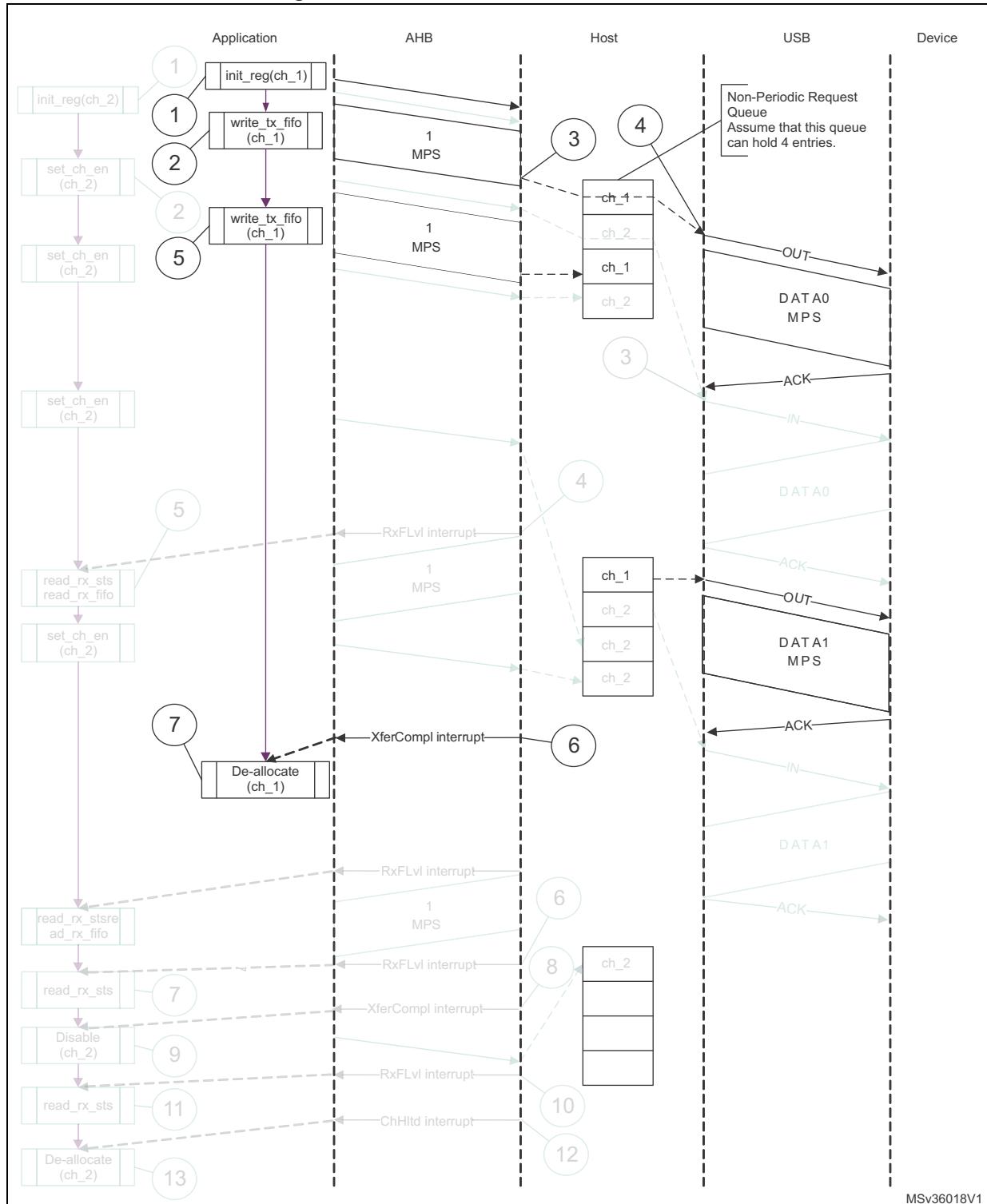
- The application is attempting to send two maximum-packet-size packets (transfer size = 1,024 bytes).
- The non-periodic transmit FIFO can hold two packets (1 KB for HS/128 bytes for FS).
- The non-periodic request queue depth = 4.

- **Normal bulk and control OUT/SETUP operations**

The sequence of operations in (channel 1) is as follows:

1. Initialize channel 1
2. Write the first packet for channel 1
3. Along with the last word write, the core writes an entry to the non-periodic request queue
4. As soon as the non-periodic queue becomes non-empty, the core attempts to send an OUT token in the current frame
5. Write the second (last) packet for channel 1
6. The core generates the XFRC interrupt as soon as the last transaction is completed successfully
7. In response to the XFRC interrupt, de-allocate the channel for other transfers
8. Handling non-ACK responses

Figure 439. Normal bulk/control OUT/SETUP



1. The grayed elements are not relevant in the context of this figure.

The channel-specific interrupt service routine for bulk and control OUT/SETUP transactions is shown in the following code samples.

- **Interrupt service routine for bulk/control OUT/SETUP and bulk/control IN transactions**

- a) Bulk/control OUT/SETUP

```
Unmask (NAK/TXERR/STALL/XFRC)
if (XFRC)
{
    Reset Error Count
    Mask ACK
    De-allocate Channel
}
else if (STALL)
{
    Transfer Done = 1
    Unmask CHH
    Disable Channel
}
else if (NAK or TXERR )
{
    Rewind Buffer Pointers
    Unmask CHH
    Disable Channel
    if (TXERR)
    {
        Increment Error Count
        Unmask ACK
    }
    else
    {
        Reset Error Count
    }
}
else if (CHH)
{
    Mask CHH
    if (Transfer Done or (Error_count == 3))
    {
        De-allocate Channel
    }
    else
    {
        Re-initialize Channel
    }
}
```

```
else if (ACK)
{
    Reset Error Count
    Mask ACK
}

The application is expected to write the data packets into the transmit FIFO when the
space is available in the transmit FIFO and the request queue. The application can
make use of the NPTXFE interrupt in OTG_GINTSTS to find the transmit FIFO space.

b) Bulk/control IN

Unmask (TXERR/XFRC/BBERR/STALL/DTERR)
if (XFRC)
{
    Reset Error Count
    Unmask CHH
    Disable Channel
    Reset Error Count
    Mask ACK
}
else if (TXERR or BBERR or STALL)
{
    Unmask CHH
    Disable Channel
    if (TXERR)
    {
        Increment Error Count
        Unmask ACK
    }
}
else if (CHH)
{
    Mask CHH
    if (Transfer Done or (Error_count == 3))
    {
        De-allocate Channel
    }
    else
    {
        Re-initialize Channel
    }
}
else if (ACK)
{
    Reset Error Count
    Mask ACK
}
```

```
else if (DTERR)
{
    Reset Error Count
}
```

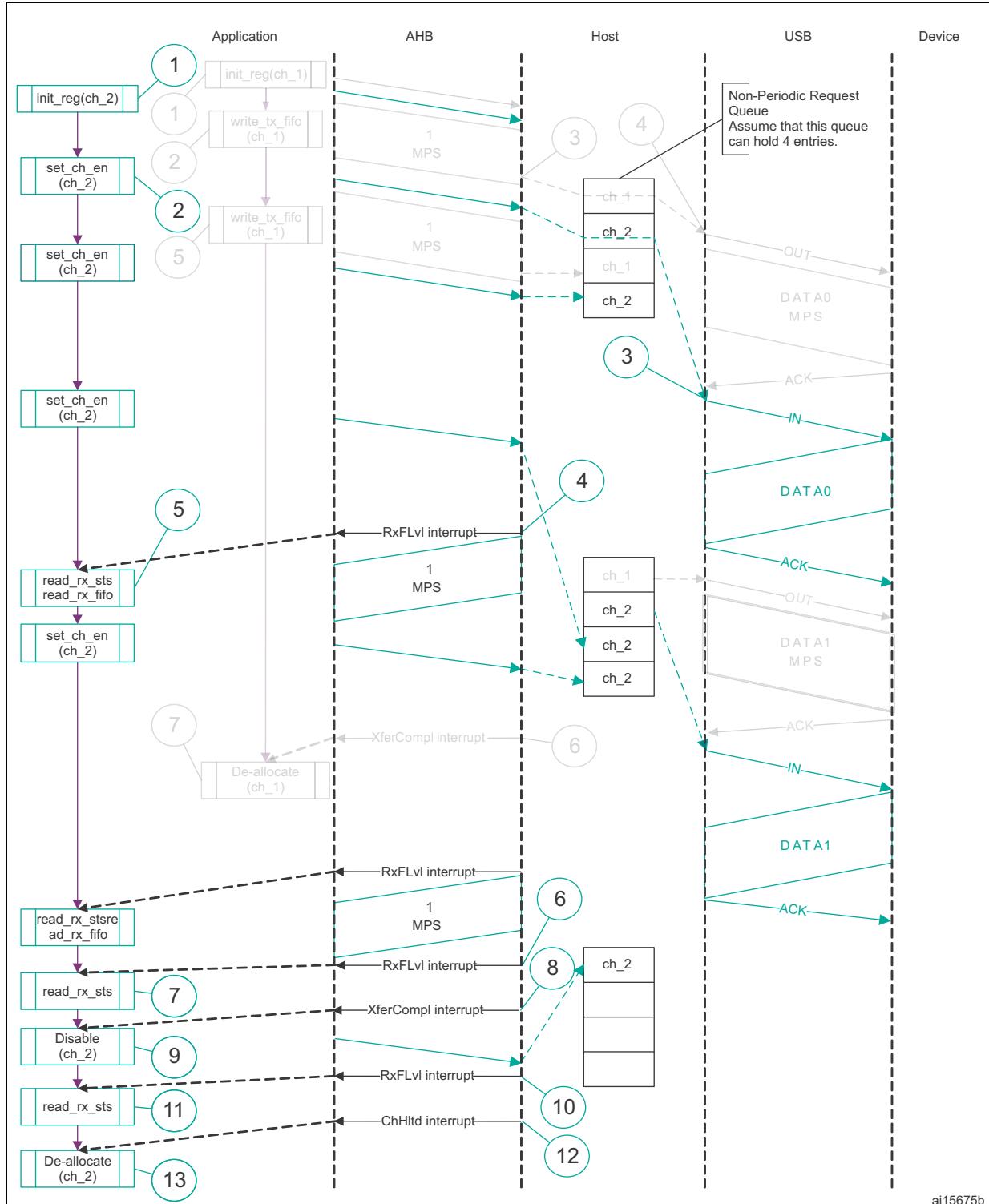
The application is expected to write the requests as and when the request queue space is available and until the XFRC interrupt is received.

- **Bulk and control IN transactions**

A typical bulk or control IN pipelined transaction-level operation is shown in [Figure 440](#). See channel 2 (ch_2). The assumptions are:

- The application is attempting to receive two maximum-packet-size packets (transfer size = 1 024 bytes).
- The receive FIFO can contain at least one maximum-packet-size packet and two status words per packet (72 bytes for FS/520 bytes for HS).
- The non-periodic request queue depth = 4.

Figure 440. Bulk/control IN transactions



1. The grayed elements are not relevant in the context of this figure.

The sequence of operations is as follows:

1. Initialize channel 2.
2. Set the CHENA bit in OTG_HCCHAR2 to write an IN request to the non-periodic request queue.
3. The core attempts to send an IN token after completing the current OUT transaction.
4. The core generates an RXFLVL interrupt as soon as the received packet is written to the receive FIFO.
5. In response to the RXFLVL interrupt, mask the RXFLVL interrupt and read the received packet status to determine the number of bytes received, then read the receive FIFO accordingly. Following this, unmask the RXFLVL interrupt.
6. The core generates the RXFLVL interrupt for the transfer completion status entry in the receive FIFO.
7. The application must read and ignore the receive packet status when the receive packet status is not an IN data packet (PKTSTS in OTG_GRXSTS ≠ 0b0010).
8. The core generates the XFRC interrupt as soon as the receive packet status is read.
9. In response to the XFRC interrupt, disable the channel and stop writing the OTG_HCCHAR2 register for further requests. The core writes a channel disable request to the non-periodic request queue as soon as the OTG_HCCHAR2 register is written.
10. The core generates the RXFLVL interrupt as soon as the halt status is written to the receive FIFO.
11. Read and ignore the receive packet status.
12. The core generates a CHH interrupt as soon as the halt status is popped from the receive FIFO.
13. In response to the CHH interrupt, de-allocate the channel for other transfers.
14. Handling non-ACK responses

- **Control transactions**

Setup, data, and status stages of a control transfer must be performed as three separate transfers. setup-, data- or status-stage OUT transactions are performed similarly to the bulk OUT transactions explained previously. Data- or status-stage IN transactions are performed similarly to the bulk IN transactions explained previously. For all three stages, the application is expected to set the EPTYP field in OTG_HCCHAR1 to control. During the setup stage, the application is expected to set the PID field in OTG_HCTSIZ1 to SETUP.

- **Interrupt OUT transactions**

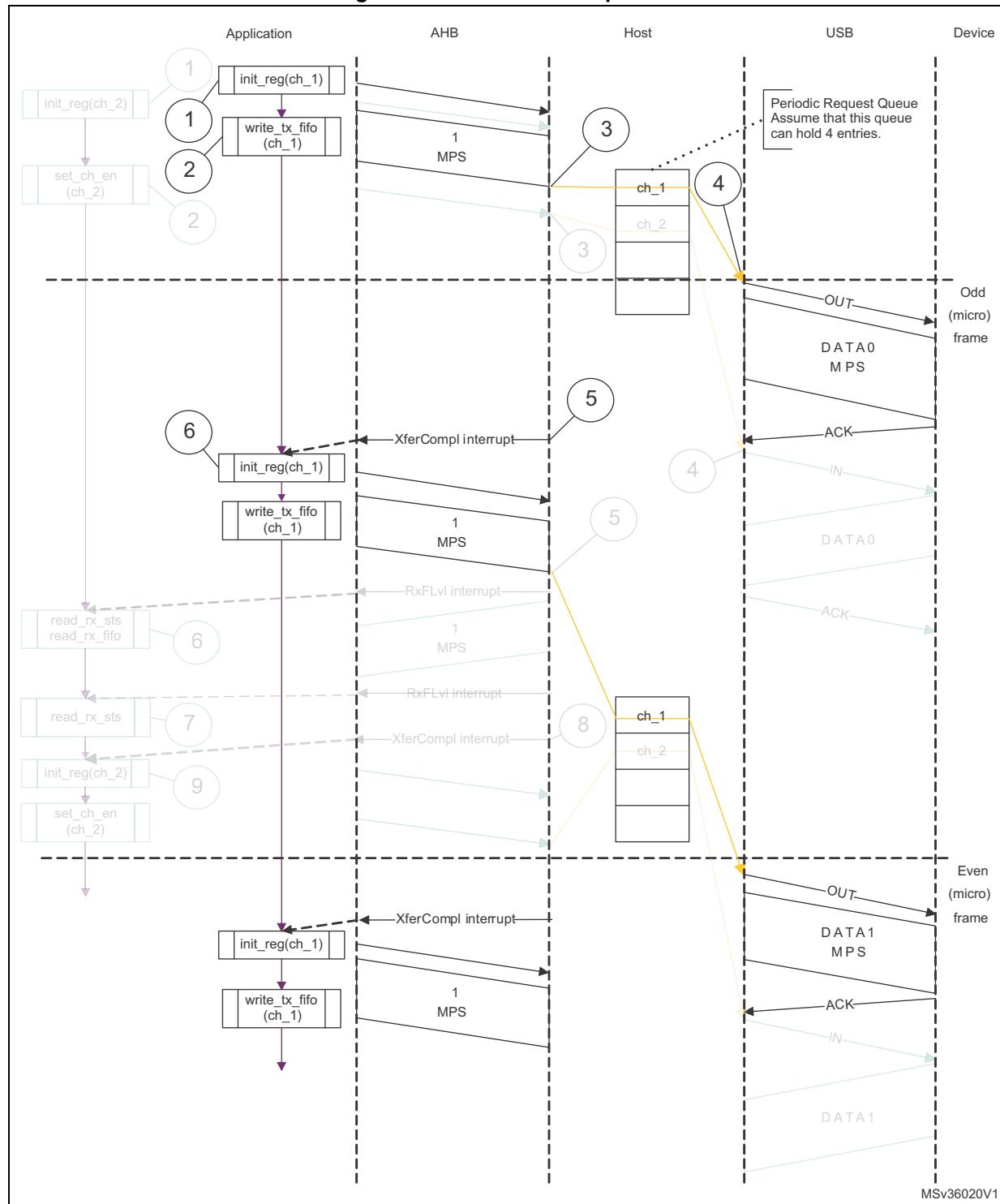
A typical interrupt OUT operation is shown in [Figure 441](#). The assumptions are:

- The application is attempting to send one packet in every frame (up to 1 maximum packet size), starting with the odd frame (transfer size = 1 024 bytes)
- The periodic transmit FIFO can hold one packet (1 KB)
- Periodic request queue depth = 4

The sequence of operations is as follows:

1. Initialize and enable channel 1. The application must set the ODDFRM bit in OTG_HCCHAR1.
2. Write the first packet for channel 1.
3. Along with the last word write of each packet, the OTG_FS/OTG_HS host writes an entry to the periodic request queue.
4. The OTG_FS/OTG_HS host attempts to send an OUT token in the next (odd) frame.
5. The OTG_FS/OTG_HS host generates an XFRC interrupt as soon as the last packet is transmitted successfully.
6. In response to the XFRC interrupt, reinitialize the channel for the next transfer.

Figure 441. Normal interrupt OUT



1. The grayed elements are not relevant in the context of this figure.

- **Interrupt service routine for interrupt OUT/IN transactions**
 - Interrupt OUT

Unmask (NAK/TXERR/STALL/XFRC/FRMOR)

```
if (XFRC)
{
    Reset Error Count
    Mask ACK
    De-allocate Channel
}
else
if (STALL or FRMOR)
{
    Mask ACK
    Unmask CHH
    Disable Channel
    if (STALL)
    {
        Transfer Done = 1
    }
}
else
if (NAK or TXERR)
{
    Rewind Buffer Pointers
    Reset Error Count
    Mask ACK
    Unmask CHH
    Disable Channel
}
else
if (CHH)
{
    Mask CHH
    if (Transfer Done or (Error_count == 3))
    {
        De-allocate Channel
    }
    else
    {
        Re-initialize Channel (in next b_interval - 1 Frame)
    }
}
else
if (ACK)
{
    Reset Error Count
    Mask ACK
}
```

The application uses the NPTXFE interrupt in OTG_GINTSTS to find the transmit FIFO space.

```
Interrupt IN
Unmask (NAK/TXERR/XFRC/BBERR/STALL/FRMOR/DTERR)
if (XFRC)
{
    Reset Error Count
    Mask ACK
    if (OTG_HCTSIZx.PKTCNT == 0)
    {
        De-allocate Channel
    }
    else
    {
        Transfer Done = 1
        Unmask CHH
        Disable Channel
    }
}
else
if (STALL or FRMOR or NAK or DTERR or BBERR)
{
    Mask ACK
    Unmask CHH
    Disable Channel
    if (STALL or BBERR)
    {
        Reset Error Count
        Transfer Done = 1
    }
    else
        if (!FRMOR)
        {
            Reset Error Count
        }
}
else
if (TXERR)
{
    Increment Error Count
    Unmask ACK
    Unmask CHH
    Disable Channel
}
else
```

```

if (CHH)
{
    Mask CHH
    if (Transfer Done or (Error_count == 3))
    {
        De-allocate Channel
    }
    else
        Re-initialize Channel (in next b_interval - 1 /Frame)
}
else
    if (ACK)
    {
        Reset Error Count
        Mask ACK
    }
}

```

- **Interrupt IN transactions**

The assumptions are:

- The application is attempting to receive one packet (up to 1 maximum packet size) in every frame, starting with odd (transfer size = 1 024 bytes).
- The receive FIFO can hold at least one maximum-packet-size packet and two status words per packet (1 031 bytes).
- Periodic request queue depth = 4.

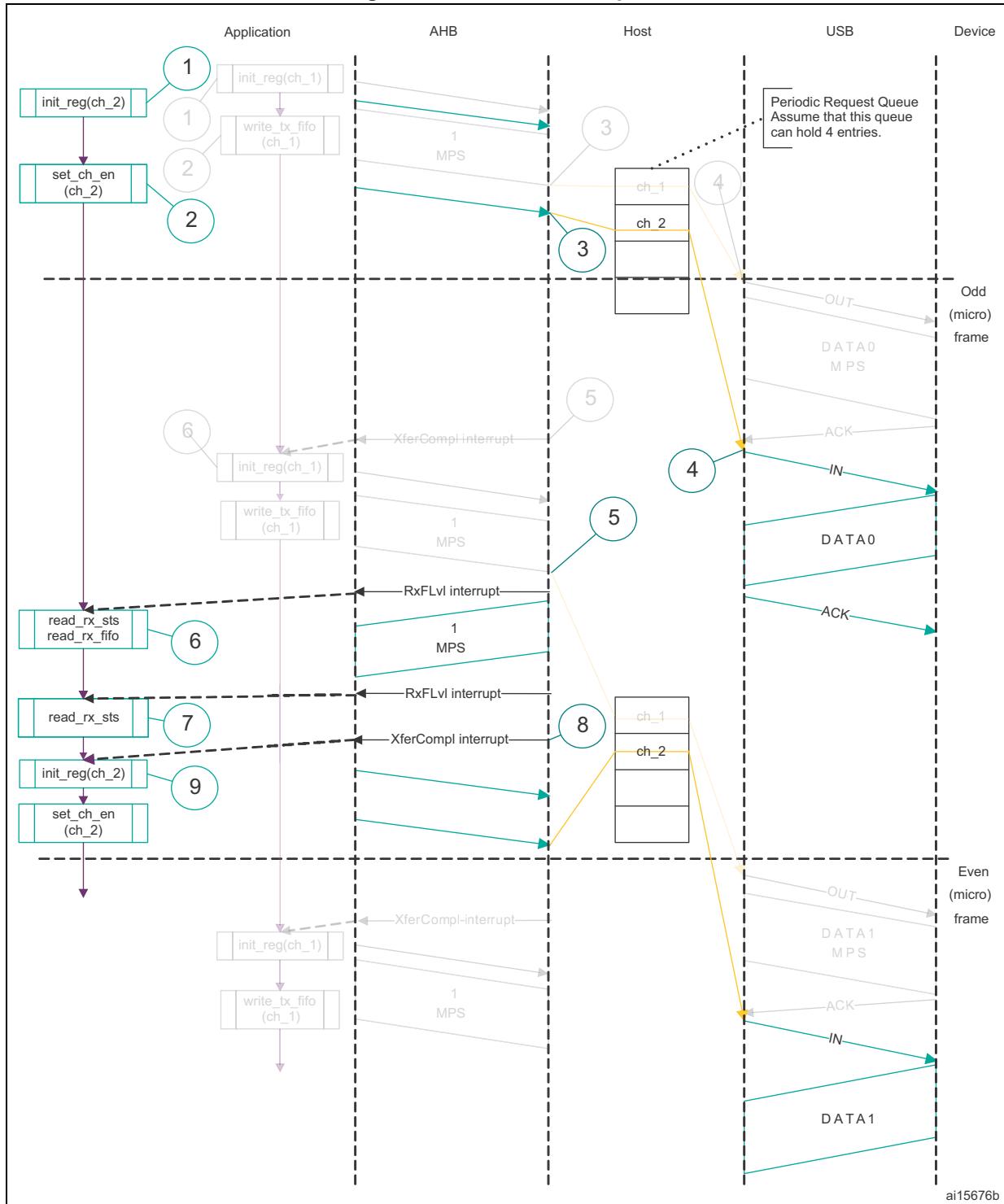
- **Normal interrupt IN operation**

The sequence of operations is as follows:

1. Initialize channel 2. The application must set the ODDFRM bit in OTG_HCCHAR2.
2. Set the CHENA bit in OTG_HCCHAR2 to write an IN request to the periodic request queue.
3. The OTG_FS/OTG_HS host writes an IN request to the periodic request queue for each OTG_HCCHAR2 register write with the CHENA bit set.
4. The OTG_FS/OTG_HS host attempts to send an IN token in the next (odd) frame.
5. As soon as the IN packet is received and written to the receive FIFO, the OTG_FS/OTG_HS host generates an RXFLVL interrupt.
6. In response to the RXFLVL interrupt, read the received packet status to determine the number of bytes received, then read the receive FIFO accordingly. The application must mask the RXFLVL interrupt before reading the receive FIFO, and unmask after reading the entire packet.
7. The core generates the RXFLVL interrupt for the transfer completion status entry in the receive FIFO. The application must read and ignore the receive packet status when the receive packet status is not an IN data packet (PKTSTS in GRXSTS ≠ 0b0010).
8. The core generates an XFRC interrupt as soon as the receive packet status is read.
9. In response to the XFRC interrupt, read the PKTCNT field in OTG_HCTSIZ2. If the PKTCNT bit in OTG_HCTSIZ2 is not equal to 0, disable the channel before re-

initializing the channel for the next transfer, if any). If PKTCNT bit in OTG_HCTSIZ2 = 0, reinitialize the channel for the next transfer. This time, the application must reset the ODDFRM bit in OTG_HCCHAR2.

Figure 442. Normal interrupt IN



1. The grayed elements are not relevant in the context of this figure.

- **Isochronous OUT transactions**

A typical isochronous OUT operation is shown in [Figure 442](#). The assumptions are:

- The application is attempting to send one packet every frame (up to 1 maximum)

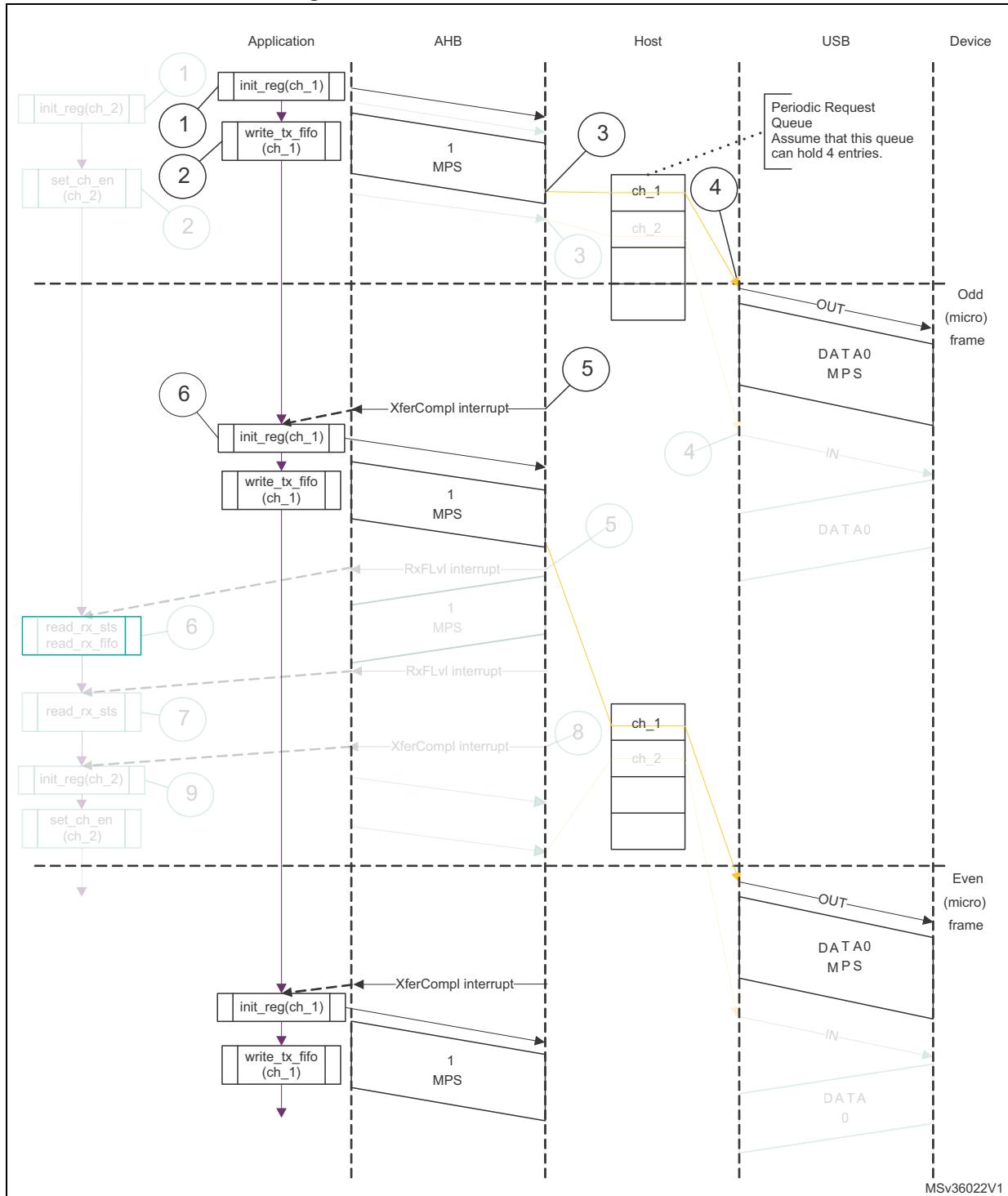
packet size), starting with an odd frame. (transfer size = 1 024 bytes).

- The periodic transmit FIFO can hold one packet (1 KB).
- Periodic request queue depth = 4.

The sequence of operations is as follows:

1. Initialize and enable channel 1. The application must set the ODDFRM bit in OTG_HCCHAR1.
2. Write the first packet for channel 1.
3. Along with the last word write of each packet, the OTG_FS/OTG_HS host writes an entry to the periodic request queue.
4. The OTG_FS/OTG_HS host attempts to send the OUT token in the next frame (odd).
5. The OTG_FS/OTG_HS host generates the XFRC interrupt as soon as the last packet is transmitted successfully.
6. In response to the XFRC interrupt, reinitialize the channel for the next transfer.
7. Handling non-ACK responses

Figure 443. Isochronous OUT transactions



1. The grayed elements are not relevant in the context of this figure.

- **Interrupt service routine for isochronous OUT/IN transactions**

Code sample: isochronous OUT

```
Unmask (FMRMOR/XFRC)
```

```
if (XFRC)
```

```
{  
    De-allocate Channel  
}  
  
else  
if (FRMOR)  
{  
    Unmask CHH  
    Disable Channel  
}  
  
else  
if (CHH)  
{  
    Mask CHH  
    De-allocate Channel  
}  
  
Code sample: Isochronous IN  
Unmask (TXERR/XFRC/FRMOR/BBERR)  
if (XFRC or FRMOR)  
{  
if (XFRC and (OTG_HCTSIZx.PKTCNT == 0))  
{  
    Reset Error Count  
    De-allocate Channel  
}  
else  
{  
    Unmask CHH  
    Disable Channel  
}  
}  
else  
if (TXERR or BBERR)  
{  
    Increment Error Count  
    Unmask CHH  
    Disable Channel  
}  
else  
if (CHH)  
{  
    Mask CHH  
    if (Transfer Done or (Error_count == 3))  
{  
        De-allocate Channel  
    }  
}
```

```
        else
        {
            Re-initialize Channel
        }
    }
```

- **Isochronous IN transactions**

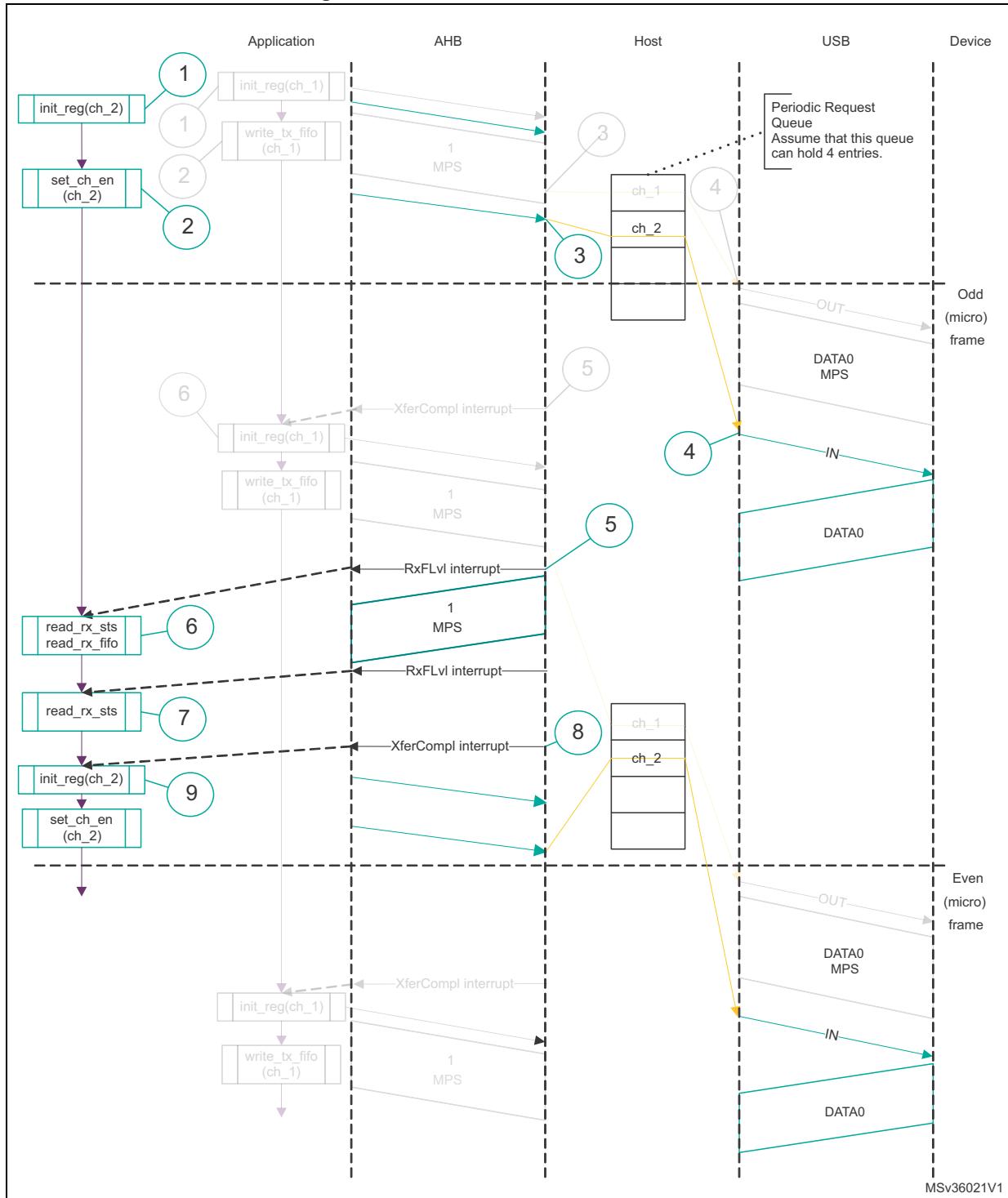
The assumptions are:

- The application is attempting to receive one packet (up to 1 maximum packet size) in every frame starting with the next odd frame (transfer size = 1 024 bytes).
- The receive FIFO can hold at least one maximum-packet-size packet and two status word per packet (1 031 bytes).
- Periodic request queue depth = 4.

The sequence of operations is as follows:

1. Initialize channel 2. The application must set the ODDFRM bit in OTG_HCCHAR2.
2. Set the CHENA bit in OTG_HCCHAR2 to write an IN request to the periodic request queue.
3. The OTG_FS/OTG_HS host writes an IN request to the periodic request queue for each OTG_HCCHAR2 register write with the CHENA bit set.
4. The OTG_FS/OTG_HS host attempts to send an IN token in the next odd frame.
5. As soon as the IN packet is received and written to the receive FIFO, the OTG_FS/OTG_HS host generates an RXFLVL interrupt.
6. In response to the RXFLVL interrupt, read the received packet status to determine the number of bytes received, then read the receive FIFO accordingly. The application must mask the RXFLVL interrupt before reading the receive FIFO, and unmask it after reading the entire packet.
7. The core generates an RXFLVL interrupt for the transfer completion status entry in the receive FIFO. This time, the application must read and ignore the receive packet status when the receive packet status is not an IN data packet (PKTSTS bit in OTG_GRXSTSR ≠ 0b0010).
8. The core generates an XFRC interrupt as soon as the receive packet status is read.
9. In response to the XFRC interrupt, read the PKTCNT field in OTG_HCTSIZ2. If PKTCNT ≠ 0 in OTG_HCTSIZ2, disable the channel before re-initializing the channel for the next transfer, if any. If PKTCNT = 0 in OTG_HCTSIZ2, reinitialize the channel for the next transfer. This time, the application must reset the ODDFRM bit in OTG_HCCHAR2.

Figure 444. Isochronous IN transactions



1. The grayed elements are not relevant in the context of this figure.

- **Selecting the queue depth**

Choose the periodic and non-periodic request queue depths carefully to match the number of periodic/non-periodic endpoints accessed.

The non-periodic request queue depth affects the performance of non-periodic

transfers. The deeper the queue (along with sufficient FIFO size), the more often the core is able to pipeline non-periodic transfers. If the queue size is small, the core is able to put in new requests only when the queue space is freed up.

The core's periodic request queue depth is critical to perform periodic transfers as scheduled. Select the periodic queue depth, based on the number of periodic transfers scheduled in a microframe. If the periodic request queue depth is smaller than the periodic transfers scheduled in a microframe, a frame overrun condition occurs.

- **Handling babble conditions**

OTG_FS/OTG_HS controller handles two cases of babble: packet babble and port babble. Packet babble occurs if the device sends more data than the maximum packet size for the channel. Port babble occurs if the core continues to receive data from the device at EOF2 (the end of frame 2, which is very close to SOF).

When OTG_FS/OTG_HS controller detects a packet babble, it stops writing data into the Rx buffer and waits for the end of packet (EOP). When it detects an EOP, it flushes already written data in the Rx buffer and generates a Babble interrupt to the application.

When OTG_FS/OTG_HS controller detects a port babble, it flushes the Rx FIFO and disables the port. The core then generates a port disabled interrupt (HPRTINT in OTG_GINTSTS, PENCHNG in OTG_HPRT). On receiving this interrupt, the application must determine that this is not due to an overcurrent condition (another cause of the port disabled interrupt) by checking POCA in OTG_HPRT, then perform a soft reset. The core does not send any more tokens after it has detected a port babble condition.

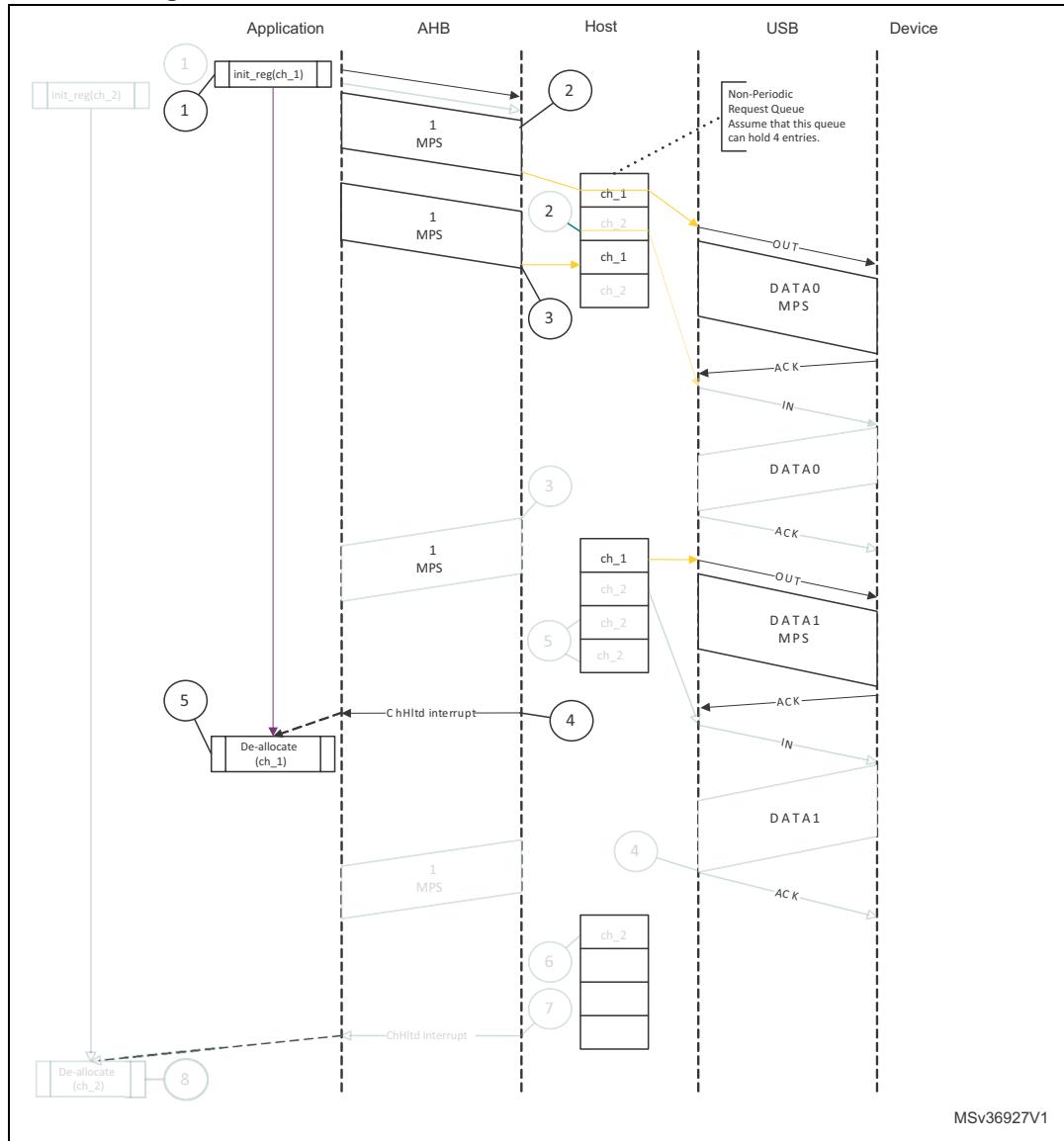
Note: The following paragraphs, ranging from here to the beginning of [Section 32.16](#), and covering DMA configurations, apply only to USB OTG HS.

- **Bulk and control OUT/SETUP transactions in DMA mode**

The sequence of operations is as follows:

1. Initialize and enable channel 1 as explained in [Section : Channel initialization](#).
2. The OTG_HS host starts fetching the first packet as soon as the channel is enabled. For internal DMA mode, the OTG_HS host uses the programmed DMA address to fetch the packet.
3. After fetching the last 32-bit word of the second (last) packet, the OTG_HS host masks channel 1 internally for further arbitration.
4. The OTG_HS host generates a CHH interrupt as soon as the last packet is sent.
5. In response to the CHH interrupt, de-allocate the channel for other transfers.

Figure 445. Normal bulk/control OUT/SETUP transactions - DMA



MSv36927V1

- **NAK and NYET handling with internal DMA:**

1. The OTG_HS host sends a bulk OUT transaction.
2. The device responds with NAK or NYET.
3. If the application has unmasked NAK or NYET, the core generates the corresponding interrupt(s) to the application. The application is not required to service these interrupts, since the core takes care of rewinding the buffer pointers and re-initializing the Channel without application intervention.
4. The core automatically issues a ping token.
5. When the device returns an ACK, the core continues with the transfer. Optionally, the application can utilize these interrupts, in which case the NAK or NYET interrupt is masked by the application.

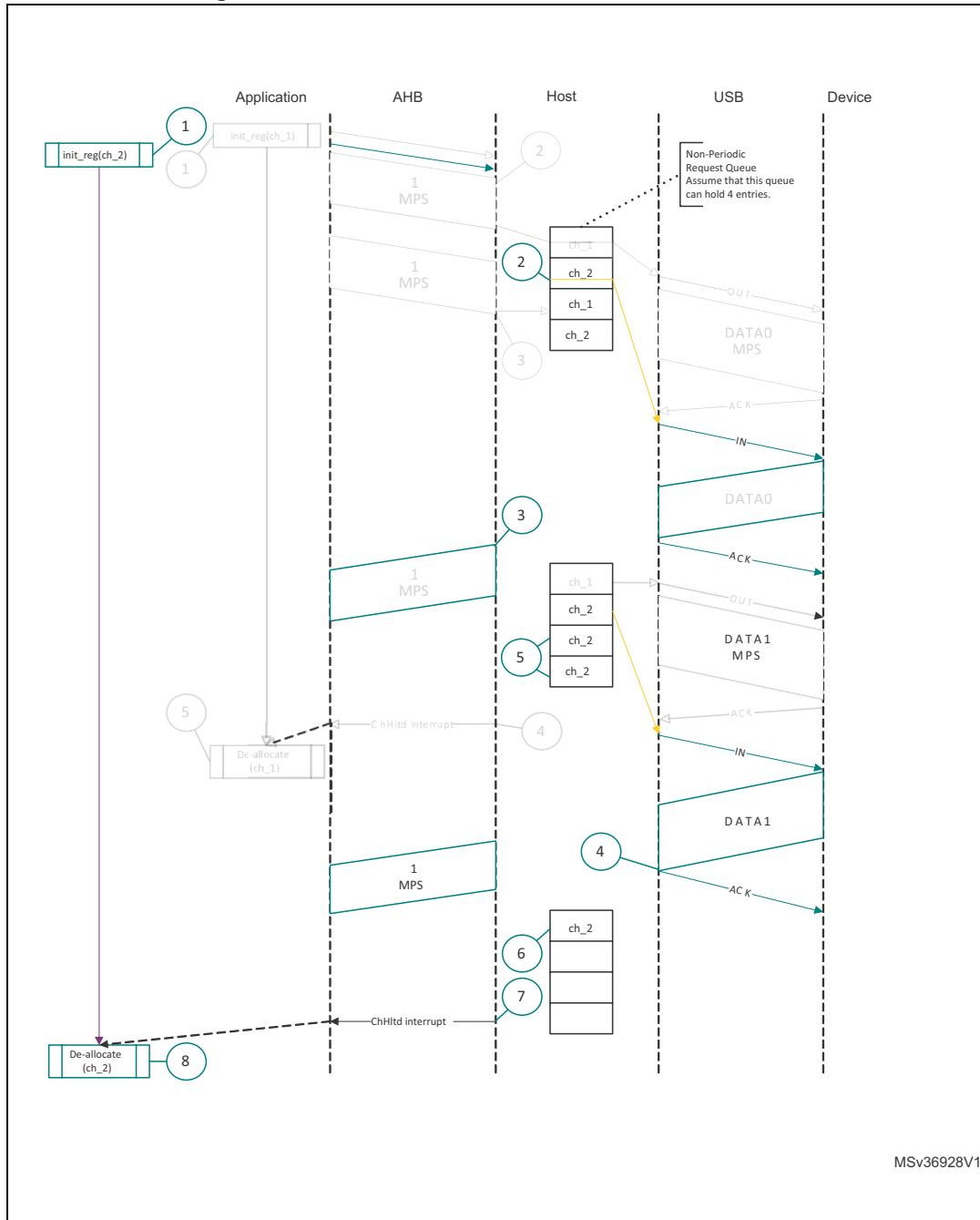
The core does not generate a separate interrupt when NAK or NYET is received by the host functionality.

- **Bulk and control IN transactions in DMA mode**

The sequence of operations is as follows:

1. Initialize and enable the used channel (channel x) as explained in [*Section : Channel initialization*](#).
2. The OTG_HS host writes an IN request to the request queue as soon as the channel receives the grant from the arbiter (arbitration is performed in a round-robin fashion).
3. The OTG_HS host starts writing the received data to the system memory as soon as the last byte is received with no errors.
4. When the last packet is received, the OTG_HS host sets an internal flag to remove any extra IN requests from the request queue.
5. The OTG_HS host flushes the extra requests.
6. The final request to disable channel x is written to the request queue. At this point, channel 2 is internally masked for further arbitration.
7. The OTG_HS host generates the CHH interrupt as soon as the disable request comes to the top of the queue.
8. In response to the CHH interrupt, de-allocate the channel for other transfers.

Figure 446. Normal bulk/control IN transaction - DMA

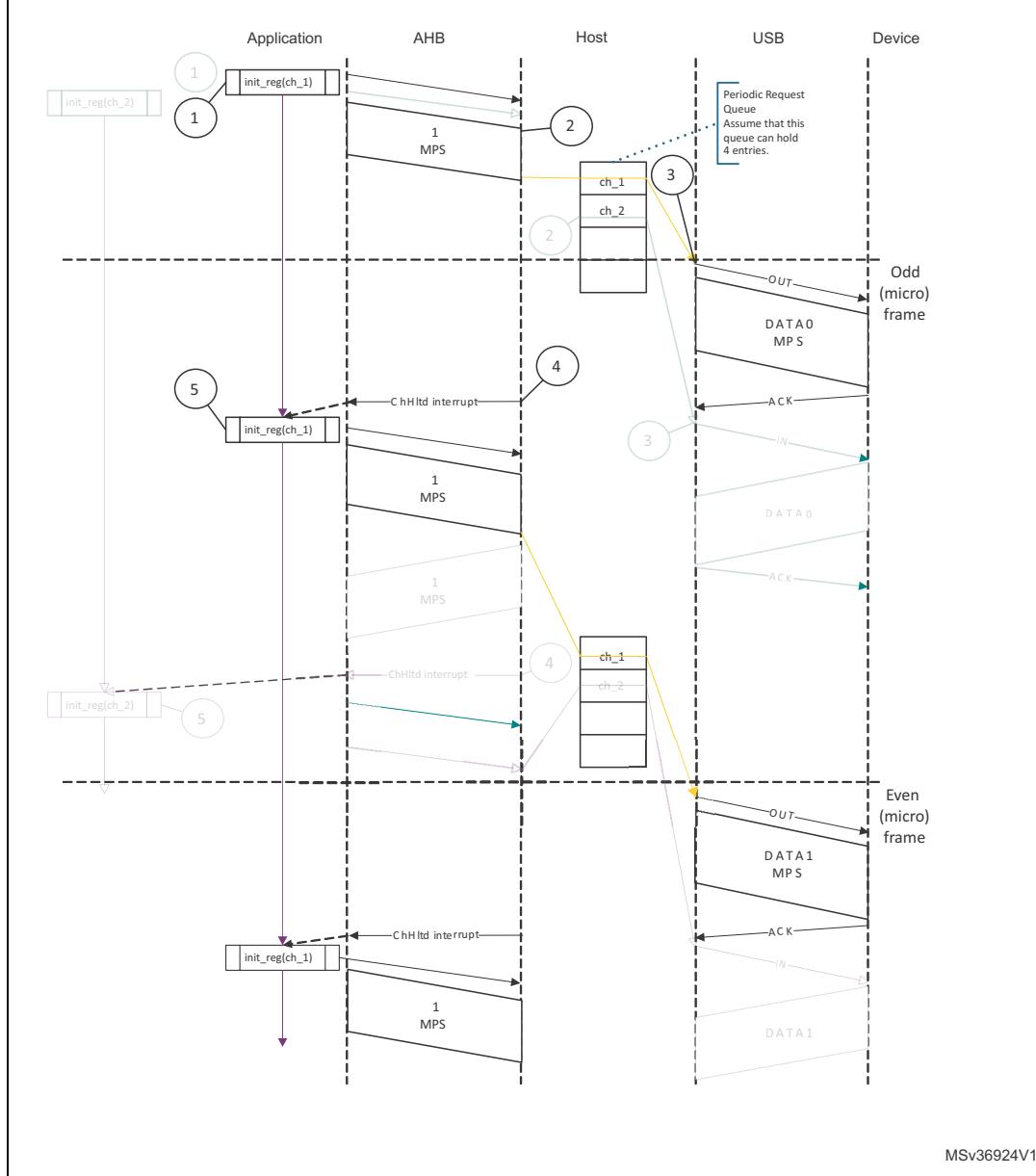


- **Interrupt OUT transactions in DMA mode**

1. Initialize and enable channel x as explained in [Section : Channel initialization](#).
2. The OTG_HS host starts fetching the first packet as soon the channel is enabled and writes the OUT request along with the last 32-bit word fetch. In high-bandwidth transfers, the OTG_HS host continues fetching the next packet (up to the value specified in the MC field) before switching to the next channel.
3. The OTG_HS host attempts to send the OUT token at the beginning of the next odd frame/micro-frame.

4. After successfully transmitting the packet, the OTG_HS host generates a CHH interrupt.
5. In response to the CHH interrupt, reinitialize the channel for the next transfer.

Figure 447. Normal interrupt OUT transactions - DMA mode

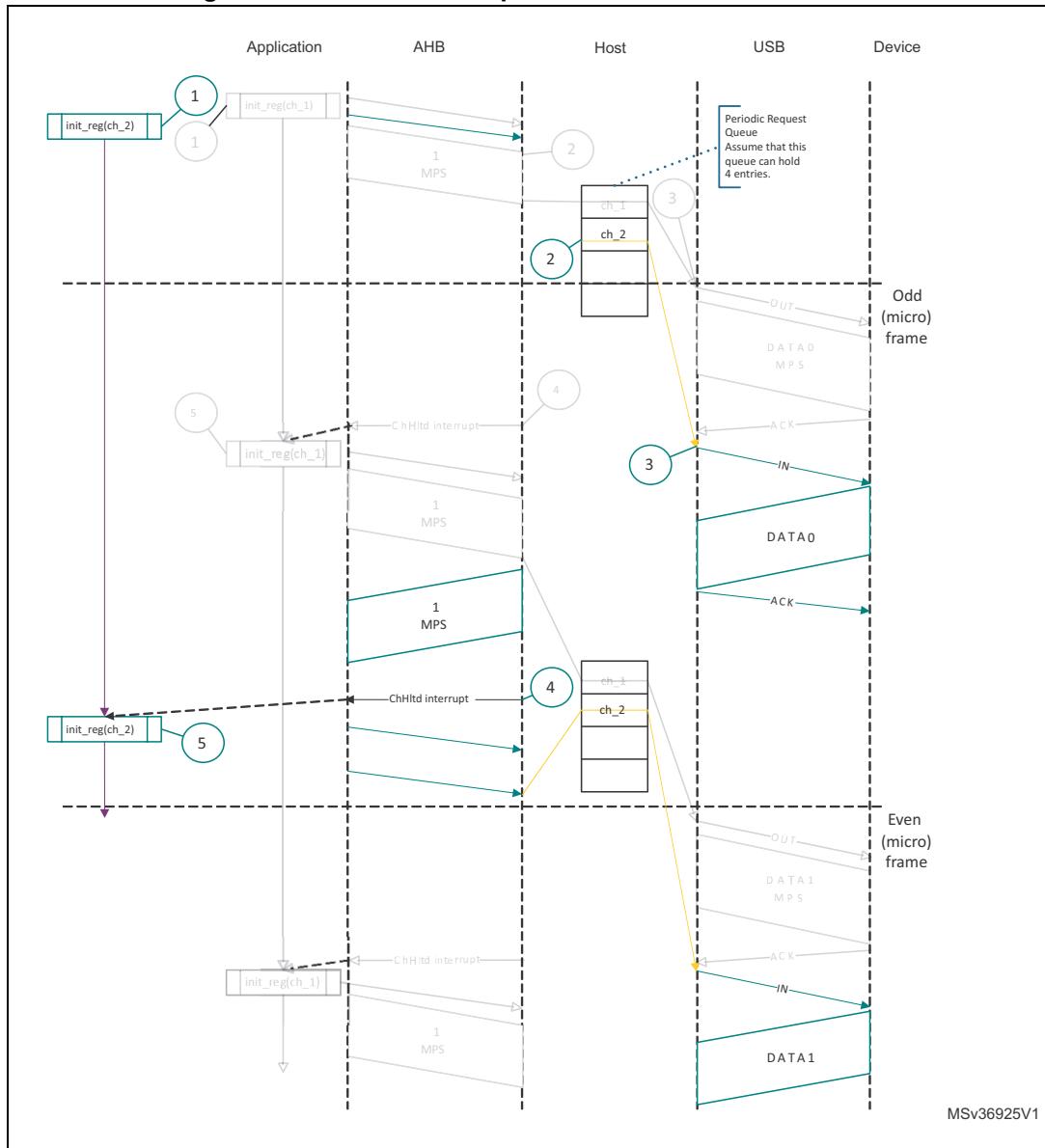


- **Interrupt IN transactions in DMA mode**

- The sequence of operations (channelx) is as follows:
1. Initialize and enable channel x as explained in [Section : Channel initialization](#).
 2. The OTG_HS host writes an IN request to the request queue as soon as the channel x gets the grant from the arbiter (round-robin with fairness). In high-bandwidth transfers, the OTG_HS host writes consecutive writes up to MC times.

3. The OTG_HS host attempts to send an IN token at the beginning of the next (odd) frame/micro-frame.
4. As soon as the packet is received and written to the receive FIFO, the OTG_HS host generates a CHH interrupt.
5. In response to the CHH interrupt, reinitialize the channel for the next transfer.

Figure 448. Normal interrupt IN transactions - DMA mode



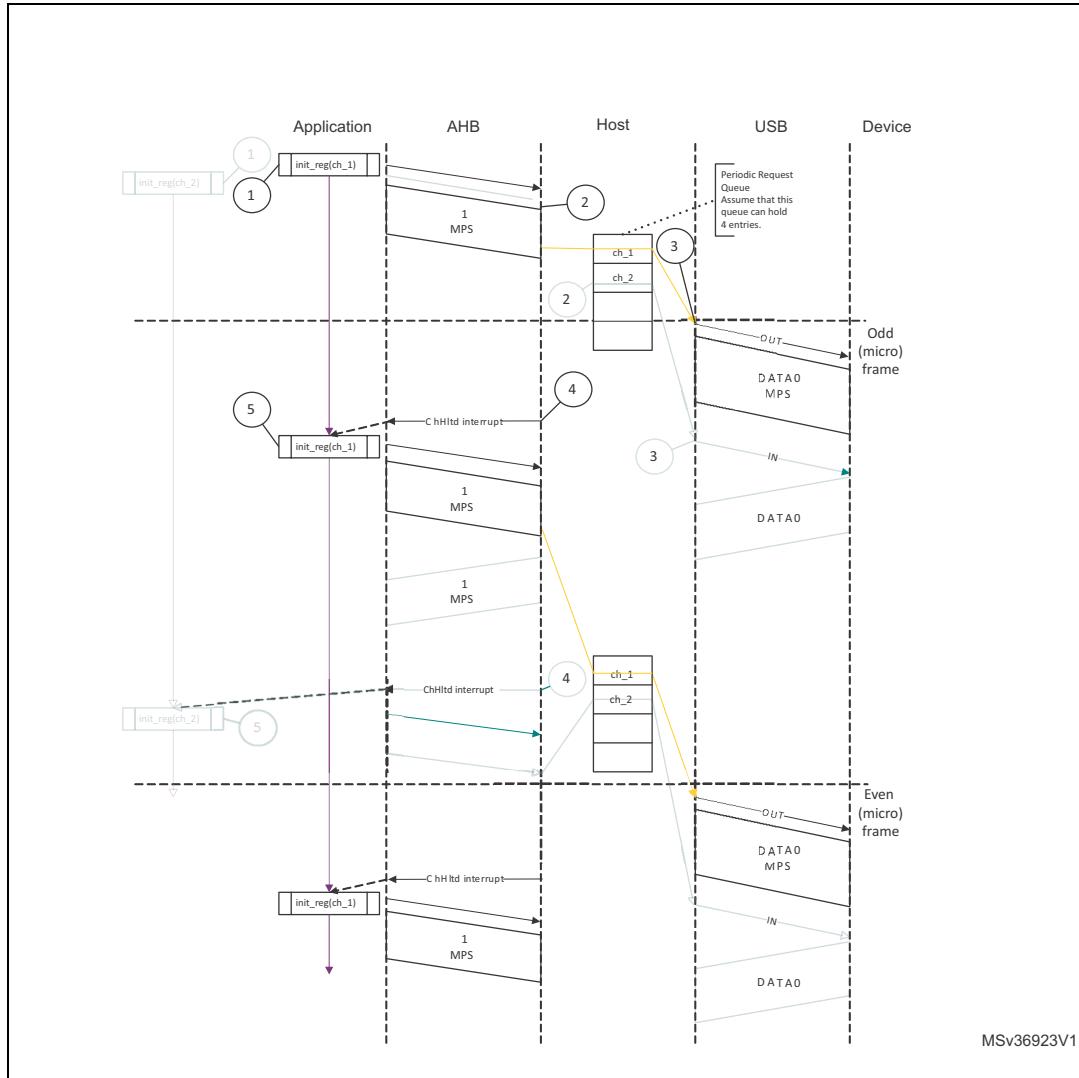
- **Isochronous OUT transactions in DMA mode**

1. Initialize and enable channel x as explained in [Section : Channel initialization](#).
2. The OTG_HS host starts fetching the first packet as soon as the channel is enabled, and writes the OUT request along with the last 32-bit word fetch. In high-bandwidth

transfers, the OTG_HS host continues fetching the next packet (up to the value specified in the MC field) before switching to the next channel.

3. The OTG_HS host attempts to send an OUT token at the beginning of the next (odd) frame/micro-frame.
4. After successfully transmitting the packet, the OTG_HS host generates a CHH interrupt.
5. In response to the CHH interrupt, reinitialize the channel for the next transfer.

Figure 449. Normal isochronous OUT transaction - DMA mode



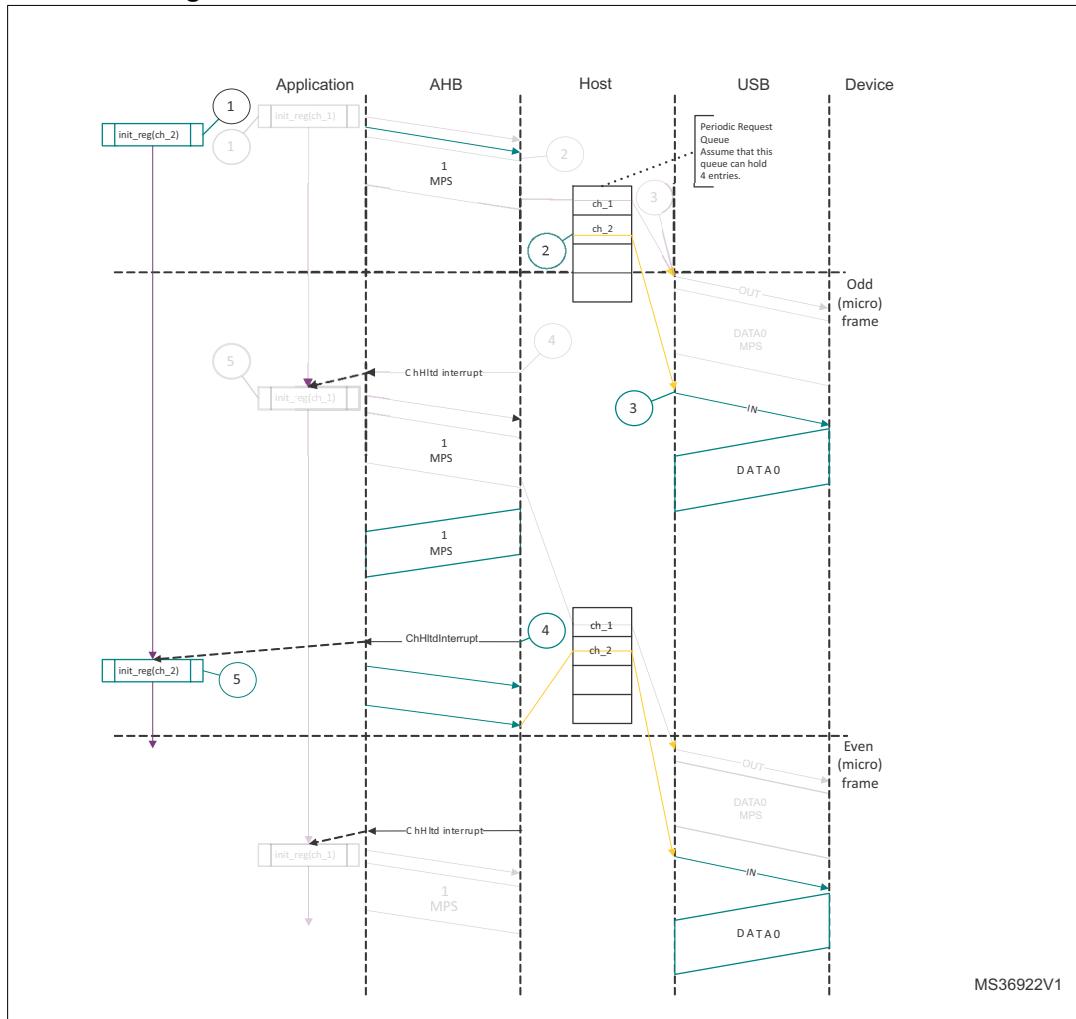
- Isochronous IN transactions in DMA mode**

The sequence of operations ((channel x) is as follows:

1. Initialize and enable channel x as explained in [Section : Channel initialization](#).
2. The OTG_HS host writes an IN request to the request queue as soon as the channel x gets the grant from the arbiter (round-robin with fairness). In high-bandwidth transfers, the OTG_HS host performs consecutive write operations up to MC times.

3. The OTG_HS host attempts to send an IN token at the beginning of the next (odd) frame/micro-frame.
4. As soon as the packet is received and written to the receive FIFO, the OTG_HS host generates a CHH interrupt.
5. In response to the CHH interrupt, reinitialize the channel for the next transfer.

Figure 450. Normal isochronous IN transactions - DMA mode



- **Bulk and control OUT/SETUP split transactions in DMA mode**

- The sequence of operations in (channel x) is as follows:
1. Initialize and enable channel x for start split as explained in [Section : Channel initialization](#).
 2. The OTG_HS host starts fetching the first packet as soon the channel is enabled and writes the OUT request along with the last 32-bit word fetch.
 3. After successfully transmitting start split, the OTG_HS host generates the CHH interrupt.
 4. In response to the CHH interrupt, set the COMPLSPLT bit in OTG_HCSPLT1 to send the complete split.

5. After successfully transmitting complete split, the OTG_HS host generates the CHH interrupt.
6. In response to the CHH interrupt, de-allocate the channel.

- **Bulk/control IN split transactions in DMA mode**

The sequence of operations (channel x) is as follows:

1. Initialize and enable channel x as explained in [Section : Channel initialization](#).
2. The OTG_HS host writes the start split request to the nonperiodic request after getting the grant from the arbiter. The OTG_HS host masks the channel x internally for the arbitration after writing the request.
3. As soon as the IN token is transmitted, the OTG_HS host generates the CHH interrupt.
4. In response to the CHH interrupt, set the COMPLSPLT bit in OTG_HCSPLT2 and re-enable the channel to send the complete split token. This unmasks channel x for arbitration.
5. The OTG_HS host writes the complete split request to the nonperiodic request after receiving the grant from the arbiter.
6. The OTG_HS host starts writing the packet to the system memory after receiving the packet successfully.
7. As soon as the received packet is written to the system memory, the OTG_HS host generates a CHH interrupt.
8. In response to the CHH interrupt, de-allocate the channel.

- **Interrupt OUT split transactions in DMA mode**

The sequence of operations in (channel x) is as follows:

1. Initialize and enable channel 1 for start split as explained in [Section : Channel initialization](#). The application must set the ODDFRM bit in OTG_HCCHAR1.
2. The OTG_HS host starts reading the packet.
3. The OTG_HS host attempts to send the start split transaction.
4. After successfully transmitting the start split, the OTG_HS host generates the CHH interrupt.
5. In response to the CHH interrupt, set the COMPLSPLT bit in OTG_HCSPLT1 to send the complete split.
6. After successfully completing the complete split transaction, the OTG_HS host generates the CHH interrupt.
7. In response to CHH interrupt, de-allocate the channel.

- **Interrupt IN split transactions in DMA mode**

The sequence of operations in (channel x) is as follows:

1. Initialize and enable channel x for start split as explained in [Section : Channel initialization](#).
2. The OTG_HS host writes an IN request to the request queue as soon as channel x receives the grant from the arbiter.
3. The OTG_HS host attempts to send the start split IN token at the beginning of the next odd micro-frame.
4. The OTG_HS host generates the CHH interrupt after successfully transmitting the start split IN token.
5. In response to the CHH interrupt, set the COMPLSPLT bit in OTG_HCSPLT2 to send the complete split.

6. As soon as the packet is received successfully, the OTG_HS host starts writing the data to the system memory.
7. The OTG_HS host generates the CHH interrupt after transferring the received data to the system memory.
8. In response to the CHH interrupt, de-allocate or reinitialize the channel for the next start split.

- **Isochronous OUT split transactions in DMA mode**

The sequence of operations (channel x) is as follows:

1. Initialize and enable channel x for start split (begin) as explained in [Section : Channel initialization](#). The application must set the ODDFRM bit in OTG_HCCHAR1. Program the MPS field.
2. The OTG_HS host starts reading the packet.
3. After successfully transmitting the start split (begin), the OTG_HS host generates the CHH interrupt.
4. In response to the CHH interrupt, reinitialize the registers to send the start split (end).
5. After successfully transmitting the start split (end), the OTG_HS host generates a CHH interrupt.
6. In response to the CHH interrupt, de-allocate the channel.

- **Isochronous IN split transactions in DMA mode**

The sequence of operations (channel x) is as follows:

1. Initialize and enable channel x for start split as explained in [Section : Channel initialization](#).
2. The OTG_HS host writes an IN request to the request queue as soon as channel x receives the grant from the arbiter.
3. The OTG_HS host attempts to send the start split IN token at the beginning of the next odd micro-frame.
4. The OTG_HS host generates the CHH interrupt after successfully transmitting the start split IN token.
5. In response to the CHH interrupt, set the COMPLSPLT bit in OTG_HCSPLT2 to send the complete split.
6. As soon as the packet is received successfully, the OTG_HS host starts writing the data to the system memory.

The OTG_HS host generates the CHH interrupt after transferring the received data to the system memory. In response to the CHH interrupt, de-allocate the channel or reinitialize the channel for the next start split.

32.16.6 Device programming model

Endpoint initialization on USB reset

1. Set the NAK bit for all OUT endpoints
 - SNAK = 1 in OTG_DOEPCTLx (for all OUT endpoints)
2. Unmask the following interrupt bits
 - INEP0 = 1 in OTG_DAINTMSK (control 0 IN endpoint)
 - OUTEP0 = 1 in OTG_DAINTMSK (control 0 OUT endpoint)
 - STUPM = 1 in OTG_DOEPMSK
 - XFRCM = 1 in OTG_DOEPMSK
 - XFRCM = 1 in OTG_DIEPMSK
 - TOM = 1 in OTG_DIEPMSK
3. Set up the data FIFO RAM for each of the FIFOs
 - Program the OTG_GRXFSIZ register, to be able to receive control OUT data and setup data. If thresholding is not enabled, at a minimum, this must be equal to 1 max packet size of control endpoint 0 + 2 words (for the status of the control OUT data packet) + 10 words (for setup packets).
 - Program the OTG_DIEPTXF0 register (depending on the FIFO number chosen) to be able to transmit control IN data. At a minimum, this must be equal to 1 max packet size of control endpoint 0.
4. Program the following fields in the endpoint-specific registers for control OUT endpoint 0 to receive a SETUP packet
 - STUPCNT = 3 in OTG_DOEPTSIZ0 (to receive up to 3 back-to-back SETUP packets)
5. For USB OTG_HS in DMA mode, the OTG_DOEPDMA0 register should have a valid memory address to store any SETUP packets received.

At this point, all initialization required to receive SETUP packets is done.

Endpoint initialization on enumeration completion

1. On the Enumeration Done interrupt (ENUMDNE in OTG_GINTSTS), read the OTG_DSTS register to determine the enumeration speed.
2. Program the MPSIZ field in OTG_DIEPCTL0 to set the maximum packet size. This step configures control endpoint 0. The maximum packet size for a control endpoint depends on the enumeration speed.
3. For USB OTG_HS in DMA mode, program the OTG_DOEPCTL0 register to enable control OUT endpoint 0, to receive a SETUP packet.

At this point, the device is ready to receive SOF packets and is configured to perform control transfers on control endpoint 0.

Endpoint initialization on SetAddress command

This section describes what the application must do when it receives a SetAddress command in a SETUP packet.

1. Program the OTG_DCFG register with the device address received in the SetAddress command
2. Program the core to send out a status IN packet

Endpoint initialization on SetConfiguration/SetInterface command

This section describes what the application must do when it receives a SetConfiguration or SetInterface command in a SETUP packet.

1. When a SetConfiguration command is received, the application must program the endpoint registers to configure them with the characteristics of the valid endpoints in the new configuration.
2. When a SetInterface command is received, the application must program the endpoint registers of the endpoints affected by this command.
3. Some endpoints that were active in the prior configuration or alternate setting are not valid in the new configuration or alternate setting. These invalid endpoints must be deactivated.
4. Unmask the interrupt for each active endpoint and mask the interrupts for all inactive endpoints in the OTG_DAINTMSK register.
5. Set up the data FIFO RAM for each FIFO.
6. After all required endpoints are configured; the application must program the core to send a status IN packet.

At this point, the device core is configured to receive and transmit any type of data packet.

Endpoint activation

This section describes the steps required to activate a device endpoint or to configure an existing device endpoint to a new type.

1. Program the characteristics of the required endpoint into the following fields of the OTG_DIEPCTLx register (for IN or bidirectional endpoints) or the OTG_DOEPCTLx register (for OUT or bidirectional endpoints).
 - Maximum packet size
 - USB active endpoint = 1
 - Endpoint start data toggle (for interrupt and bulk endpoints)
 - Endpoint type
 - Tx FIFO number
2. Once the endpoint is activated, the core starts decoding the tokens addressed to that endpoint and sends out a valid handshake for each valid token received for the endpoint.

Endpoint deactivation

This section describes the steps required to deactivate an existing endpoint.

1. In the endpoint to be deactivated, clear the USB active endpoint bit in the OTG_DIEPCTLx register (for IN or bidirectional endpoints) or the OTG_DOEPCTLx register (for OUT or bidirectional endpoints).
2. Once the endpoint is deactivated, the core ignores tokens addressed to that endpoint, which results in a timeout on the USB.

Note:

The application must meet the following conditions to set up the device core to handle traffic:

NPTXFEM and RXFLVLM in the OTG_GINTMSK register must be cleared.

Operational model

SETUP and OUT data transfers:

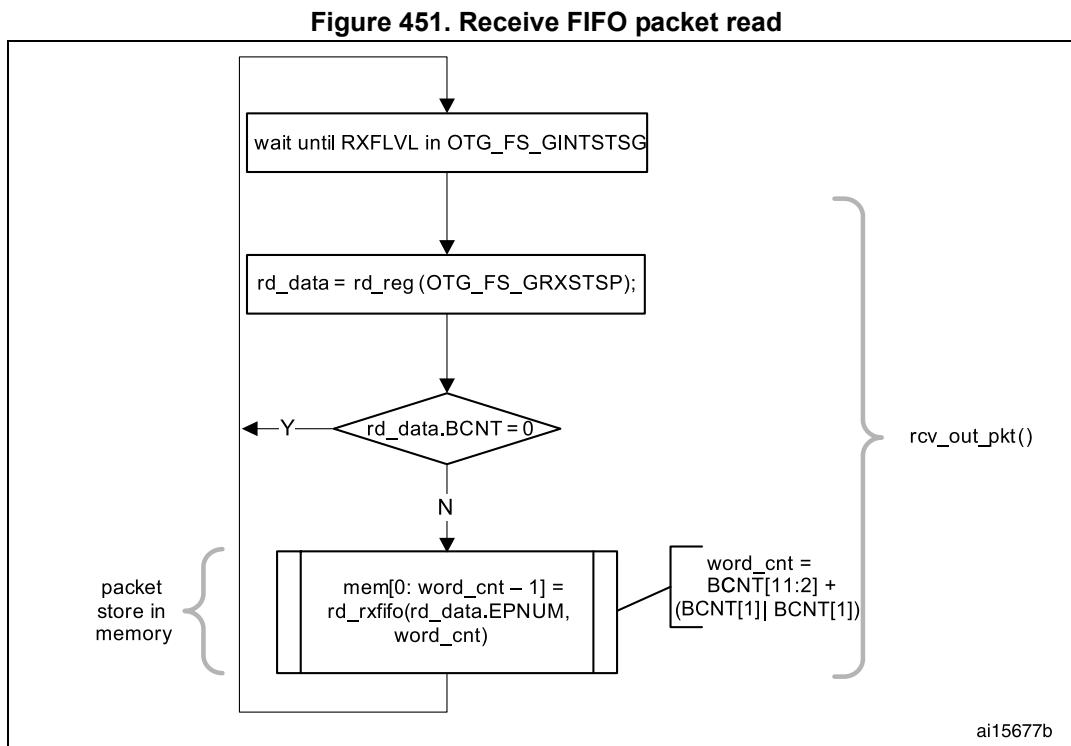
This section describes the internal data flow and application-level operations during data OUT transfers and SETUP transactions.

- **Packet read**

This section describes how to read packets (OUT data and SETUP packets) from the receive FIFO.

1. On catching an RXFLVL interrupt (OTG_GINTSTS register), the application must read the receive status pop register (OTG_GRXSTSP).
2. The application can mask the RXFLVL interrupt (in OTG_GINTSTS) by writing to RXFLVLM = 0 (in OTG_GINTMSK), until it has read the packet from the receive FIFO.
3. If the received packet's byte count is not 0, the byte count amount of data is popped from the receive data FIFO and stored in memory. If the received packet byte count is 0, no data is popped from the receive data FIFO.
4. The receive status readout of the packet of FIFO indicates one of the following:
 - a) Global OUT NAK pattern:
PKTSTS = Global OUT NAK, BCNT = 0x000, EPNUM = (0x0), DPID = (0b00).
These data indicate that the global OUT NAK bit has taken effect.
 - b) SETUP packet pattern:
PKTSTS = SETUP, BCNT = 0x008, EPNUM = Control EP Num, DPID = DATA0. These data indicate that a SETUP packet for the specified endpoint is now available for reading from the receive FIFO.
 - c) Setup stage done pattern:
PKTSTS = Setup Stage Done, BCNT = 0x0, EPNUM = Control EP Num, DPID = (0b00).
These data indicate that the setup stage for the specified endpoint has completed and the data stage has started. After this entry is popped from the receive FIFO, the core asserts a setup interrupt on the specified control OUT endpoint.
 - d) Data OUT packet pattern:
PKTSTS = DataOUT, BCNT = size of the received data OUT packet ($0 \leq BCNT \leq 1\,024$), EPNUM = EPNUM on which the packet was received, DPID = Actual Data PID.
 - e) Data transfer completed pattern:
PKTSTS = Data OUT transfer done, BCNT = 0x0, EPNUM = OUT EP Num on which the data transfer is complete, DPID = (0b00).
These data indicate that an OUT data transfer for the specified OUT endpoint has completed. After this entry is popped from the receive FIFO, the core asserts a transfer completed interrupt on the specified OUT endpoint.
5. After the data payload is popped from the receive FIFO, the RXFLVL interrupt (OTG_GINTSTS) must be unmasked.
6. Steps 1–5 are repeated every time the application detects assertion of the interrupt line due to RXFLVL in OTG_GINTSTS. Reading an empty receive FIFO can result in undefined core behavior.

Figure 451 provides a flowchart of the above procedure.



SETUP transactions

This section describes how the core handles SETUP packets and the application's sequence for handling SETUP transactions.

- **Application requirements**

1. To receive a SETUP packet, the STUPCNT field (OTG_DOEPTSIZx) in a control OUT endpoint must be programmed to a non-zero value. When the application programs the STUPCNT field to a non-zero value, the core receives SETUP packets and writes them to the receive FIFO, irrespective of the NAK status and EPENA bit setting in OTG_DOEPCTLx. The STUPCNT field is decremented every time the control endpoint receives a SETUP packet. If the STUPCNT field is not programmed to a proper value before receiving a SETUP packet, the core still receives the SETUP packet and decrements the STUPCNT field, but the application may not be able to determine the correct number of SETUP packets received in the setup stage of a control transfer.
 - STUPCNT = 3 in OTG_DOEPTSIZx
2. The application must always allocate some extra space in the receive data FIFO, to be able to receive up to three SETUP packets on a control endpoint.
 - The space to be reserved is 10 words. Three words are required for the first SETUP packet, 1 word is required for the setup stage done word and 6 words are required to store two extra SETUP packets among all control endpoints.
 - 3 words per SETUP packet are required to store 8 bytes of SETUP data and 4 bytes of SETUP status (setup packet pattern). The core reserves this space in the

receive data FIFO to write SETUP data only, and never uses this space for data packets.

3. The application must read the 2 words of the SETUP packet from the receive FIFO.
4. The application must read and discard the setup stage done word from the receive FIFO.

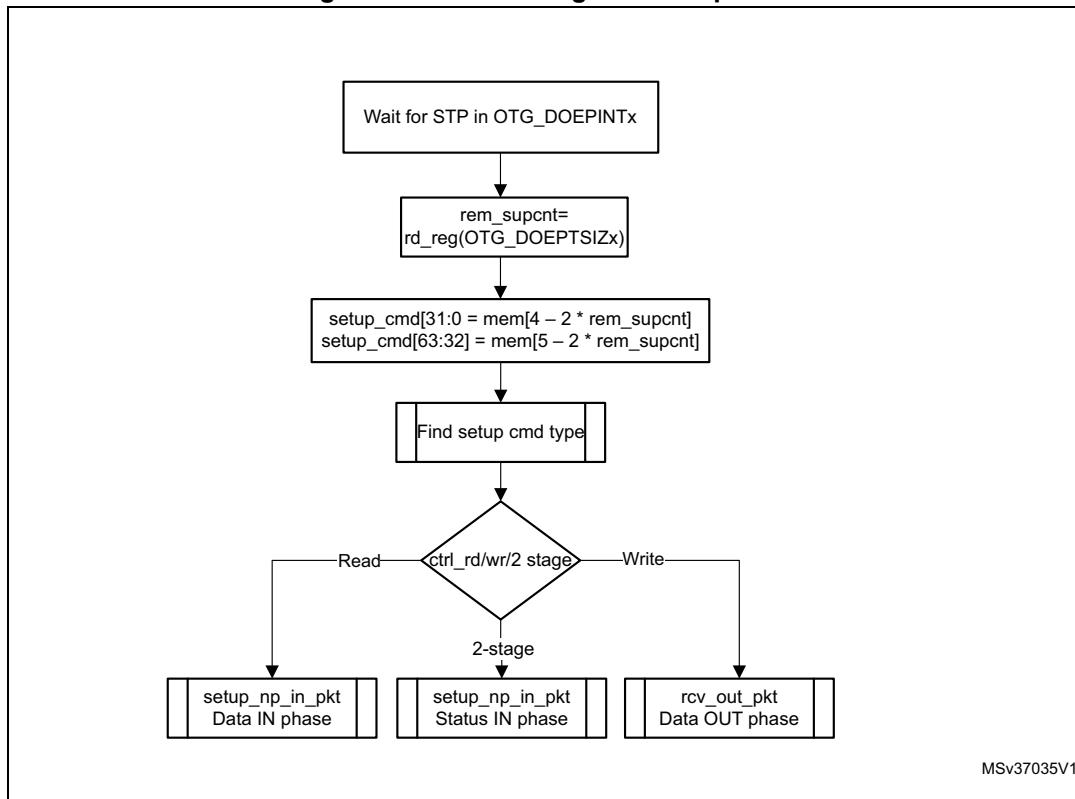
- **Internal data flow**

1. When a SETUP packet is received, the core writes the received data to the receive FIFO, without checking for available space in the receive FIFO and irrespective of the endpoint's NAK and STALL bit settings.
 - The core internally sets the IN NAK and OUT NAK bits for the control IN/OUT endpoints on which the SETUP packet was received.
2. For every SETUP packet received on the USB, 3 words of data are written to the receive FIFO, and the STUPCNT field is decremented by 1.
 - The first word contains control information used internally by the core
 - The second word contains the first 4 bytes of the SETUP command
 - The third word contains the last 4 bytes of the SETUP command
3. When the setup stage changes to a data IN/OUT stage, the core writes an entry (setup stage done word) to the receive FIFO, indicating the completion of the setup stage.
4. On the AHB side, SETUP packets are emptied by the application.
5. When the application pops the setup stage done word from the receive FIFO, the core interrupts the application with an STUP interrupt (OTG_DOEPINTx), indicating it can process the received SETUP packet.
6. The core clears the endpoint enable bit for control OUT endpoints.

- **Application programming sequence**

1. Program the OTG_DOEPTSIZx register.
 - STUPCNT = 3
2. Wait for the RXFLVL interrupt (OTG_GINTSTS) and empty the data packets from the receive FIFO.
3. Assertion of the STUP interrupt (OTG_DOEPINTx) marks a successful completion of the SETUP data transfer.
 - On this interrupt, the application must read the OTG_DOEPTSIZx register to determine the number of SETUP packets received and process the last received SETUP packet.

Figure 452. Processing a SETUP packet



- Handling more than three back-to-back SETUP packets**

Per the USB 2.0 specification, normally, during a SETUP packet error, a host does not send more than three back-to-back SETUP packets to the same endpoint. However, the USB 2.0 specification does not limit the number of back-to-back SETUP packets a host can send to the same endpoint. When this condition occurs, the OTG_FS/OTG_HS controller generates an interrupt (B2BSTUP in OTG_DOEPINTx).

- Setting the global OUT NAK**

Internal data flow:

- When the application sets the Global OUT NAK (SGONAK bit in OTG_DCTL), the core stops writing data, except SETUP packets, to the receive FIFO. Irrespective of the space availability in the receive FIFO, non-isochronous OUT tokens receive a NAK handshake response, and the core ignores isochronous OUT data packets
- The core writes the Global OUT NAK pattern to the receive FIFO. The application must reserve enough receive FIFO space to write this data pattern.
- When the application pops the Global OUT NAK pattern word from the receive FIFO, the core sets the GONAKEFF interrupt (OTG_GINTSTS).
- Once the application detects this interrupt, it can assume that the core is in Global OUT NAK mode. The application can clear this interrupt by clearing the SGONAK bit in OTG_DCTL.

Application programming sequence:

1. To stop receiving any kind of data in the receive FIFO, the application must set the Global OUT NAK bit by programming the following field:
 - SGONAK = 1 in OTG_DCTL
2. Wait for the assertion of the GONAKEFF interrupt in OTG_GINTSTS. When asserted, this interrupt indicates that the core has stopped receiving any type of data except SETUP packets.
3. The application can receive valid OUT packets after it has set SGONAK in OTG_DCTL and before the core asserts the GONAKEFF interrupt (OTG_GINTSTS).
4. The application can temporarily mask this interrupt by writing to the GONAKEFFM bit in the OTG_GINTMSK register.
 - GONAKEFFM = 0 in the OTG_GINTMSK register
5. Whenever the application is ready to exit the Global OUT NAK mode, it must clear the SGONAK bit in OTG_DCTL. This also clears the GONAKEFF interrupt (OTG_GINTSTS).
 - CGONAK = 1 in OTG_DCTL
6. If the application has masked this interrupt earlier, it must be unmasked as follows:
 - GONAKEFFM = 1 in OTG_GINTMSK

- **Disabling an OUT endpoint**

The application must use this sequence to disable an OUT endpoint that it has enabled.

Application programming sequence:

1. Before disabling any OUT endpoint, the application must enable Global OUT NAK mode in the core.
 - SGONAK = 1 in OTG_DCTL
2. Wait for the GONAKEFF interrupt (OTG_GINTSTS)
3. Disable the required OUT endpoint by programming the following fields:
 - EPDIS = 1 in OTG_DOEPCTLx
 - SNAK = 1 in OTG_DOEPCTLx
4. Wait for the EPDISD interrupt (OTG_DOEPINTx), which indicates that the OUT endpoint is completely disabled. When the EPDISD interrupt is asserted, the core also clears the following bits:
 - EPDIS = 0 in OTG_DOEPCTLx
 - EPENA = 0 in OTG_DOEPCTLx
5. The application must clear the Global OUT NAK bit to start receiving data from other non-disabled OUT endpoints.
 - SGONAK = 0 in OTG_DCTL

- **Generic non-isochronous OUT data transfers**

This section describes a regular non-isochronous OUT data transfer (control, bulk, or interrupt).

Application requirements:

1. Before setting up an OUT transfer, the application must allocate a buffer in the memory to accommodate all data to be received as part of the OUT transfer.
2. For OUT transfers, the transfer size field in the endpoint's transfer size register must be a multiple of the maximum packet size of the endpoint, adjusted to the word boundary.
 - transfer size[EPNUM] = $n \times (\text{MPSIZ}[EPNUM] + 4 - (\text{MPSIZ}[EPNUM] \bmod 4))$
 - packet count[EPNUM] = n
 - $n > 0$
3. On any OUT endpoint interrupt, the application must read the endpoint's transfer size register to calculate the size of the payload in the memory. The received payload size can be less than the programmed transfer size.
 - Payload size in memory = application programmed initial transfer size – core updated final transfer size
 - Number of USB packets in which this payload was received = application programmed initial packet count – core updated final packet count

Internal data flow:

1. The application must set the transfer size and packet count fields in the endpoint-specific registers, clear the NAK bit, and enable the endpoint to receive the data.
2. Once the NAK bit is cleared, the core starts receiving data and writes it to the receive FIFO, as long as there is space in the receive FIFO. For every data packet received on the USB, the data packet and its status are written to the receive FIFO. Every packet (maximum packet size or short packet) written to the receive FIFO decrements the packet count field for that endpoint by 1.
 - OUT data packets received with bad data CRC are flushed from the receive FIFO automatically.
 - After sending an ACK for the packet on the USB, the core discards non-isochronous OUT data packets that the host, which cannot detect the ACK, resends. The application does not detect multiple back-to-back data OUT packets on the same endpoint with the same data PID. In this case the packet count is not decremented.
 - If there is no space in the receive FIFO, isochronous or non-isochronous data packets are ignored and not written to the receive FIFO. Additionally, non-isochronous OUT tokens receive a NAK handshake reply.
 - In all the above three cases, the packet count is not decremented because no data are written to the receive FIFO.
3. When the packet count becomes 0 or when a short packet is received on the endpoint, the NAK bit for that endpoint is set. Once the NAK bit is set, the isochronous or non-isochronous data packets are ignored and not written to the receive FIFO, and non-isochronous OUT tokens receive a NAK handshake reply.
4. After the data are written to the receive FIFO, the application reads the data from the receive FIFO and writes it to external memory, one packet at a time per endpoint.
5. At the end of every packet write on the AHB to external memory, the transfer size for the endpoint is decremented by the size of the written packet.
6. The OUT data transfer completed pattern for an OUT endpoint is written to the receive FIFO on one of the following conditions:
 - The transfer size is 0 and the packet count is 0
 - The last OUT data packet written to the receive FIFO is a short packet ($0 \leq \text{packet size} < \text{maximum packet size}$)

7. When either the application pops this entry (OUT data transfer completed), a transfer completed interrupt is generated for the endpoint and the endpoint enable is cleared.

Application programming sequence:

1. Program the OTG_DOEPTSIZx register for the transfer size and the corresponding packet count.
2. Program the OTG_DOEPCTLx register with the endpoint characteristics, and set the EPENA and CNAK bits.
 - EPENA = 1 in OTG_DOEPCTLx
 - CNAK = 1 in OTG_DOEPCTLx
3. Wait for the RXFLVL interrupt (in OTG_GINTSTS) and empty the data packets from the receive FIFO.
 - This step can be repeated many times, depending on the transfer size.
4. Asserting the XFRC interrupt (OTG_DOEPINTx) marks a successful completion of the non-isochronous OUT data transfer.
5. Read the OTG_DOEPTSIZx register to determine the size of the received data payload.

- **Generic isochronous OUT data transfer**

This section describes a regular isochronous OUT data transfer.

Application requirements:

1. All the application requirements for non-isochronous OUT data transfers also apply to isochronous OUT data transfers.
2. For isochronous OUT data transfers, the transfer size and packet count fields must always be set to the number of maximum-packet-size packets that can be received in a single frame and no more. Isochronous OUT data transfers cannot span more than 1 frame.
3. The application must read all isochronous OUT data packets from the receive FIFO (data and status) before the end of the periodic frame (EOPF interrupt in OTG_GINTSTS).
4. To receive data in the following frame, an isochronous OUT endpoint must be enabled after the EOPF (OTG_GINTSTS) and before the SOF (OTG_GINTSTS).

Internal data flow:

1. The internal data flow for isochronous OUT endpoints is the same as that for non-isochronous OUT endpoints, but for a few differences.
2. When an isochronous OUT endpoint is enabled by setting the endpoint enable and clearing the NAK bits, the Even/Odd frame bit must also be set appropriately. The core receives data on an isochronous OUT endpoint in a particular frame only if the following condition is met:
 - EONUM (in OTG_DOEPCTLx) = FNSOF[0] (in OTG_DSTS)
3. When the application completely reads an isochronous OUT data packet (data and status) from the receive FIFO, the core updates the RXDPID field in OTG_DOEPTSIZx with the data PID of the last isochronous OUT data packet read from the receive FIFO.

Application programming sequence:

1. Program the OTG_DOEPTSIZx register for the transfer size and the corresponding packet count
2. Program the OTG_DOEPCTLx register with the endpoint characteristics and set the endpoint enable, ClearNAK, and Even/Odd frame bits.
 - EPENA = 1
 - CNAK = 1
 - EONUM = (0: Even/1: Odd)
3. Wait for the RXFLVL interrupt (in OTG_GINTSTS) and empty the data packets from the receive FIFO
 - This step can be repeated many times, depending on the transfer size.
4. The assertion of the XFRC interrupt (in OTG_DOEPINTx) marks the completion of the isochronous OUT data transfer. This interrupt does not necessarily mean that the data in memory are good.
5. This interrupt cannot always be detected for isochronous OUT transfers. Instead, the application can detect the INCOMPISOOUT interrupt in OTG_GINTSTS.
6. Read the OTG_DOEPTSIZx register to determine the size of the received transfer and to determine the validity of the data received in the frame. The application must treat the data received in memory as valid only if one of the following conditions is met:
 - RXDPID = DATA0 (in OTG_DOEPTSIZx) and the number of USB packets in which this payload was received = 1
 - RXDPID = DATA1 (in OTG_DOEPTSIZx) and the number of USB packets in which this payload was received = 2
 - RXDPID = D2 (in OTG_DOEPTSIZx) and the number of USB packets in which this payload was received = 3[HS]

The number of USB packets in which this payload was received = Application programmed initial packet count – core updated final packet count

The application can discard invalid data packets.

- **Incomplete isochronous OUT data transfers**

This section describes the application programming sequence when isochronous OUT data packets are dropped inside the core.

Internal data flow:

1. For isochronous OUT endpoints, the XFRC interrupt (in OTG_DOEPINTx) may not always be asserted. If the core drops isochronous OUT data packets, the application could fail to detect the XFRC interrupt (OTG_DOEPINTx) under the following circumstances:
 - When the receive FIFO cannot accommodate the complete ISO OUT data packet, the core drops the received ISO OUT data
 - When the isochronous OUT data packet is received with CRC errors
 - When the isochronous OUT token received by the core is corrupted
 - When the application is very slow in reading the data from the receive FIFO
2. When the core detects an end of periodic frame before transfer completion to all isochronous OUT endpoints, it asserts the incomplete isochronous OUT data interrupt (INCOMPISOOUT in OTG_GINTSTS), indicating that an XFRC interrupt (in OTG_DOEPINTx) is not asserted on at least one of the isochronous OUT endpoints. At

At this point, the endpoint with the incomplete transfer remains enabled, but no active transfers remain in progress on this endpoint on the USB.

Application programming sequence:

1. Asserting the INCOMPISOOUT interrupt (OTG_GINTSTS) indicates that in the current frame, at least one isochronous OUT endpoint has an incomplete transfer.
2. If this occurs because isochronous OUT data is not completely emptied from the endpoint, the application must ensure that the application empties all isochronous OUT data (data and status) from the receive FIFO before proceeding.
 - When all data are emptied from the receive FIFO, the application can detect the XFRC interrupt (OTG_DOEPINTx). In this case, the application must re-enable the endpoint to receive isochronous OUT data in the next frame.
3. When it receives an INCOMPISOOUT interrupt (in OTG_GINTSTS), the application must read the control registers of all isochronous OUT endpoints (OTG_DOEPCTLx) to determine which endpoints had an incomplete transfer in the current microframe. An endpoint transfer is incomplete if both the following conditions are met:
 - EONUM bit (in OTG_DOEPCTLx) = FNSOF[0] (in OTG_DSTS)
 - EPENA = 1 (in OTG_DOEPCTLx)
4. The previous step must be performed before the SOF interrupt (in OTG_GINTSTS) is detected, to ensure that the current frame number is not changed.
5. For isochronous OUT endpoints with incomplete transfers, the application must discard the data in the memory and disable the endpoint by setting the EPDIS bit in OTG_DOEPCTLx.
6. Wait for the EPDISD interrupt (in OTG_DOEPINTx) and enable the endpoint to receive new data in the next frame.
 - Because the core can take some time to disable the endpoint, the application may not be able to receive the data in the next frame after receiving bad isochronous data.

- **Stalling a non-isochronous OUT endpoint**

This section describes how the application can stall a non-isochronous endpoint.

1. Put the core in the Global OUT NAK mode.
2. Disable the required endpoint
 - When disabling the endpoint, instead of setting the SNAK bit in OTG_DOEPCTL, set STALL = 1 (in OTG_DOEPCTL).
The STALL bit always takes precedence over the NAK bit.
3. When the application is ready to end the STALL handshake for the endpoint, the STALL bit (in OTG_DOEPCTLx) must be cleared.
4. If the application is setting or clearing a STALL for an endpoint due to a SetFeature.Endpoint Halt or ClearFeature.Endpoint Halt command, the STALL bit must be set or cleared before the application sets up the status stage transfer on the control endpoint.

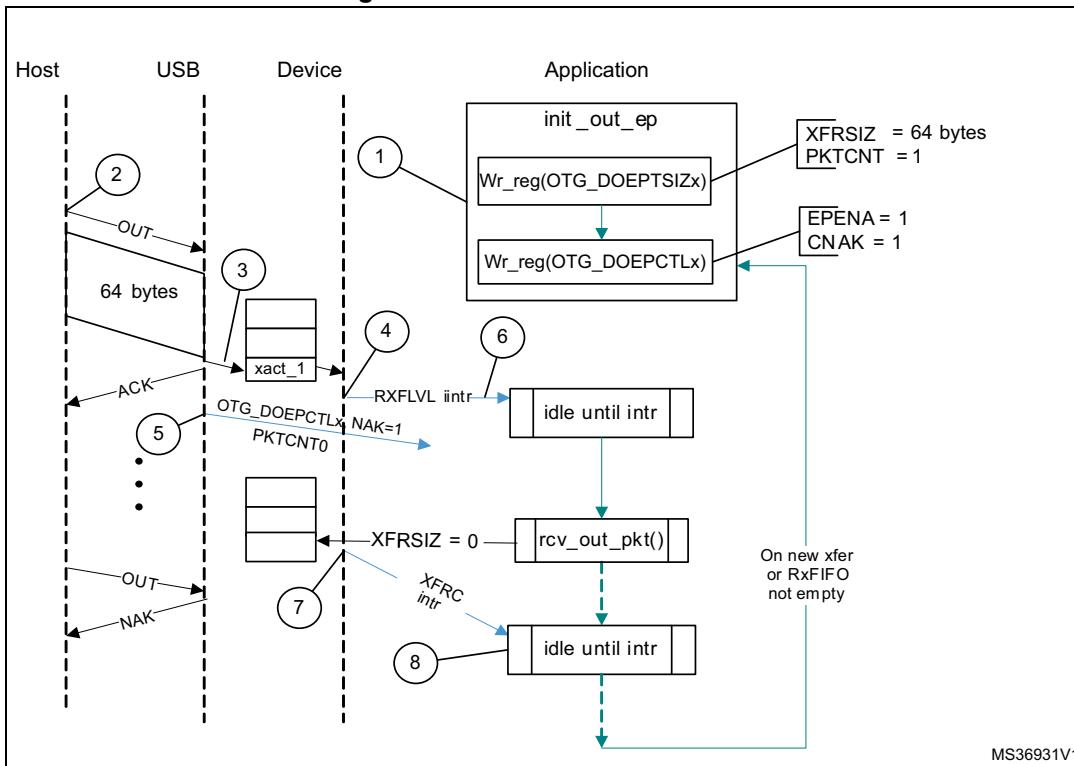
Examples

This section describes and depicts some fundamental transfer types and scenarios.

- Bulk OUT transaction

[Figure 453](#) depicts the reception of a single Bulk OUT data packet from the USB to the AHB and describes the events involved in the process.

Figure 453. Bulk OUT transaction



After a SetConfiguration/SetInterface command, the application initializes all OUT endpoints by setting CNAK = 1 and EPENA = 1 (in OTG_DOEPCTLx), and setting a suitable XFRSIZ and PKTCNT in the OTG_DOEPTSIZx register.

- host attempts to send data (OUT token) to an endpoint.
- When the core receives the OUT token on the USB, it stores the packet in the Rx FIFO because space is available there.
- After writing the complete packet in the Rx FIFO, the core then asserts the RXFLVL interrupt (in OTG_GINTSTS).
- On receiving the PKTCNT number of USB packets, the core internally sets the NAK bit for this endpoint to prevent it from receiving any more packets.
- The application processes the interrupt and reads the data from the Rx FIFO.
- When the application has read all the data (equivalent to XFRSIZ), the core generates an XFRC interrupt (in OTG_DOEPINTx).
- The application processes the interrupt and uses the setting of the XFRC interrupt bit (in OTG_DOEPINTx) to determine that the intended transfer is complete.

IN data transfers

- **Packet write**

This section describes how the application writes data packets to the endpoint FIFO when dedicated transmit FIFOs are enabled.

1. The application can either choose the polling or the interrupt mode.
 - In polling mode, the application monitors the status of the endpoint transmit data FIFO by reading the OTG_DTXFSTSx register, to determine if there is enough space in the data FIFO.
 - In interrupt mode, the application waits for the TXFE interrupt (in OTG_DIEPINTx) and then reads the OTG_DTXFSTSx register, to determine if there is enough space in the data FIFO.
 - To write a single non-zero length data packet, there must be space to write the entire packet in the data FIFO.
 - To write zero length packet, the application must not look at the FIFO space.
2. Using one of the above mentioned methods, when the application determines that there is enough space to write a transmit packet, the application must first write into the endpoint control register, before writing the data into the data FIFO. Typically, the application, must do a read modify write on the OTG_DIEPCTLx register to avoid modifying the contents of the register, except for setting the endpoint enable bit.

The application can write multiple packets for the same endpoint into the transmit FIFO, if space is available. For periodic IN endpoints, the application must write packets only for one microframe. It can write packets for the next periodic transaction only after getting transfer complete for the previous transaction.

- **Setting IN endpoint NAK**

Internal data flow:

1. When the application sets the IN NAK for a particular endpoint, the core stops transmitting data on the endpoint, irrespective of data availability in the endpoint's transmit FIFO.
2. Non-isochronous IN tokens receive a NAK handshake reply
 - Isochronous IN tokens receive a zero-data-length packet reply
3. The core asserts the INEPNE (IN endpoint NAK effective) interrupt in OTG_DIEPINTx in response to the SNAK bit in OTG_DIEPCTLx.
4. Once this interrupt is seen by the application, the application can assume that the endpoint is in IN NAK mode. This interrupt can be cleared by the application by setting the CNAK bit in OTG_DIEPCTLx.

Application programming sequence:

1. To stop transmitting any data on a particular IN endpoint, the application must set the IN NAK bit. To set this bit, the following field must be programmed.
 - SNAK = 1 in OTG_DIEPCTLx
2. Wait for assertion of the INEPNE interrupt in OTG_DIEPINTx. This interrupt indicates that the core has stopped transmitting data on the endpoint.
3. The core can transmit valid IN data on the endpoint after the application has set the NAK bit, but before the assertion of the NAK Effective interrupt.
4. The application can mask this interrupt temporarily by writing to the INEPNEM bit in OTG_DIEPMSK.
 - INEPNEM = 0 in OTG_DIEPMSK
5. To exit endpoint NAK mode, the application must clear the NAK status bit (NAKSTS) in OTG_DIEPCTLx. This also clears the INEPNE interrupt (in OTG_DIEPINTx).

- CNAK = 1 in OTG_DIEPCTLx
- 6. If the application masked this interrupt earlier, it must be unmasked as follows:
 - INEPNEM = 1 in OTG_DIEPMSK

- **IN endpoint disable**

Use the following sequence to disable a specific IN endpoint that has been previously enabled.

Application programming sequence:

1. The application must stop writing data on the AHB for the IN endpoint to be disabled.
2. The application must set the endpoint in NAK mode.
 - SNAK = 1 in OTG_DIEPCTLx
3. Wait for the INEPNE interrupt in OTG_DIEPINTx.
4. Set the following bits in the OTG_DIEPCTLx register for the endpoint that must be disabled.
 - EPDIS = 1 in OTG_DIEPCTLx
 - SNAK = 1 in OTG_DIEPCTLx
5. Assertion of the EPDISD interrupt in OTG_DIEPINTx indicates that the core has completely disabled the specified endpoint. Along with the assertion of the interrupt, the core also clears the following bits:
 - EPENA = 0 in OTG_DIEPCTLx
 - EPDIS = 0 in OTG_DIEPCTLx
6. The application must read the OTG_DIEPTSIZx register for the periodic IN EP, to calculate how much data on the endpoint were transmitted on the USB.
7. The application must flush the data in the endpoint transmit FIFO, by setting the following fields in the OTG_GRSTCTL register:
 - TXFNUM (in OTG_GRSTCTL) = Endpoint transmit FIFO number
 - TXFFLSH in (OTG_GRSTCTL) = 1

The application must poll the OTG_GRSTCTL register, until the TXFFLSH bit is cleared by the core, which indicates the end of flush operation. To transmit new data on this endpoint, the application can re-enable the endpoint at a later point.

- **Generic non-periodic IN data transfers**

Application requirements:

1. Before setting up an IN transfer, the application must ensure that all data to be transmitted as part of the IN transfer are part of a single buffer.
2. For IN transfers, the transfer size field in the endpoint transfer size register denotes a payload that constitutes multiple maximum-packet-size packets and a single short packet. This short packet is transmitted at the end of the transfer.
 - To transmit a few maximum-packet-size packets and a short packet at the end of the transfer:

Transfer size[EPNUM] = $x \times \text{MPSIZ}[EPNUM] + sp$
 If ($sp > 0$), then packet count[EPNUM] = $x + 1$.
 Otherwise, packet count[EPNUM] = x
 - To transmit a single zero-length data packet:

Transfer size[EPNUM] = 0

Packet count[EPNUM] = 1

- To transmit a few maximum-packet-size packets and a zero-length data packet at the end of the transfer, the application must split the transfer into two parts. The first sends maximum-packet-size data packets and the second sends the zero-length data packet alone.

First transfer: transfer size[EPNUM] = $x \times \text{MPSIZ}[\text{epnum}]$; packet count = n ;

Second transfer: transfer size[EPNUM] = 0; packet count = 1;

3. Once an endpoint is enabled for data transfers, the core updates the transfer size register. At the end of the IN transfer, the application must read the transfer size register to determine how much data posted in the transmit FIFO have already been sent on the USB.
4. Data fetched into transmit FIFO = Application-programmed initial transfer size – core-updated final transfer size
 - Data transmitted on USB = (application-programmed initial packet count – core updated final packet count) $\times \text{MPSIZ}[\text{EPNUM}]$
 - Data yet to be transmitted on USB = (Application-programmed initial transfer size – data transmitted on USB)

Internal data flow:

1. The application must set the transfer size and packet count fields in the endpoint-specific registers and enable the endpoint to transmit the data.
2. The application must also write the required data to the transmit FIFO for the endpoint.
3. Every time a packet is written into the transmit FIFO by the application, the transfer size for that endpoint is decremented by the packet size. The data is fetched from the memory by the application, until the transfer size for the endpoint becomes 0. After writing the data into the FIFO, the “number of packets in FIFO” count is incremented (this is a 3-bit count, internally maintained by the core for each IN endpoint transmit FIFO. The maximum number of packets maintained by the core at any time in an IN endpoint FIFO is eight). For zero-length packets, a separate flag is set for each FIFO, without any data in the FIFO.
4. Once the data are written to the transmit FIFO, the core reads them out upon receiving an IN token. For every non-isochronous IN data packet transmitted with an ACK handshake, the packet count for the endpoint is decremented by one, until the packet count is zero. The packet count is not decremented on a timeout.
5. For zero length packets (indicated by an internal zero length flag), the core sends out a zero-length packet for the IN token and decrements the packet count field.
6. If there are no data in the FIFO for a received IN token and the packet count field for that endpoint is zero, the core generates an “IN token received when Tx FIFO is empty” (ITTXFE) interrupt for the endpoint, provided that the endpoint NAK bit is not set. The core responds with a NAK handshake for non-isochronous endpoints on the USB.
7. The core internally rewinds the FIFO pointers and no timeout interrupt is generated.
8. When the transfer size is 0 and the packet count is 0, the transfer complete (XFRC) interrupt for the endpoint is generated and the endpoint enable is cleared.

Application programming sequence:

1. Program the OTG_DIEPTSI x register with the transfer size and corresponding packet count.
2. Program the OTG_DIEPCTL x register with the endpoint characteristics and set the CNAK and EPENA (endpoint enable) bits.
3. When transmitting non-zero length data packet, the application must poll the OTG_DTXFSTS x register (where x is the FIFO number associated with that endpoint) to determine whether there is enough space in the data FIFO. The application can optionally use TXFE (in OTG_DIEPINT x) before writing the data.

- **Generic periodic IN data transfers**

This section describes a typical periodic IN data transfer.

Application requirements:

1. Application requirements 1, 2, 3, and 4 of [Generic non-periodic IN data transfers on page 1345](#) also apply to periodic IN data transfers, except for a slight modification of requirement 2.
 - The application can only transmit multiples of maximum-packet-size data packets or multiples of maximum-packet-size packets, plus a short packet at the end. To transmit a few maximum-packet-size packets and a short packet at the end of the transfer, the following conditions must be met:

$$\text{transfer size[EPNUM]} = x \times \text{MPSIZ[EPNUM]} + \text{sp}$$

(where x is an integer ≥ 0 , and $0 \leq \text{sp} < \text{MPSIZ[EPNUM]}$)

If ($\text{sp} > 0$), $\text{packet count[EPNUM]} = x + 1$
 Otherwise, $\text{packet count[EPNUM]} = x$;
 $\text{MCNT[EPNUM]} = \text{packet count[EPNUM]}$
 - The application cannot transmit a zero-length data packet at the end of a transfer. It can transmit a single zero-length data packet by itself. To transmit a single zero-length data packet:
 - $\text{transfer size[EPNUM]} = 0$
 - $\text{packet count[EPNUM]} = 1$
 - $\text{MCNT[EPNUM]} = \text{packet count[EPNUM]}$
2. The application can only schedule data transfers one frame at a time.
 - $(\text{MCNT} - 1) \times \text{MPSIZ} \leq \text{XFERSIZ} \leq \text{MCNT} \times \text{MPSIZ}$
 - $\text{PKTCNT} = \text{MCNT}$ (in OTG_DIEPTSI x)
 - If $\text{XFERSIZ} < \text{MCNT} \times \text{MPSIZ}$, the last data packet of the transfer is a short packet.
 - Note that: MCNT is in OTG_DIEPTSI x , MPSIZ is in OTG_DIEPCTL x , PKTCNT is in OTG_DIEPTSI x and XFERSIZ is in OTG_DIEPTSI x
3. The complete data to be transmitted in the frame must be written into the transmit FIFO by the application, before the IN token is received. Even when 1 word of the data to be transmitted per frame is missing in the transmit FIFO when the IN token is received, the core behaves as when the FIFO is empty. When the transmit FIFO is empty:
 - A zero data length packet would be transmitted on the USB for isochronous IN endpoints
 - A NAK handshake would be transmitted on the USB for interrupt IN endpoints

Internal data flow:

1. The application must set the transfer size and packet count fields in the endpoint-specific registers and enable the endpoint to transmit the data.
2. The application must also write the required data to the associated transmit FIFO for the endpoint.
3. Every time the application writes a packet to the transmit FIFO, the transfer size for that endpoint is decremented by the packet size. The data are fetched from application memory until the transfer size for the endpoint becomes 0.
4. When an IN token is received for a periodic endpoint, the core transmits the data in the FIFO, if available. If the complete data payload (complete packet, in dedicated FIFO mode) for the frame is not present in the FIFO, then the core generates an IN token received when Tx FIFO empty interrupt for the endpoint.
 - A zero-length data packet is transmitted on the USB for isochronous IN endpoints
 - A NAK handshake is transmitted on the USB for interrupt IN endpoints
5. The packet count for the endpoint is decremented by 1 under the following conditions:
 - For isochronous endpoints, when a zero- or non-zero-length data packet is transmitted
 - For interrupt endpoints, when an ACK handshake is transmitted
 - When the transfer size and packet count are both 0, the transfer completed interrupt for the endpoint is generated and the endpoint enable is cleared.
6. At the “Periodic frame Interval” (controlled by PFIVL in OTG_DCFG), when the core finds non-empty any of the isochronous IN endpoint FIFOs scheduled for the current frame non-empty, the core generates an IISOIXFR interrupt in OTG_GINTSTS.

Application programming sequence:

1. Program the OTG_DIEPCTLx register with the endpoint characteristics and set the CNAK and EPENA bits.
2. Write the data to be transmitted in the next frame to the transmit FIFO.
3. Asserting the ITTXFE interrupt (in OTG_DIEPINTx) indicates that the application has not yet written all data to be transmitted to the transmit FIFO.
4. If the interrupt endpoint is already enabled when this interrupt is detected, ignore the interrupt. If it is not enabled, enable the endpoint so that the data can be transmitted on the next IN token attempt.
5. Asserting the XFRC interrupt (in OTG_DIEPINTx) with no ITTXFE interrupt in OTG_DIEPINTx indicates the successful completion of an isochronous IN transfer. A read to the OTG_DIEPTSI_Zx register must give transfer size = 0 and packet count = 0, indicating all data were transmitted on the USB.
6. Asserting the XFRC interrupt (in OTG_DIEPINTx), with or without the ITTXFE interrupt (in OTG_DIEPINTx), indicates the successful completion of an interrupt IN transfer. A read to the OTG_DIEPTSI_Zx register must give transfer size = 0 and packet count = 0, indicating all data were transmitted on the USB.
7. Asserting the incomplete isochronous IN transfer (IISOIXFR) interrupt in OTG_GINTSTS with none of the aforementioned interrupts indicates the core did not receive at least 1 periodic IN token in the current frame.

- **Incomplete isochronous IN data transfers**

This section describes what the application must do on an incomplete isochronous IN data transfer.

Internal data flow:

1. An isochronous IN transfer is treated as incomplete in one of the following conditions:
 - a) The core receives a corrupted isochronous IN token on at least one isochronous IN endpoint. In this case, the application detects an incomplete isochronous IN transfer interrupt (IISOIXFR in OTG_GINTSTS).
 - b) The application is slow to write the complete data payload to the transmit FIFO and an IN token is received before the complete data payload is written to the FIFO. In this case, the application detects an IN token received when Tx FIFO empty interrupt in OTG_DIEPINTx. The application can ignore this interrupt, as it eventually results in an incomplete isochronous IN transfer interrupt (IISOIXFR in OTG_GINTSTS) at the end of periodic frame.
The core transmits a zero-length data packet on the USB in response to the received IN token.
2. The application must stop writing the data payload to the transmit FIFO as soon as possible.
3. The application must set the NAK bit and the disable bit for the endpoint.
4. The core disables the endpoint, clears the disable bit, and asserts the endpoint disable interrupt for the endpoint.

Application programming sequence:

1. The application can ignore the IN token received when Tx FIFO empty interrupt in OTG_DIEPINTx on any isochronous IN endpoint, as it eventually results in an incomplete isochronous IN transfer interrupt (in OTG_GINTSTS).
2. Assertion of the incomplete isochronous IN transfer interrupt (in OTG_GINTSTS) indicates an incomplete isochronous IN transfer on at least one of the isochronous IN endpoints.
3. The application must read the endpoint control register for all isochronous IN endpoints to detect endpoints with incomplete IN data transfers.
4. The application must stop writing data to the Periodic Transmit FIFOs associated with these endpoints on the AHB.
5. Program the following fields in the OTG_DIEPCTLx register to disable the endpoint:
 - SNAK = 1 in OTG_DIEPCTLx
 - EPDIS = 1 in OTG_DIEPCTLx
6. The assertion of the endpoint disabled interrupt in OTG_DIEPINTx indicates that the core has disabled the endpoint.
 - At this point, the application must flush the data in the associated transmit FIFO or overwrite the existing data in the FIFO by enabling the endpoint for a new transfer in the next microframe. To flush the data, the application must use the OTG_GRSTCTL register.

- **Stalling non-isochronous IN endpoints**

This section describes how the application can stall a non-isochronous endpoint.

Application programming sequence:

1. Disable the IN endpoint to be stalled. Set the STALL bit as well.
2. EPDIS = 1 in OTG_DIEPCTLx, when the endpoint is already enabled
 - STALL = 1 in OTG_DIEPCTLx
 - The STALL bit always takes precedence over the NAK bit
3. Assertion of the endpoint disabled interrupt (in OTG_DIEPINTx) indicates to the application that the core has disabled the specified endpoint.
4. The application must flush the non-periodic or periodic transmit FIFO, depending on the endpoint type. In case of a non-periodic endpoint, the application must re-enable the other non-periodic endpoints that do not need to be stalled, to transmit data.
5. Whenever the application is ready to end the STALL handshake for the endpoint, the STALL bit must be cleared in OTG_DIEPCTLx.
6. If the application sets or clears a STALL bit for an endpoint due to a SetFeature.Endpoint Halt command or ClearFeature.Endpoint Halt command, the STALL bit must be set or cleared before the application sets up the status stage transfer on the control endpoint.

Special case: stalling the control OUT endpoint

The core must stall IN/OUT tokens if, during the data stage of a control transfer, the host sends more IN/OUT tokens than are specified in the SETUP packet. In this case, the application must enable the ITTXFE interrupt in OTG_DIEPINTx and the OTEPDIS interrupt in OTG_DOEPINTx during the data stage of the control transfer, after the core has transferred the amount of data specified in the SETUP packet. Then, when the application receives this interrupt, it must set the STALL bit in the corresponding endpoint control register, and clear this interrupt.

32.16.7 Worst case response time

When the OTG_FS/OTG_HS controller acts as a device, there is a worst case response time for any tokens that follow an isochronous OUT. This worst case response time depends on the AHB clock frequency.

The core registers are in the AHB domain, and the core does not accept another token before updating these register values. The worst case is for any token following an isochronous OUT, because for an isochronous transaction, there is no handshake and the next token could come sooner. This worst case value is 7 PHY clocks when the AHB clock is the same as the PHY clock. When the AHB clock is faster, this value is smaller.

If this worst case condition occurs, the core responds to bulk/interrupt tokens with a NAK and drops isochronous and SETUP tokens. The host interprets this as a timeout condition for SETUP and retries the SETUP packet. For isochronous transfers, the Incomplete isochronous IN transfer interrupt (IISOIXFR) and Incomplete isochronous OUT transfer interrupt (IISOOXFR) inform the application that isochronous IN/OUT packets were dropped.

Choosing the value of TRDT in OTG_GUSBCFG

The value in TRDT (OTG_GUSBCFG) is the time it takes for the MAC, in terms of PHY clocks after it has received an IN token, to get the FIFO status, and thus the first data from the PFC block. This time involves the synchronization delay between the PHY and AHB clocks. The worst case delay for this is when the AHB clock is the same as the PHY clock. In this case, the delay is 5 clocks.

Once the MAC receives an IN token, this information (token received) is synchronized to the AHB clock by the PFC (the PFC runs on the AHB clock). The PFC then reads the data from the SPRAM and writes them into the dual clock source buffer. The MAC then reads the data out of the source buffer (4 deep).

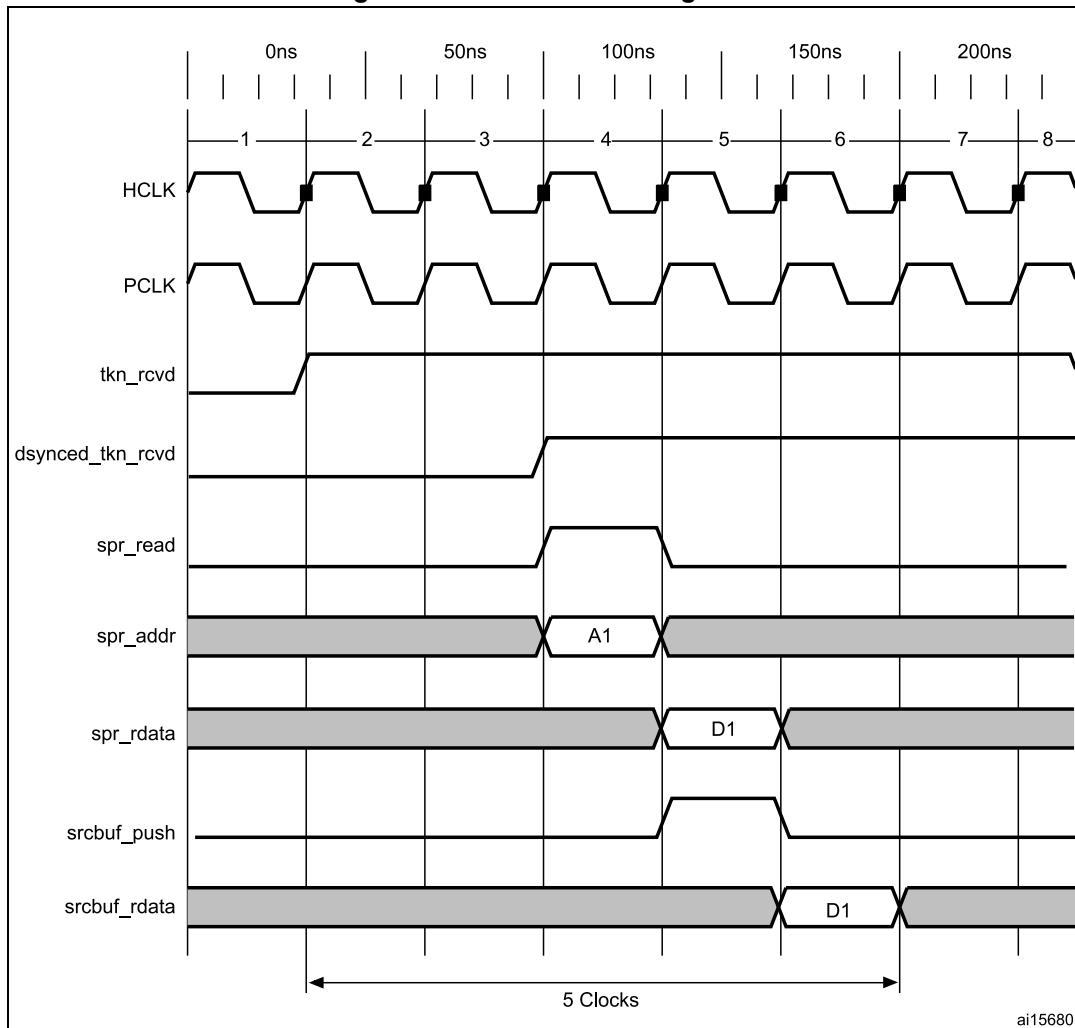
If the AHB is running at a higher frequency than the PHY, the application can use a smaller value for TRDT (in OTG_GUSBCFG).

Figure 454 has the following signals:

- tkn_rcvd: Token received information from MAC to PFC
- dynced_tkn_rcvd: Doubled sync tkn_rcvd, from PCLK to HCLK domain
- spr_read: Read to SPRAM
- spr_addr: Address to SPRAM
- spr_rdata: Read data from SPRAM
- srcbuf_push: Push to the source buffer
- srcbuf_rdata: Read data from the source buffer. Data seen by MAC

To calculate the value of TRDT, refer to [Table 232: TRDT values \(FS\)](#) or [Table 233: TRDT values \(HS\)](#).

Figure 454. TRDT max timing case



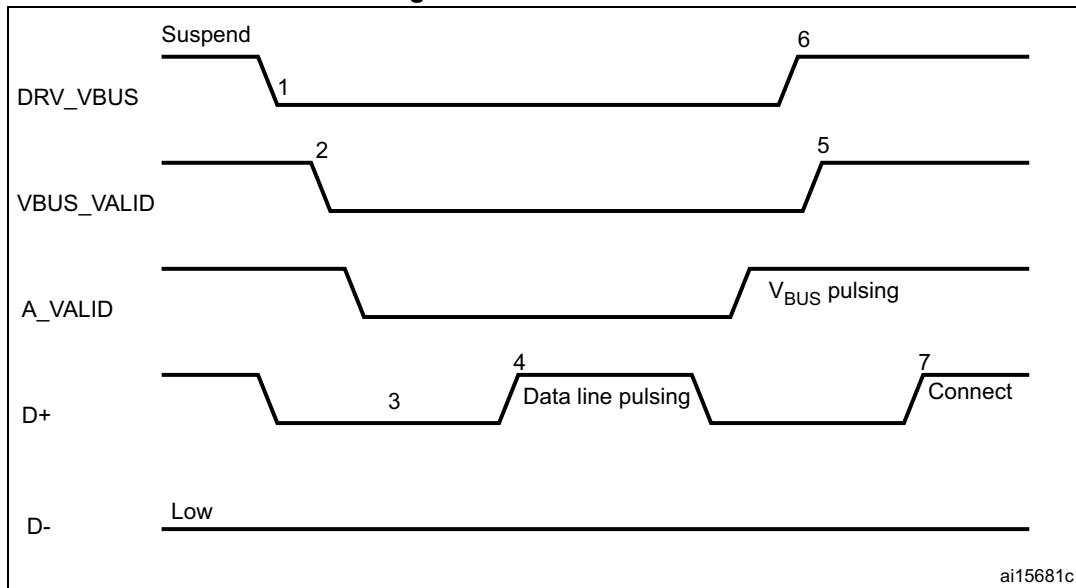
32.16.8 OTG programming model

The OTG_FS/OTG_HS controller is an OTG device supporting HNP and SRP. When the core is connected to an “A” plug, it is referred to as an A-device. When the core is connected to a “B” plug it is referred to as a B-device. In host mode, the OTG_FS/OTG_HS controller turns off V_{BUS} to conserve power. SRP is a method by which the B-device signals the A-device to turn on V_{BUS} power. A device must perform both data-line pulsing and V_{BUS} pulsing, but a host can detect either data-line pulsing or V_{BUS} pulsing for SRP. HNP is a method by which the B-device negotiates and switches to host role. In Negotiated mode after HNP, the B-device suspends the bus and reverts to the device role.

A-device session request protocol

The application must set the SRP-capable bit in the core USB configuration register. This enables the OTG_FS/OTG_HS controller to detect SRP as an A-device.

Figure 455. A-device SRP



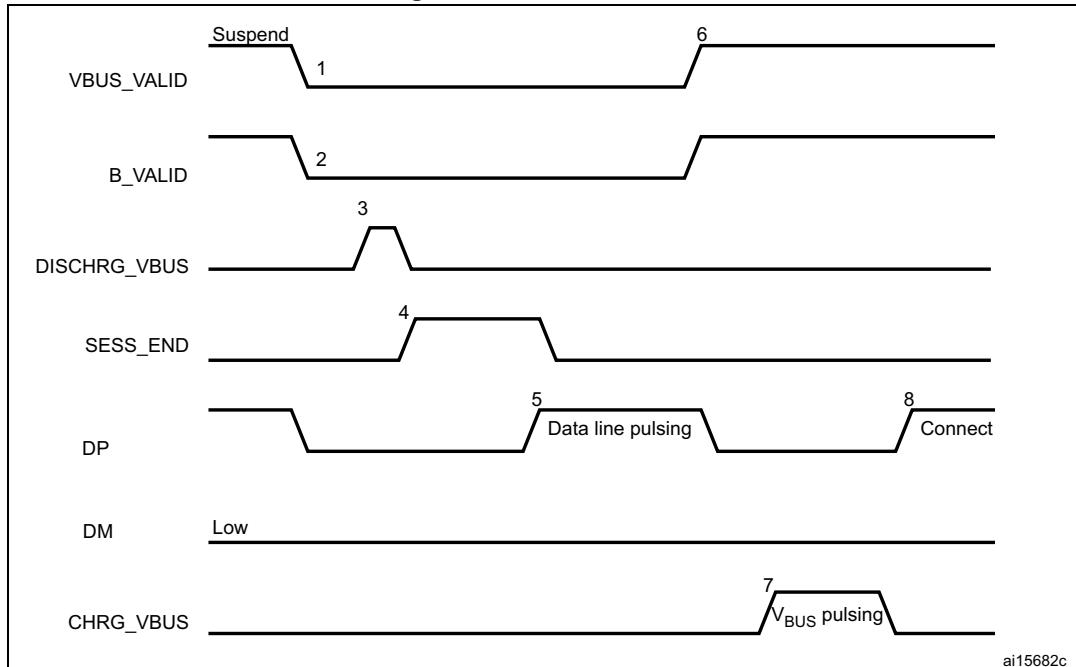
The following points refer and describe the signal numeration shown in the [Figure 455](#):

1. To save power, the application suspends and turns off port power when the bus is idle by writing the port suspend and port power bits in the host port control and status register.
2. PHY indicates port power off by deasserting the V_{BUS}_VALID signal.
3. The device must detect SE0 for at least 2 ms to start SRP when V_{BUS} power is off.
4. To initiate SRP, the device turns on its data line pull-up resistor for 5 to 10 ms. The OTG_FS/OTG_HS controller detects data-line pulsing.
5. The device drives V_{BUS} above the A-device session valid (2.0 V minimum) for V_{BUS} pulsing.
The OTG_FS/OTG_HS controller interrupts the application on detecting SRP. The session request detected bit is set in Global interrupt status register (SRQINT set in OTG_GINTSTS).
6. The application must service the session request detected interrupt and turn on the port power bit by writing the port power bit in the host port control and status register. The PHY indicates port power-on by asserting the V_{BUS}_VALID signal.
7. When the USB is powered, the device connects, completing the SRP process.

B-device session request protocol

The application must set the SRP-capable bit in the core USB configuration register. This enables the OTG_FS/OTG_HS controller to initiate SRP as a B-device. SRP is a means by which the OTG_FS/OTG_HS controller can request a new session from the host.

Figure 456. B-device SRP



1. VBUS_VALID = V_{BUS} valid signal from PHY
 B_VALID = B-peripheral valid session to PHY
 DISCHRG_VBUS = discharge signal to PHY
 SESS_END = session end signal to PHY
 CHRG_VBUS = charge V_{BUS} signal to PHY
 DP = Data plus line
 DM = Data minus line

The following points refer and describe the signal numeration shown in the [Figure 456](#):

1. To save power, the host suspends and turns off port power when the bus is idle. The OTG_FS/OTG_HS controller sets the early suspend bit in the core interrupt register after 3 ms of bus idleness. Following this, the OTG_FS/OTG_HS controller sets the USB suspend bit in the core interrupt register. The OTG_FS/OTG_HS controller informs the PHY to discharge V_{BUS} .
2. The PHY indicates the session's end to the device. This is the initial condition for SRP. The OTG_FS/OTG_HS controller requires 2 ms of SE0 before initiating SRP. For a USB 1.1 full-speed serial transceiver, the application must wait until V_{BUS} discharges to 0.2 V after BSVLD (in OTG_GOTGCTL) is deasserted. This discharge time can be obtained from the transceiver vendor and varies from one transceiver to another.
3. The OTG_FS/OTG_HS core informs the PHY to speed up V_{BUS} discharge.
4. The application initiates SRP by writing the session request bit in the OTG control and status register. The OTG_FS/OTG_HS controller performs data-line pulsing followed by V_{BUS} pulsing.
5. The host detects SRP from either the data-line or V_{BUS} pulsing, and turns on V_{BUS} . The PHY indicates V_{BUS} power-on to the device.
6. The OTG_FS/OTG_HS controller performs V_{BUS} pulsing. The host starts a new session by turning on V_{BUS} , indicating SRP success. The OTG_FS/OTG_HS controller interrupts the application by setting the session request

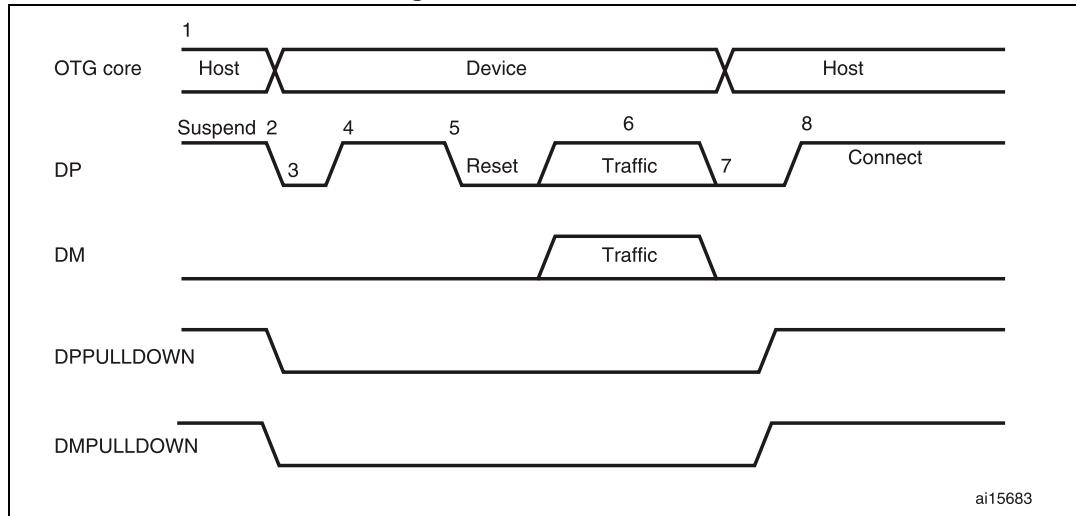
success status change bit in the OTG interrupt status register. The application reads the session request success bit in the OTG control and status register.

7. When the USB is powered, the OTG_FS/OTG_HS controller connects, completing the SRP process.

A-device host negotiation protocol

HNP switches the USB host role from the A-device to the B-device. The application must set the HNP-capable bit in the core USB configuration register to enable the OTG_FS/OTG_HS controller to perform HNP as an A-device.

Figure 457. A-device HNP



1. DPPULLDOWN = signal from core to PHY to enable/disable the pull-down on the DP line inside the PHY.
DMPULLDOWN = signal from core to PHY to enable/disable the pull-down on the DM line inside the PHY.

The following points refer and describe the signal numeration shown in the [Figure 457](#):

1. The OTG_FS/OTG_HS controller sends the B-device a SetFeature b_hnp_enable descriptor to enable HNP support. The B-device's ACK response indicates that the B-device supports HNP. The application must set host Set HNP enable bit in the OTG

control and status register to indicate to the OTG_FS/OTG_HS controller that the B-device supports HNP.

2. When it has finished using the bus, the application suspends by writing the port suspend bit in the host port control and status register.
3. When the B-device observes a USB suspend, it disconnects, indicating the initial condition for HNP. The B-device initiates HNP only when it must switch to the host role; otherwise, the bus continues to be suspended.

The OTG_FS/OTG_HS controller sets the host negotiation detected interrupt in the OTG interrupt status register, indicating the start of HNP.

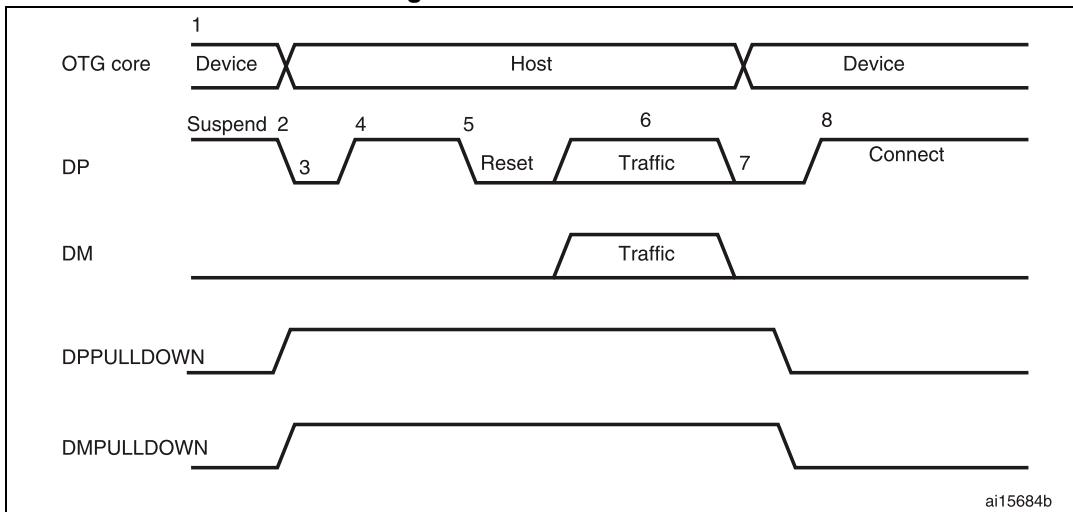
The OTG_FS/OTG_HS controller deasserts the DM pull down and DM pull down in the PHY to indicate a device role. The PHY enables the OTG_DP pull-up resistor to indicate a connect for B-device.

The application must read the current mode bit in the OTG control and status register to determine device mode operation.

4. The B-device detects the connection, issues a USB reset, and enumerates the OTG_FS/OTG_HS controller for data traffic.
 5. The B-device continues the host role, initiating traffic, and suspends the bus when done.
- The OTG_FS/OTG_HS controller sets the early suspend bit in the core interrupt register after 3 ms of bus idleness. Following this, the OTG_FS/OTG_HS controller sets the USB suspend bit in the core interrupt register.
6. In Negotiated mode, the OTG_FS/OTG_HS controller detects the suspend, disconnects, and switches back to the host role. The OTG_FS/OTG_HS controller asserts the DM pull down and DM pull down in the PHY to indicate its assumption of the host role.
 7. The OTG_FS/OTG_HS controller sets the connector ID status change interrupt in the OTG interrupt status register. The application must read the connector ID status in the OTG control and status register to determine the OTG_FS/OTG_HS controller operation as an A-device. This indicates the completion of HNP to the application. The application must read the Current mode bit in the OTG control and status register to determine host mode operation.
 8. The B-device connects, completing the HNP process.

B-device host negotiation protocol

HNP switches the USB host role from B-device to A-device. The application must set the HNP-capable bit in the core USB configuration register to enable the OTG_FS/OTG_HS controller to perform HNP as a B-device.

Figure 458. B-device HNP

1. DPPULLDOWN = signal from core to PHY to enable/disable the pull-down on the DP line inside the PHY.
DMPULLDOWN = signal from core to PHY to enable/disable the pull-down on the DM line inside the PHY.

The following points refer and describe the signal numeration shown in the [Figure 458](#):

1. The A-device sends the SetFeature b_hnp_enable descriptor to enable HNP support. The OTG_FS/OTG_HS controller's ACK response indicates that it supports HNP. The application must set the device HNP enable bit in the OTG control and status register to indicate HNP support.
The application sets the HNP request bit in the OTG control and status register to indicate to the OTG_FS/OTG_HS controller to initiate HNP.
2. When it has finished using the bus, the A-device suspends by writing the port suspend bit in the host port control and status register.
The OTG_FS/OTG_HS controller sets the Early suspend bit in the core interrupt register after 3 ms of bus idleness. Following this, the OTG_FS/OTG_HS controller sets the USB suspend bit in the core interrupt register.
The OTG_FS/OTG_HS controller disconnects and the A-device detects SE0 on the bus, indicating HNP. The OTG_FS/OTG_HS controller asserts the DP pull down and DM pull down in the PHY to indicate its assumption of the host role.
The A-device responds by activating its OTG_DP pull-up resistor within 3 ms of detecting SE0. The OTG_FS/OTG_HS controller detects this as a connect.
The OTG_FS/OTG_HS controller sets the host negotiation success status change interrupt in the OTG interrupt status register, indicating the HNP status. The application must read the host negotiation success bit in the OTG control and status register to

determine host negotiation success. The application must read the current Mode bit in the core interrupt register (OTG_GINTSTS) to determine host mode operation.

3. The application sets the reset bit (PRST in OTG_HPORT) and the OTG_FS/OTG_HS controller issues a USB reset and enumerates the A-device for data traffic.
4. The OTG_FS/OTG_HS controller continues the host role of initiating traffic, and when done, suspends the bus by writing the port suspend bit in the host port control and status register.
5. In Negotiated mode, when the A-device detects a suspend, it disconnects and switches back to the host role. The OTG_FS/OTG_HS controller deasserts the DP pull down and DM pull down in the PHY to indicate the assumption of the device role.
6. The application must read the current mode bit in the core interrupt (OTG_GINTSTS) register to determine the host mode operation.
7. The OTG_FS/OTG_HS controller connects, completing the HNP process.

33 USB PHY controller (USBPHYC) available on the STM32F7x3xx and STM32F730xx devices only

33.1 USBPHYC introduction

The USBPHYC enables the control and observation of a High Speed USB PHY's configuration and status, as well as the control/monitoring of its dedicated LDO.

33.2 USBPHYC main features

The USBPHYC is configured to directly control appropriate functions available on the High Speed PHY:

- The High Speed PHY power supply regulator control and monitoring
- The configuration of the PLL to adapt to different input frequencies (in a defined list)
- Many fine tuning performance controls

33.3 USBPHYC functional description

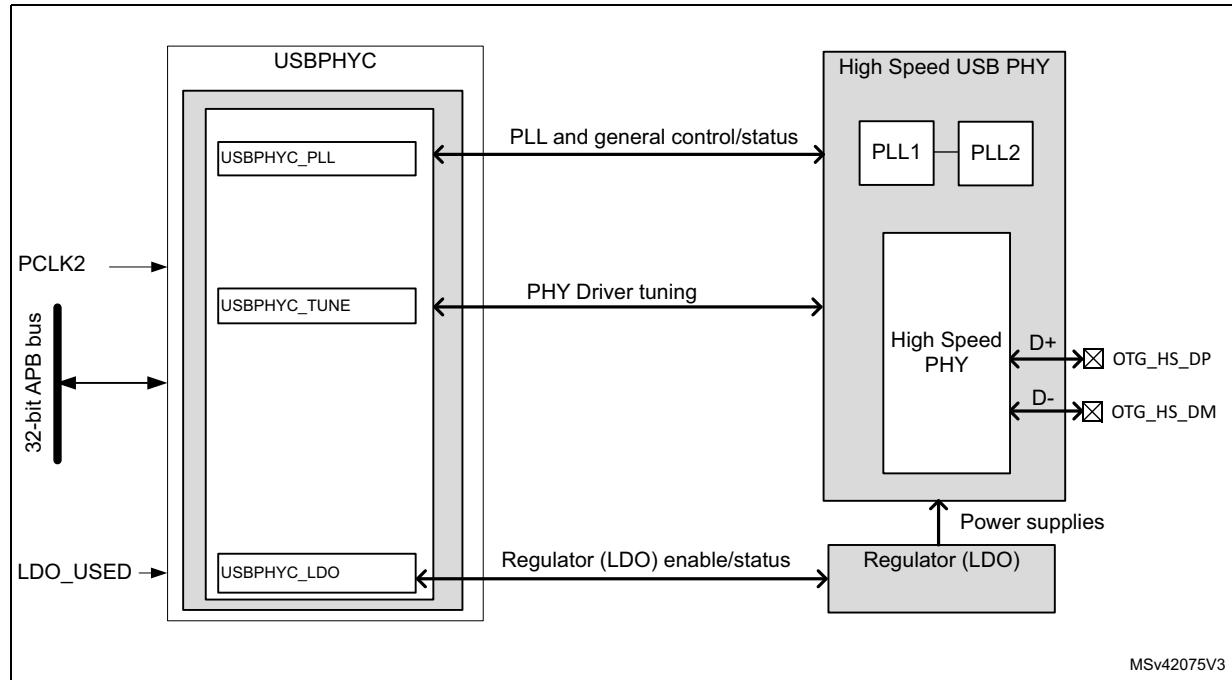
The USBPHYC works as a controller, enabling an easy access to the numerous functions available on the High Speed PHY with an integrated PLL system:

- Sets the PLL values for the PHY
- Controls and monitors the USB PHY's LDO
- Sets other controls (and monitors) on the PHY

33.3.1 USBPHYC block diagram

The USBPHYC block diagram illustrates the connections to the peripheral bus, to the different sections of the PHY including the integrated PLL sub-system and also to the associated LDO.

Figure 459. USBPHYC block diagram



33.3.2 USBPHYC reset and clocks

As an APB peripheral, the only clock used directly in the USBPHYC is the PCLK of the APB bus interface (PCLK2). The APB bus reset resets the registers.

33.4 USBPHYC register interface

33.4.1 USBPHYC PLL1 control register (USBPHYC_PLL1)

Address offset: 0x000

Reset value: 0x0000 0000

This register is used to control the PLL1 of the High Speed PHY.

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

Bits 31:4 Reserved

Bits 3:1 **PLL1SEL[2:0]**: Controls the PHY PLL1 input clock frequency selection

- 000: 12 MHz
- 001: 12.5 MHz
- 010: 12.5 MHz
- 011: 16 MHz
- 100: 24 MHz
- 101: 25 MHz
- 110: 25 MHz
- 111: reserved

Bit 0 **PLL1EN**: Enable the PLL1 inside PHY

- 0: PLL1 disabled
- 1: PLL1 enabled

33.4.2 USBPHYC tuning control register (USBPHYC_TUNE)

Address offset: 0x00C

Reset value: 0x0000 0004

This register is used to control the tuning interface of the High Speed PHY.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SQLBYP	SHTCCTC TLPROT	HSRXOFF[1:0]	HSFALL PREEM	STAGS EL	HFRXG NEQEN	SQLCH CTL[1]	
									rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQLCH CTL[0]	HSDRVCHKZ TRM[1:0]	HSDRVCHKITRM[3:0]				HSDRV RFRED	FSDRV RFADJ	HSDRVC URINGR	HSDRV DCLEV	HSDRV DCCUR	HSDRVS LEW	LFSCA PEN	INCUR RINT	INCUR REN	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:4 Reserved

Bit 23 **SQLBYP**: This pin is used to bypass the squelch inter-locking circuitry

- 0: Do not bypass the squelch inter-locking
- 1: Bypass the squelch inter-locking

Bit 22 **SHTCCTCTLPROT**: Enables the short circuit protection circuitry in LS/FS driver

- 0: short circuit protection disabled
- 1: short circuit protection enabled

Bits 21:20 **HSRXOFF[1:0]**: HS Receiver Offset adjustment

- 00: off
- 01: + 5mV
- 10: +10 mV
- 11: -5 mV

Bit 19 **HSFALLPREEM**: HS Fall time control of single ended signals during pre-emphasis:

- 0: control on
- 1: control off

Bit 18 **STAGSEL**: HS Tx staggering enable:

- 0: Disable the basic staggering in the HS Tx Mode.
- 1: Enable the basic staggering in the HS Tx Mode.

Bit 17 **HDRXGNEQEN**: Enables the HS Rx Gain Equalizer:

- 0: Disable the Gain Equalizer
- 1: Enable the Gain Equalizer

Bits 16:15 **SQLCHCTL[1:0]**: Adjust the squelch DC threshold value

- 00: No shift in threshold.
- 01: Squelch DC threshold shift by -5 mv.
- 10: Squelch DC threshold shift by +7 mv.
- 11: Squelch DC threshold shift by +14 mv.

Bits 14:13 **HSDRVCHKZTRM[1:0]**: Controls the PHY bus HS driver impedance tuning for choke compensation

- 00: No impedance offset
- 01: Reduce the impedance by 2 ohms
- 10: Reduce the impedance by 4 ohms
- 11: Reduce the impedance by 6 ohms

Bits 12:9 **HSDRVCHKITRM[3:0]**: HS Driver current trimming pins for choke compensation.

- 0000: 18.87 mA target current / nominal + 0%
- 0001: 19.165 mA target current / nominal + 1.56%
- 0010: 19.46 mA target current / nominal + 3.12%
- 0011: 19.755 mA target current / nominal + 4.68%
- 0100: 20.05 mA target current / nominal + 6.24%
- 0101: 20.345 mA target current / nominal + 7.8%
- 0110: 20.64 mA target current / nominal + 9.36%
- 0111: 20.935 mA target current / nominal + 10.92%
- 1000: 21.23 mA target current / nominal + 12.48%
- 1001: 21.525 mA target current / nominal + 14.04%
- 1010: 21.82 mA target current / nominal + 15.6%
- 1011: 22.115 mA target current / nominal + 17.16%
- 1100: 22.458 mA target current / nominal + 19.01%
- 1101: 22.755 mA target current / nominal + 20.58%
- 1110: 23.052 mA target current / nominal + 22.16%
- 1111: 23.348 mA target current / nominal + 23.73%

Bit 8 **HSDRVRFRED**: High Speed rise-fall reduction enable.

- 0: Default rise/fall time.
- 1: Increases the rise/fall time by 20%.

Bit 7 **FSDRVRFADJ**: Tuning pin to adjust the full speed rise/fall time.

- 0: Disables the full speed rise/fall tuning option.
- 1: Enables the full speed rise/fall tuning option.

Bit 6 **HSDRVCURINCR**: Enable the HS driver current increase feature.

- 0: Disables the HSDRVDCLEV feature.
- 1: Enables the HSDRVDCLEV feature.

Bit 5 **HSDRVDCLEV**: Increases the HS Driver DC level. Not applicable during the HS Test J and Test K data transfer.

- 0: Increases the HS driver DC level by 5 to 7 mV if HSDRVCURINCR = '1'
- 1: Increases the HS driver DC level by 10 to 14 mV if HSDRVCURINCR = '1'

Bit 4 **HSDRVDCCUR**: Decreases the HS driver DC level.

- 0: Keeps the normal HS driver DC level.
- 1: Decreases the HS driver DC level by 5 to 7 mV.

Bit 3 **HSDRVSLEW**: Controls the HS driver slew rate.

0: Keeps the normal slew rate.

1: Slows the driver slew rate by 10%.

Bit 2 **LFSCAPEN**: Enables the Low Full Speed feedback capacitor.

0: Disables the feedback capacitor.

1: Enables the feedback capacitor.

Bit 1 **INCURRINT**: Controls PHY current boosting.

0: Provides a current boosting of 1mA if INCURREN = '1'.

1: Provides a current boosting of 2mA if INCURREN = '1'.

Bit 0 **INCURREN**: Controls the current boosting function.

0: Disables the current boosting.

1: Enables the current boosting.

33.4.3 USBPHYC LDO control and status register (USBPHYC_LDO)

Address offset: 0x018

Reset value: 0x0000 0001

This register is used to control the register (LDO) associated with the HS USB PHY

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

Bits 31:3 Reserved

Bit 2 **LDO_DISABLE**: Controls disable of the High Speed PHY's LDO.

0: LDO enabled

1: LDO disabled

Bit 1 **LDO_STATUS**: Monitors the status of the PHY's LDO.

0: LDO not ready

1: LDO ready

Bit 0 **LDO_USED**: Indicates the presence of the LDO in the chip.

0: LDO is not used

1: LDO is used

33.4.4 USBPHYC register map

Table 236. USBPHYC register map and reset values

Offset	Register	Reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0x00	USBPHYC_PLL1	Res.																																		
		Reset value																																		
.0x010-.0x008	Reserved																																			
	Reset value																																			
0x00C	USBPHYC_TUNE	Res.																																		
		Reset value																																		
.0x010-.0x014	Reserved																																			
	Reset value																																			
0x018	USBPHYC_LDO	Res.																																		
		Reset value																																		

Refer to [Section 1.5.2 on page 56](#) for the register boundary addresses.

34 Debug support (DBG)

34.1 Overview

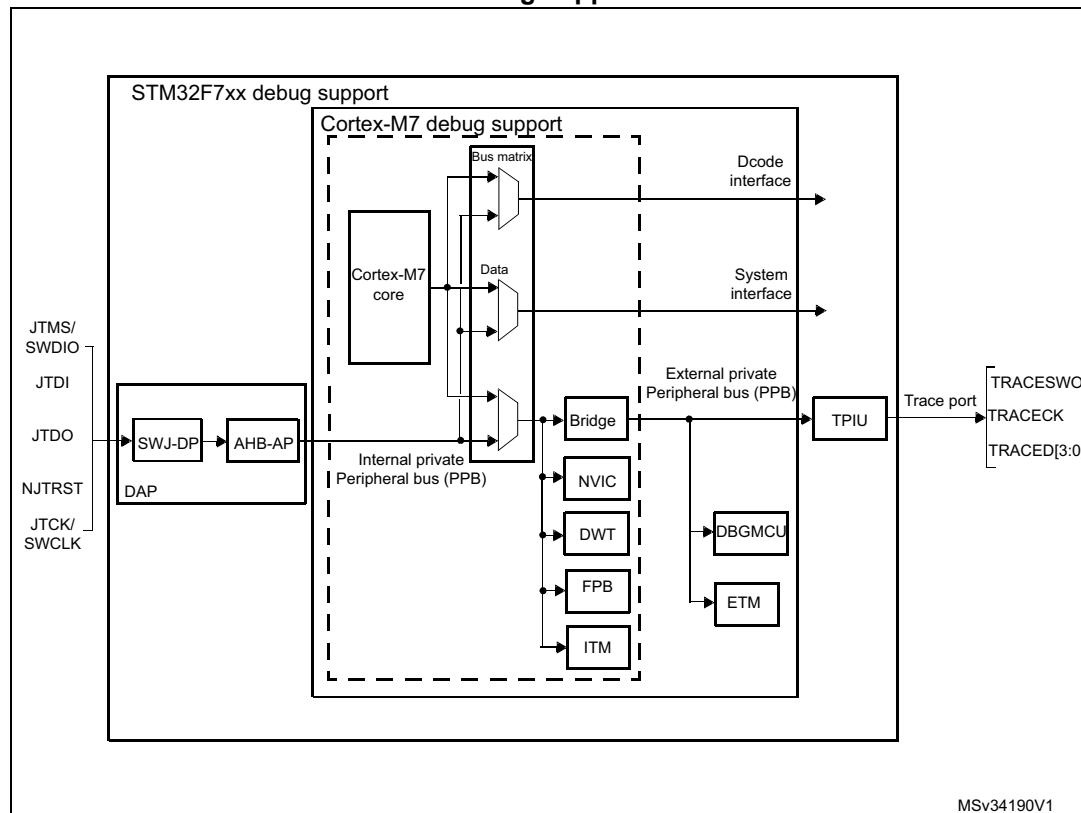
The STM32F72xxx and STM32F73xxx are built around a Cortex[®]-M7 with FPU core which contains hardware extensions for advanced debugging features. The debug extensions allow the core to be stopped either on a given instruction fetch (breakpoint) or data access (watchpoint). When stopped, the core's internal 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 STM32F72xxx and STM32F73xxx MCUs.

Two interfaces for debug are available:

- Serial wire
- JTAG debug port

Figure 460. Block diagram of STM32 MCU and Cortex[®]-M7 with FPU -level debug support



MSv34190V1

Note:

The debug features embedded in the Cortex[®]-M7 with FPU core are a subset of the ARM[®] CoreSight Components Technical Reference Manual.

The ARM® Cortex®-M7 with FPU 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
- TPIU: Trace port unit interface (available on larger packages, where the corresponding pins are mapped)
- ETM: Embedded Trace Macrocell (available on larger packages, where the corresponding pins are mapped)

It also includes debug features dedicated to the STM32F72xxx and STM32F73xxx:

- 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®-M7 with FPU core, refer to the Cortex®-M7 with FPU technical reference manual and to the CoreSight Components Technical Reference Manual (see [Section 34.2: Reference ARM® documentation](#)).

34.2 Reference ARM® documentation

- Cortex®-M7 with FPU technical reference manual (TRM)
(see Related documents on page 1)
- ARM® Debug Interface V5 architecture specification
- ARM® CoreSight Components Technical Reference Manual

34.3 SWJ debug port (serial wire and JTAG)

The core of the STM32F72xxx and STM32F73xxx 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 461. SWJ debug port

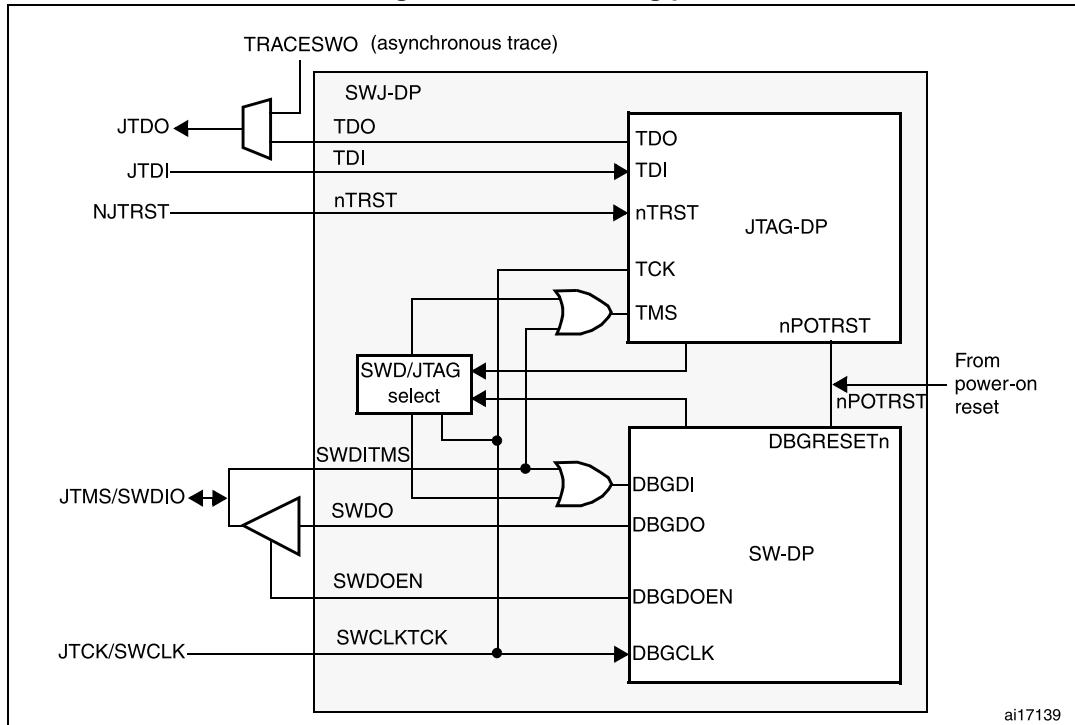


Figure 461 shows that the asynchronous TRACE output (TRACESWO) is multiplexed with TDO. This means that the asynchronous trace can only be used with SW-DP, not JTAG-DP.

34.3.1 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

34.4 Pinout and debug port pins

The STM32F72xxx and STM32F73xxx MCUs are available in various packages with different numbers of available pins. As a result, some functionality related to pin availability (TPIU parallel output interface) may differ between packages.

34.4.1 SWJ debug port pins

Five pins are used as outputs from the STM32F72xxx and STM32F73xxx for the SWJ-DP as *alternate functions* of general-purpose I/Os. These pins are available on all packages.

Table 237. 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	IO	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
NJTRST	I	JTAG Test nReset	-	-	PB4

34.4.2 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 STM32F72xxx and STM32F73xxx MCUs offer the possibility of disabling some or all of the SWJ-DP ports and so, of releasing (in gray in the table below) the associated pins for general-purpose IO (GPIO) usage. For more details on how to disable SWJ-DP port pins, refer to [Section 6.3.2: I/O pin alternate function multiplexer and mapping](#).

Table 238. Flexible SWJ-DP pin assignment

Available debug ports	SWJ IO pin assigned				
	PA13 / JTMS / SWDIO	PA14 / JTCK / SWCLK	PA15 / JTDI	PB3 / JTDO	PB4 / NJTRST
Full SWJ (JTAG-DP + SW-DP) - Reset State	X	X	X	X	X
Full SWJ (JTAG-DP + SW-DP) but without NJTRST	X	X	X	X	
JTAG-DP Disabled and SW-DP Enabled	X	X			
JTAG-DP Disabled and SW-DP Disabled					Released

Note:

When the APB bridge write buffer is full, it takes one extra APB cycle when writing the GPIO_AFR 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 TRST, TDI and TMS, to 0 for TCK)
- Cycle 2: the GPIO controller takes the control signals of the SWJTAG IO pins (like controls of direction, pull-up/down, Schmitt trigger activation, etc.).

34.4.3 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 IO levels, the device embeds internal pull-ups and pull-downs on the JTAG input pins:

- NJTRST: Internal pull-up
- JTDI: Internal pull-up
- JTMS/SWDIO: Internal pull-up
- TCK/SWCLK: Internal pull-down

Once a JTAG IO 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:

- NJTRST: AF input pull-up
- JTDI: AF input pull-up
- JTMS/SWDIO: AF input pull-up
- JTCK/SWCLK: AF input pull-down
- JTDO: AF output 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 JTCK, the device needs an integrated pull-down.

Having embedded pull-ups and pull-downs removes the need to add external resistors.

34.4.4 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 change the GPIO (PA15, PB3 and PB4) configuration mode in the GPIO_MODER register. 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 IO pin configuration in the IOPORT controller has no effect.

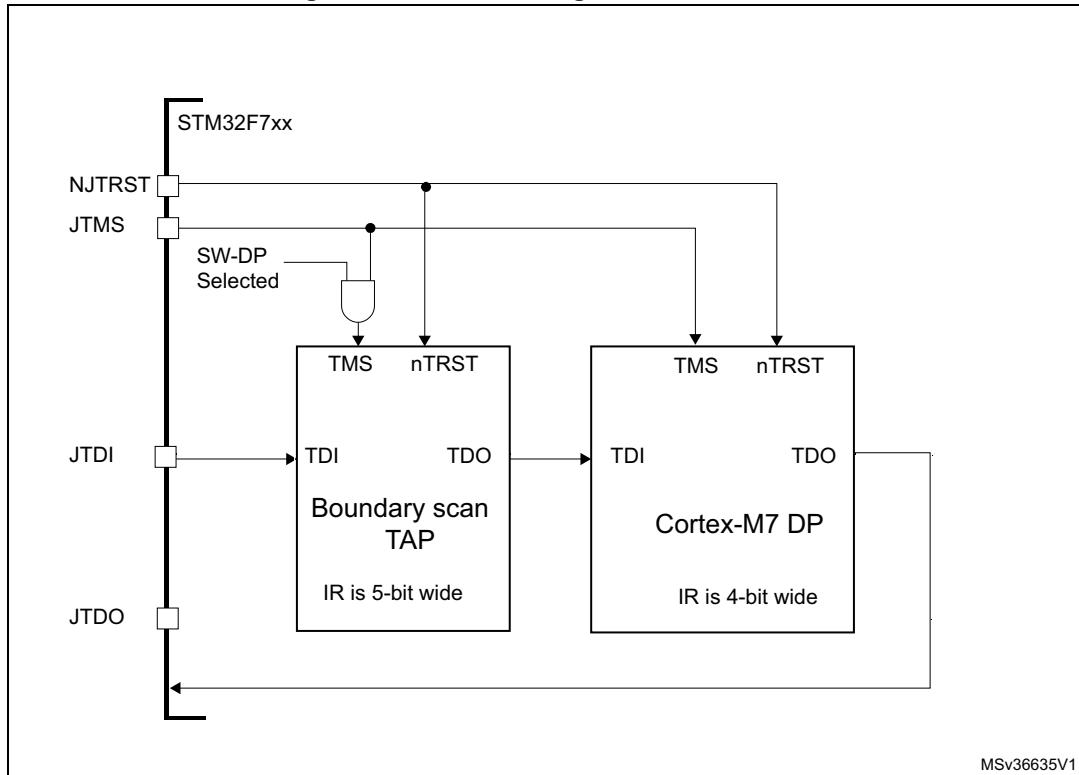
34.5 STM32F72xxx and STM32F73xxx JTAG Debug Port connection

The STM32F72xxx and STM32F73xxx MCUs integrate two serially connected JTAG Debug Ports, the boundary scan Debug Port (IR is 5-bit wide) and the Cortex®-M7 with FPU Debug Port (IR is 4-bit wide).

To access the Debug Port of the Cortex®-M7 with FPU for debug purposes:

1. First, it is necessary to shift the BYPASS instruction of the boundary scan Debug Port.
2. Then, for each IR shift, the scan chain contains 9 bits (=5+4) and the unused Debug Port instruction must be shifted in using the BYPASS instruction.
3. For each data shift, the unused Debug Port, 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 boundary scan Debug Port is automatically disabled (JTMS forced high).

Figure 462. JTAG Debug Port connections

34.6 ID codes and locking mechanism

There are several ID codes inside the STM32F72xxx and STM32F73xxx MCUs. 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.

34.6.1 MCU device ID code

The STM32F72xxx and STM32F73xxx MCUs integrate 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 34.16 on page 1385](#)). 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.

Only the DEV_ID(11:0) should be used for identification by the debugger/programmer tools.

DBGMCU_IDCODE

Address: 0xE004 2000

Only 32-bits access supported. Read-only.

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

Bits 31:16 **REV_ID[15:0]**: Revision identifier

This field indicates the revision of the device:
0x1000 = Revision A and 1

Bits 15:12 Reserved, must be kept at reset value.

Bits 11:0 **DEV_ID[11:0]**: Device identifier
The device ID is 0x452.

34.6.2 Boundary scan Debug Port

JTAG ID code

The Debug Port of the STM32F72xxx and STM32F73xxx BSC (boundary scan) integrates a JTAG ID code equal to 0x06449041.

34.6.3 Cortex®-M7 with FPU Debug Port

The Debug Port of the ARM® Cortex®-M7 with FPU 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 0x5BA00477 (corresponds to Cortex®-M7 with FPU, see [Section 34.2: Reference ARM® documentation](#)).

34.6.4 Cortex®-M7 with FPU JEDEC-106 ID code

The ARM® Cortex®-M7 with FPU integrates a JEDEC-106 ID code. It is located in the 4KB ROM table mapped on the internal PPB bus at address 0xE00FF000_0xE00FFFFF.

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.

34.7 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®-M7 with FPU technical *reference manual (TRM)*, for references, see [Section 34.2: Reference ARM® documentation](#)).

Table 239. JTAG debug port data registers

IR(3:0)	Data register	Details
1111	BYPASS [1 bit]	-
1110	IDCODE [32 bits]	ID CODE 0x06449041 (ARM® Cortex®-M7 with FPU ID Code)
1010	DPACC [35 bits]	Debug port access register This initiates a debug port and allows access to a debug port register. – When transferring data IN: 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: 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: 010 = OK/FAULT 001 = WAIT OTHER = reserved Refer to Table 240 for a description of the A(3:2) bits

Table 239. JTAG debug port data registers (continued)

IR(3:0)	Data register	Details
1011	APACC [35 bits]	<p>Access port access register</p> <p>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>Abort register</p> <ul style="list-style-type: none"> – Bits 31:1 = Reserved – Bit 0 = DAPABORT: write 1 to generate a DAP abort.

Table 240. 32-bit debug port registers addressed through the shifted value A[3:2]

Address	A(3:2) value	Description
0x0	00	Reserved, must be kept at reset value.
0x4	01	DP CTRL/STAT register. Used to: – Request a system or debug power-up – Configure the transfer operation for AP accesses – Control the pushed compare and pushed verify operations. – Read some status flags (overrun, power-up acknowledges)
0x8	10	DP SELECT register: Used to select the current access port and the active 4-words register window. – Bits 31:24: APSEL: select the current AP – Bits 23:8: reserved – Bits 7:4: APBANKSEL: select the active 4-words register window on the current AP – Bits 3:0: reserved
0xC	11	DP RDBUFF register: Used to allow the debugger to get the final result after a sequence of operations (without requesting new JTAG-DP operation)

34.8 SW debug port

34.8.1 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.

34.8.2 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 241. Packet request (8-bits)

Bit	Name	Description
0	Start	Must be “1”
1	APnDP	0: DP Access 1: AP Access
2	RnW	0: Write Request 1: Read Request
4:3	A(3:2)	Address field of the DP or AP registers (refer to Table 240)
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®-M7 with FPU *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 242. 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 243. 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.

34.8.3 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 to **0x5BA02477** (corresponding to Cortex®-M7 with FPU).

- 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®-M7 with FPU TRM* and the *CoreSight Components Technical Reference Manual*.

34.8.4 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.

34.8.5 SW-DP registers

Access to these registers are initiated when APnDP=0

Table 244. 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. 0x5BA02477 (identifies the SW-DP)
00	Write	-	ABORT	-

Table 244. SW-DP registers (continued)

A(3:2)	R/W	CTRLSEL bit of SELECT register	Register	Notes
01	Read/Write	0	DP-CTRL/STAT	Purpose is to: – request a system or debug power-up – configure the transfer operation for AP accesses – control the pushed compare and pushed verify operations. – read some status flags (overrun, power-up acknowledges)
01	Read/Write	1	WIRE CONTROL	Purpose is to configure the physical serial port protocol (like the duration of the turnaround time)
10	Read	-	READ RESEND	Enables recovery of the read data from a corrupted debugger transfer, without repeating the original AP transfer.
10	Write	-	SELECT	The purpose is to select the current access port and the active 4-words register window
11	Read/Write	-	READ BUFFER	This read buffer is useful because AP accesses are posted (the result of a read AP request is available on the next AP transaction). This read buffer captures data from the AP, presented as the result of a previous read, without initiating a new transaction

34.8.6 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

34.9 AHB-AP (AHB access port) - valid for both JTAG-DP and 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:

- c) Bits [7:4] = the bits [7:4] APBANKSEL of the DP SELECT register
- d) 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®-M7 with FPU includes 9 x 32-bits registers:

Table 245. Cortex®-M7 with FPU 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	-
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®-M7 with FPU TRM* for further details.

34.10 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 246. Core debug registers

Register	Description
DHCSR	The 32-bit Debug Halting Control and Status Register This provides status information about the state of the processor enable core debug halt and step the processor
DCRSR	The 17-bit Debug Core Register Selector Register: This selects the processor register to transfer data to or from.
DCRDR	The 32-bit Debug Core Register Data Register: This holds data for reading and writing registers to and from the processor selected by the DCRSR (Selector) register.
DEMCR	The 32-bit Debug Exception and Monitor Control Register: This provides Vector Catching and Debug Monitor Control. 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®-M7 with FPU 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.

34.11 Capability of the debugger host to connect under system reset

The reset system of the STM32F72xxx and STM32F73xxx MCU 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®-M7 with FPU 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.*

34.12 FPB (Flash patch breakpoint)

Typically in Cortex-M architecture the FPB unit allows to:

- implement hardware breakpoints
- patch code and data from code space to system space. This feature gives the possibility to correct software bugs located in the Code Memory Space.

Where the use of a Software Patch or a Hardware Breakpoint is exclusive.

But there are some major changes in Pelican FPB:

- Flash patching is no more supported (there is no FP_REMAP register)
- All comparators are for instruction addresses (up to 8 instruction breakpoints)
- Programmer's model for breakpoint comparators is enhanced to allow hardware breakpoint in full address range.

34.13 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

34.14 ITM (instrumentation trace macrocell)

34.14.1 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®-M7 with FPU 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 the user programs or uses the ITM.

34.14.2 Time stamp packets, synchronization and overflow packets

Time stamp packets encode time stamp 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 247. 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 = DWTEA: 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)
		Bit 0 = ITMENA: Global Enable Bit of the ITM
@E0000E40	ITM trace privilege	Bit 3: mask to enable tracing ports31:24
		Bit 2: mask to enable tracing ports23:16
		Bit 1: mask to enable tracing ports15:8
		Bit 0: mask to enable tracing ports7:0
@E0000E00	ITM trace enable	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 34.17.2: TRACE pin assignment](#) and [Section 34.16.3: 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)

34.15 ETM (Embedded trace macrocell)

34.15.1 General description

The ETM enables the reconstruction of program execution. Data are traced using the Data Watchpoint and Trace (DWT) component or the Instruction Trace Macrocell (ITM) whereas instructions are traced using the Embedded Trace Macrocell (ETM).

The ETM transmits information as packets and is triggered by embedded resources. These resources must be programmed independently and the trigger source is selected using the Trigger Event Register (0xE0041008). An event could be a simple event (address match from an address comparator) or a logic equation between 2 events. The trigger source is one of the fourth comparators of the DWT module. The following events can be monitored:

- Clock cycle matching
- Data address matching

For more informations on the trigger resources refer to [Section 34.13: DWT \(data watchpoint trigger\)](#).

The packets transmitted by the ETM are output to the TPIU (Trace Port Interface Unit). The formatter of the TPIU adds some extra packets (refer to [Section 34.17: Pelican TPIU \(trace port interface unit\)](#)) and then outputs the complete packet sequence to the debugger host.

Note: N.B: Cortex-M7 ETM is compliant with ARM ETM architecture v4, which programming model is not backward compatible with Cortex-M4 ETM one (ETM architecture v3.5).

34.15.2 Signal protocol, packet types

This part is described in the chapter 6 ETM v4 architecture specification (IHI0064B).

34.15.3 Main ETM registers

For more information on registers refer to the Pelican ETM technical reference manual (DDI0494-2a) and the ETM v4 architecture specification (IHI0064B).

34.15.4 Configuration example

To output a simple value to the TPIU:

- Configure trace I/Os: enable TRACE_CLKINEN in the STM32F72xxx and STM32F73xxx debug configuration register (DBGMCU_CR).
- Write @ E000EDFC 01000000; SCS: set TRCENA, otherwise trace registers are not accessible.
- Write @ E00400F0 00000000; TPIU: select SYNC PORT Mode
- Write @ E0040004 00000008; TPIU: select TPIU PORT SIZE=4
- Write @ E0001020 002002CA; WT: PC MATCH Comparator (PC=0x2002CA)
- Write @ E0001024 00000000; DWT: No mask apply on comparator
- Write @ E0001028 00000008; DWT: ETM trig on PC on match

ETM:

- Write @ E0041004 00000000; Disable ETM
- Read @ E004100C 00000003; ETM should be in Idle state
- Write @ E0041040 00000002; Instruction trace source ID = 0x2
- Write @ E0041080 00000001; Resource for ViewInst enabling event is “always TRUE”
- Write @ E004108C 000000FF; Processor comparator selection for Start: pc_match0 (=>DWT match)
- Write @ E0041004 00000001; Enable ETM

34.16 MCU debug component (DBGMCU)

The MCU debug component helps the debugger provide support for:

- Low-power modes
- Clock control for timers, watchdog, I2C and bxCAN during a breakpoint
- Control of the trace pins assignment

34.16.1 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.

34.16.2 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.

34.16.3 Debug MCU configuration register

This register allows the configuration of the MCU under DEBUG. This concerns:

- Low-power mode support
- Timer and watchdog counter support
- bxCAN communication support
- Trace pin assignment

This **DBGMCU_CR** is mapped on the External PPB bus at address 0xE0042004

It is asynchronously reset by the PORESET (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.

34.16.4 **DBGMCU_CR** register

Address: 0xE004 2004

Only 32-bit access supported

POR Reset: 0x0000 0000 (not reset by system reset)

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

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

Bits 7:5 **TRACE_MODE[1:0] and TRACE_CLKINEN**: Trace clock and pin assignment control

- With TRACE_CLKINEN=0:
TRACE_MODE=xx: TRACE output disabled (both synchronous and asynchronous)
- With TRACE_CLKINEN=1:
 - TRACE_MODE[1:0]=00: Aynchronous trace interface enabled at pad level (TRACESWO available only on TDO pad when using Serial Wire mode) / Synchronous trace interface enabled
 - TRACE_MODE[1:0] different to =00: Asynchronous trace interface disabled / Synchronous trace interface enabled

Bits 4:3 Reserved, must be kept at reset value.

Bit 2 **DBG_STANDBY**: Debug Standby mode

0: (FCLK=Off, HCLK=Off) The whole digital part is unpowered.

From software point of view, exiting from Standby is identical than fetching reset vector (except a few status bit indicated that the MCU is resuming from Standby)

1: (FCLK=On, HCLK=On) In this case, the digital part is not unpowered and FCLK and HCLK are provided by the internal RC oscillator which remains active. In addition, the MCU generate a system reset during Standby mode so that exiting from Standby is identical than fetching from reset

Bit 1 **DBG_STOP**: Debug Stop mode

0: (FCLK=Off, HCLK=Off) In STOP mode, the clock controller disables all clocks (including HCLK and FCLK). When exiting from STOP mode, the clock configuration is identical to the one after RESET (CPU clocked by the 8 MHz internal RC oscillator (HSI)). Consequently, the software must reprogram the clock controller to enable the PLL, the Xtal, etc.

1: (FCLK=On, HCLK=On) In this case, when entering STOP mode, FCLK and HCLK are provided by the internal RC oscillator which remains active in STOP mode. When exiting STOP mode, the software must reprogram the clock controller to enable the PLL, the Xtal, etc. (in the same way it would do in case of DBG_STOP=0)

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).

34.16.5 Debug MCU APB1 freeze register (DBGMCU_APB1_FZ)

The DBGMCU_APB1_FZ register is used to configure the MCU under DEBUG. It concerns APB2 peripherals. It is mapped on the external PPB bus at address 0xE004 2008

It is asynchronously reset by the POR (and not the system reset). It can be written by the debugger under system reset.

Address: 0xE004 2008

Only 32-bit access is supported.

Power-on-reset (POR): 0x0000 0000 (not reset by system reset)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	DBG_CAN1_STOP	Res.	DBG_I2C3_SMBUS_TIMEOUT	DBG_I2C2_SMBUS_TIMEOUT	DBG_I2C1_SMBUS_TIMEOUT	Res.	Res.	Res.	Res.	Res.
						rw		rw	rw						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	DBG_IWDG_STOP	DBG_WWDG_STOP	DBG_RTC_STOP	DBG_LPTIM1_STOP	DBG_TIM14_STOP	DBG_TIM13_STOP	DBG_TIM12_STOP	DBG_TIM7_STOP	DBG_TIM6_STOP	DBG_TIM5_STOP	DBG_TIM4_STOP	DBG_TIM3_STOP	DBG_TIM2_STOP
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bit 25 **DBG_CAN1_STOP:** Debug CAN1 stopped when Core is halted

- 0: Same behavior as in normal mode
- 1: The CAN1 receive registers are frozen

Bit 24 Reserved, must be kept at reset value.

Bit 23 **DBG_I2C3_SMBUS_TIMEOUT:** SMBUS timeout mode stopped when Core is halted

- 0: Same behavior as in normal mode
- 1: The SMBUS timeout is frozen

Bit 22 **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 21 **DBG_I2C1_SMBUS_TIMEOUT:** SMBUS timeout mode stopped when Core is halted

- 0: Same behavior as in normal mode
- 1: The SMBUS timeout is frozen

Bits 20:13 Reserved, must be kept at reset value.

Bit 12 **DBG_IWDG_STOP:** Debug independent watchdog stopped when core is halted

- 0: The independent watchdog counter clock continues even if the core is halted
- 1: The independent watchdog counter clock is stopped when the core is halted

Bit 11 **DBG_WWDG_STOP:** Debug Window Watchdog stopped when Core is halted

- 0: The window watchdog counter clock continues even if the core is halted
- 1: The window watchdog counter clock is stopped when the core is halted

- Bit 10 **DBG_RTC_STOP:** RTC stopped when Core is halted
 0: The RTC counter clock continues even if the core is halted
 1: The RTC counter clock is stopped when the core is halted
- Bit 9 **DBG_LPTIM1_STOP:** LPTIM1 counter stopped when core is halted
 0: The clock of LPTIM1 counter is fed even if the core is halted
 1: The clock of LPTIM1 counter is stopped when the core is halted
- Bits 8:0 **DBG_TIMx_STOP:** TIMx counter stopped when core is halted ($x=2..7, 12..14$)
 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

34.16.6 Debug MCU APB2 Freeze register (DBGMCU_APB2_FZ)

The DBGMCU_APB2_FZ register is used to configure the MCU under Debug. It concerns APB2 peripherals.

This register is mapped on the external PPB bus at address 0xE004 200C

It is asynchronously reset by the POR (and not the system reset). It can be written by the debugger under system reset.

Address: 0xE004 200C

Only 32-bit access is supported.

POR: 0x0000 0000 (not reset by system reset)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	DBG_TIM11_STOP	DBG_TIM10_STOP	DBG_TIM9_STOP												
													rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	DBG_TIM8_STOP	DBG_TIM1_STOP													
													rw	rw	

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

Bits 18:16 **DBG_TIMx_STOP:** TIMx counter stopped when core is halted ($x=9..11$)

- 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

Bits 15:2 Reserved, must be kept at reset value.

Bit 1 **DBG_TIM8_STOP:** TIM8 counter stopped when core is halted

- 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 0 **DBG_TIM1_STOP:** TIM1 counter stopped when core is halted

- 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

34.17 Pelican TPIU (trace port interface unit)

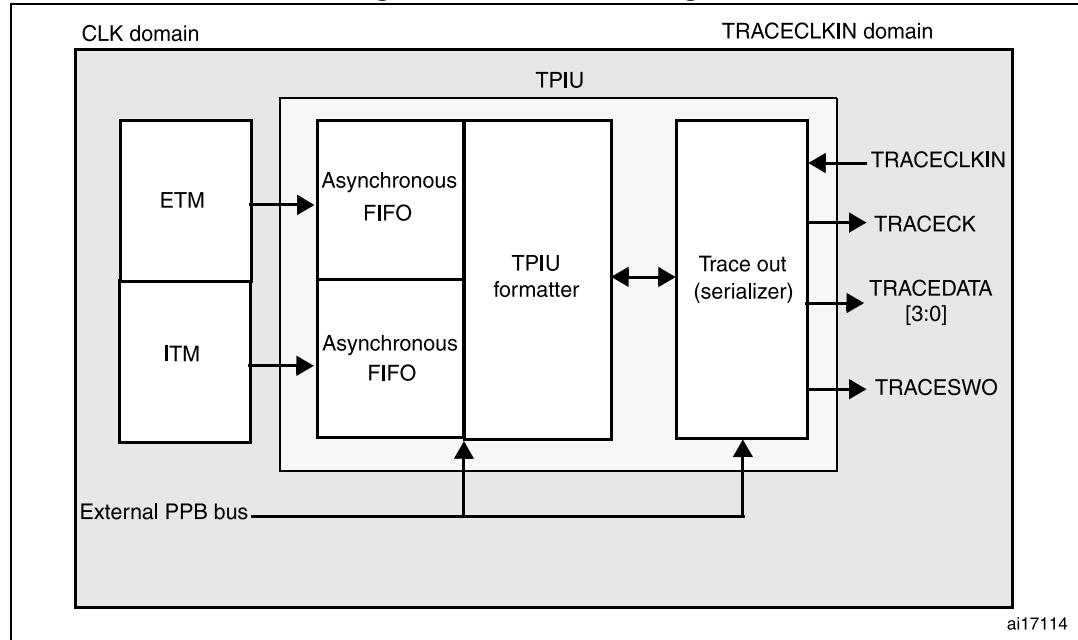
34.17.1 Introduction

The TPIU acts as a bridge between the on-chip trace data from the ITM, the ETM and the external trace capture device.

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).

Figure 463. TPIU block diagram



34.17.2 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 248. Asynchronous TRACE pin assignment

TPIU pin name	Trace synchronous mode	
	Type	Description
TRACESWO	O	TRACE Async Data Output

- 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 249. Synchronous TRACE pin assignment

TPIU pin name	Trace synchronous mode	
	Type	Description
TRACECK	O	TRACE Clock
TRACED[3:0]	O	TRACE Sync Data Outputs Can be 1, 2 or 4.

TPIU TRACE pin assignment

By default, these pins are NOT assigned. They can be assigned by setting the TRACE_CLKINEN and TRACE_MODE bits in 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_CLKINEN 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 250. Flexible TRACE pin assignment

DBGMCU_CR register		Pins assigned for:	TRACE IO pin assigned					
			JTDO/ TRACESWO	TRACECK	TRACED[0]	TRACED[1]	TRACED[2]	TRACED[3]
0	XX	No Trace (default state)	Released ⁽¹⁾			-		
1	00	Asynchronous Trace	TRACESWO	-	-	Released (usable as GPIO)		
1	different of 00	Synchronous Trace 1 bit ⁽²⁾	Released ⁽¹⁾	TRACECK	TRACED[0]	-	-	-
1	different of 00	Synchronous Trace 2 bit ⁽²⁾		TRACECK	TRACED[0]	TRACED[1]	-	-
1	different of 00	Synchronous Trace 4 bit ⁽²⁾		TRACECK	TRACED[0]	TRACED[1]	TRACED[2]	TRACED[3]

1. When Serial Wire mode is used, it is released. But when JTAG is used, it is assigned to JTDO.

2. Selected with Bit[3:0] Current port size from TPIU register.

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_CLKINEN 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

34.17.3 TPIU formatter

The purpose of this formatter is to build 128 bit frames containing trace data from, potentially, both the ETM and the ITM, and to allow at a trace analyzer level a correlation between trace packets and emitters.

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

34.17.4 TPIU frame synchronization packets

The TPIU 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.

34.17.5 Transmission of the synchronization frame packet

There is no Synchronization Counter register implemented in the TPIU 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 TPIU Frame synchronization packet (0x7F_FF_FF_FF) is emitted:

- after each TPIU reset release. This reset is synchronously released with the rising edge of the TRACECLKIN clock. This means that this packet is transmitted when the

TRACE_CLKINEN bit in the DBGMCU_CFG register is 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 TPIU (trace source ID added).

34.17.6 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 TRACELKIN (equal to HCLK). Consequently, the output frequency of TRACECK is equal to HCLK/2.

34.17.7 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.

Single IO trace mode is typically suitable for ITM trace output. Also, formatter is disabled in case of asynchronous trace, so merging of ETM and ITM trace streams is not possible.

TRACESWO is multiplexed with JTDO when using the SW-DP pin. This way, this functionality is available in all STM32F72xxx and STM32F73xxx packages.

This asynchronous mode requires a constant frequency for TRACELKIN. For the standard UART (NRZ) capture mechanism, 5% accuracy is needed. The Manchester encoded version is tolerant up to 10%.

34.17.8 TRACECLKIN connection inside the STM32F72xxx and STM32F73xxx

In the STM32F72xxx and STM32F73xxx, this TRACELKIN 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 STM32F72xxx and STM32F73xxx MCUs 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 TRACE_CLKINEN bit) 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.

34.17.9 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 251. Important TPIU registers

Address	Register	Description
0xE0040004	Current port size	Allows the trace port size to be selected: Bit 0: Port size = 1 Bit 1: Port size = 2 Bit 2: Port size = 3, not supported Bit 3: Port Size = 4 Only 1 bit must be set. By default, the port size is one bit. (0x00000001)
0xE00400F0	Selected pin protocol	Allows the Trace Port Protocol to be selected: 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	Bits 31-9 = always '0 Bit 8 = TrigIn = always '1 to indicate that triggers are indicated Bits 7-4 = always 0 Bits 3-2 = always 0 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. Bit 0 = always 0 The resulting default value is 0x102 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).
0xE0040300	Formatter and flush status	Not used in Cortex®-M7 with FPU, always read as 0x00000008

34.17.10 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

34.18 DBG register map

The following table summarizes the Debug registers.

Table 252. 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		
0xE004 2000	DBGMCU_IDCODE	Res.																																	
0xE004 2004	DBGMCU_CR	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		
0xE004 2008	DBGMCU_APB1_FZ	Res.																																	
0xE004 200C	DBGMCU_APB2_FZ	Res.																																	
	Reset value ⁽¹⁾																																		
	Reset value																																		
	Reset value																																		
	Reset value																																		

1. The reset value is product dependent. For more information, refer to [Section 34.6.1: MCU device ID code](#).

35 Device electronic signature

The electronic signature is stored in the Flash memory area. It can be read using the JTAG/SWD or the CPU. It contains factory-programmed identification data that allow the user firmware or other external devices to automatically match its interface to the characteristics of the STM32F72xxx and STM32F73xxx microcontrollers.

35.1 Unique device ID register (96 bits)

The unique device identifier is ideally suited:

- for use as serial numbers (for example USB string serial numbers or other end applications)
- for use as security keys in order to increase the security of code in Flash memory while using and combining this unique ID with software cryptographic primitives and protocols before programming the internal Flash memory
- to activate secure boot processes, etc.

The 96-bit unique device identifier provides a reference number which is unique for any device and in any context. These bits can never be altered by the user.

The 96-bit unique device identifier can also be read in single bytes/half-words/words in different ways and then be concatenated using a custom algorithm.

Base address: 0x1FF0 7A10

Address offset: 0x00

Read only = 0xXXXX XXXX where X is factory-programmed

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

Bits 31:0 **UID[31:0]**: X and Y coordinates on the wafer

Address offset: 0x04

Read only = 0xXXXX XXXX where X is factory-programmed

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UID[63:48]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
UID[47:32]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:8 **UID[63:40]**: LOT_NUM[23:0]

Lot number (ASCII encoded).

Bots 7:0 **UID[39:32]**: WAF_NUM[7:0]

Wafer number (8-bit unsigned number).

Address offset: 0x08

Read only = 0xXXXX XXXX where X is factory-programmed

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UID[95:80]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
UID[79:64]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **UID[95:64]**: LOT_NUM[55:24]

Lot number (ASCII encoded).

35.2 Flash size

Base address: 0x1FF0 7A22

Address offset: 0x00

Read only = 0xXXXX where X is factory-programmed

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F_SIZE															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:0 F_ID(15:0): Flash memory size

This bitfield indicates the size of the device Flash memory expressed in Kbytes.

As an example, 0x0400 corresponds to 1024 Kbytes.

35.3 Package data register

Base address: 0x1FF0 7BF0

Address offset: 0x00

Read only = 0xXXXX where X is factory-programmed

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

Bits 15:11 Reserved, must be kept at reset value.

Bits 10:8 PKG[2:0]: Package type

0x111: Reserved

0x110: LQFP176 and UFBGA176 packages with USB OTG PHY HS

0x101: LQFP144 and UFBGA144 packages with USB OTG PHY HS

0x100: WLCSP100 package with USB OTG PHY HS

0x011: LQFP176 and UFBGA176 packages without USB OTG PHY HS

0x010: LQFP144 package without USB OTG PHY HS

0x001: LQFP100 package without USB OTG PHY HS

0x000: LQFP64 package without USB OTG PHY HS

Bits 7:0 Reserved, must be kept at reset value.

36 Revision history

Table 253. Document revision history

Date	Revision	Changes
25-Jan-2017	1	<p>Initial release.</p> <ul style="list-style-type: none"> – Updated Arm word and added logo. <p>USB section:</p> <ul style="list-style-type: none"> – Complete re-mastering of the section – Added Section 32.4.2: USB OTG pin and internal signals. – Updated Section 32.15.13: OTG core ID register (OTG_CID). – Updated Section 32.15.38: OTG all endpoints interrupt mask register (OTG_DAINTMSK) replacing bit 18 by bit 19 in OEPMSK bit description. <p>Memory organization section</p> <ul style="list-style-type: none"> – Added Figure 1: Memory map. <p>System and memory overview section</p> <ul style="list-style-type: none"> – Updated Figure 2: System architecture for STM32F72xxx and STM32F73xxx devices. <p>Flash memory section:</p> <ul style="list-style-type: none"> – Updated Section 3.7.7: Flash option control register (FLASH_OPTCR1) and Section 3.7.8: Flash interface register map reset value at '0x0040 0080'. <p>DMA section</p> <ul style="list-style-type: none"> – Updated Section 8.2: DMA main features 'up to 16channels per stream'. – Updated Section 8.5.5: DMA stream x configuration register (DMA_SxCR) and Section 8.5.11: DMA register map CHSEL mapped from 2 to 3 bits. <p>RCC section</p> <ul style="list-style-type: none"> Updated Figure 14: Clock tree OTG_HS_SCL renamed by OTG_HS_ULPI_CK. Update Section 5.3.21: RCC clock control & status register (RCC_CSR) bit RMVF put in read/write. Updated Section 5.3.27: RCC register map PADRSTF in PINRSTF. Updated Section 5.3.20: RCC backup domain control register (RCC_BDCR) adding LSEDRV[1:0] in the description. Updated Section 5.3.14: RCC APB2 peripheral clock enable register (RCC_APB2ENR) ADC1EN, ADC2EN and ADC3EN are enabled when bit set to '1'. <p>PWR section:</p> <ul style="list-style-type: none"> Updated Section 4.1.3: Battery backup domain note removing 'only one I/O at a time can be used as an output' sentence. Updated Section 4.1.3: Battery backup domain step 3 of 'Access to the backup SRAM' paragraph. Updated Section 4.4.2: PWR power control/status register (PWR_CSR1) bits[19:18] UDRDY[1:0] description.
09-Mar-2018	2	<ul style="list-style-type: none"> – Updated Section 8.2: DMA main features 'up to 16channels per stream'. – Updated Section 8.5.5: DMA stream x configuration register (DMA_SxCR) and Section 8.5.11: DMA register map CHSEL mapped from 2 to 3 bits. <p>RCC section</p> <ul style="list-style-type: none"> Updated Figure 14: Clock tree OTG_HS_SCL renamed by OTG_HS_ULPI_CK. Update Section 5.3.21: RCC clock control & status register (RCC_CSR) bit RMVF put in read/write. Updated Section 5.3.27: RCC register map PADRSTF in PINRSTF. Updated Section 5.3.20: RCC backup domain control register (RCC_BDCR) adding LSEDRV[1:0] in the description. Updated Section 5.3.14: RCC APB2 peripheral clock enable register (RCC_APB2ENR) ADC1EN, ADC2EN and ADC3EN are enabled when bit set to '1'. <p>PWR section:</p> <ul style="list-style-type: none"> Updated Section 4.1.3: Battery backup domain note removing 'only one I/O at a time can be used as an output' sentence. Updated Section 4.1.3: Battery backup domain step 3 of 'Access to the backup SRAM' paragraph. Updated Section 4.4.2: PWR power control/status register (PWR_CSR1) bits[19:18] UDRDY[1:0] description.

Table 253. Document revision history (continued)

Date	Revision	Changes
09-Mar-2018	2 (continued)	<p>FMC section: Updated Section : SDRAM Control registers 1,2 (FMC_SDCR1,2) replacing 'KCK_FMC' by 'HCLK' in RPIPE[1:0] description.</p> <p>RTC section Updated Section 25.6.3: RTC control register (RTC_CR) WUTE bit description adding note. Updated Figure 286: RTC block diagram WUCKSEL dividers for /2,4,8,16.</p> <p>Serial audio section: – Updated Section 29.5.18: SAI register map MCJDIV by MCKDIV bit name.</p> <p>SDMMC section: Updated Section 30.8.8: SDMMC data length register (SDMMC_DLEN) note.</p> <p>General-purpose timer section: – Updated Section 19.4.3: TIMx slave mode control register (TIMx_SMCR)(x = 2 to 5) Bits 6:4 TS: Trigger selection description removing 'reserved'. – Added Section 20.3.12: Retriggerable one pulse mode (OPM) (TIM12 only). – Added Section 20.3.13: UIF bit remapping. – Added Section 20.3.13: UIF bit remapping.</p> <p>USART section: – Updated Section 27.5.4: USART baud rate generation note, replacing '0d16' by '16d'.</p>

Table 253. Document revision history (continued)

Date	Revision	Changes
26-Jun-2018	3	<p>Updated documentation convention section:</p> <ul style="list-style-type: none"> – Added Section 1.1: General information. <p>Updated system architecture section:</p> <ul style="list-style-type: none"> – Updated Figure 2: System architecture for STM32F72xxx and STM32F73xxx devices. <p>Updated memory organization section:</p> <ul style="list-style-type: none"> – Updated Figure 1: Memory map. <p>Updated embedded Flash memory section:</p> <ul style="list-style-type: none"> – Updated Section 3.3.1: Flash memory organization and added Table 4: STM32F730xx Flash memory organization. <p>Updated PWR section:</p> <ul style="list-style-type: none"> – Updated title of Figure 5: STM32F7x2xx and STM32F730xx power supply overview and Figure 6: STM32F7x3xx and STM32F730xx power supply overview*. – Added note 3 below Figure 6: STM32F7x3xx and STM32F730xx power supply overview about packages supporting the integrated OTG_HS PHY. <p>Updated RCC section:</p> <ul style="list-style-type: none"> – Updated note1 and note2 below Figure 14: Clock tree. – Updated note1 adding STM32F730xx devices in Section 5.3.9: RCC APB2 peripheral reset register (RCC_APB2RSTR) and Section 5.3.14: RCC APB2 peripheral clock enable register (RCC_APB2ENR). <p>Updated ADC section:</p> <ul style="list-style-type: none"> – Updated Section 14.10: Temperature sensor. <p>Updated SAI section:</p> <ul style="list-style-type: none"> – Updated title of Section 29.5.10: Interrupt mask register (SAI_AIM) and Section 29.5.11: Interrupt mask register (SAI_BIM). <p>Updated USB section:</p> <ul style="list-style-type: none"> – Updated Table 222: OTG implementation and added note2. <p>Updated USB PHY section:</p> <ul style="list-style-type: none"> – Updated title of Section 33: USB PHY controller (USBPHYC) available on the STM32F7x3xx and STM32F730xx devices only. <p>Updated electronic signature section:</p> <ul style="list-style-type: none"> – Updated Section 35.3: Package data register: '0x1FF0 7BF0' instead of '0x1FFF 7BF0' and Bits 10:8 PKG[2:0] description.

Table 253. Document revision history (continued)

Date	Revision	Changes
26-Jun-2018	3 (continued)	<p>Updated I2C section:</p> <p>Updated Section 26.4.1: I2C block diagram:</p> <ul style="list-style-type: none"> – Removed 'For I2C I/Os supporting 20mA ... refer to section: I2C implementation' paragraph. – Removed 'this independent clock source refer to RCC for more details' paragraph. <p>Updated Section 26.4.4: I2C initialization removing the reference to RCC.</p> <p>Updated Section 26.4.8: I2C master mode master communication initialization (address phase) note.</p> <p>Updated Section 26.6: I2C interrupts:</p> <ul style="list-style-type: none"> – Updated Table 158: I2C Interrupt requests according to new IP guideline (acronym column). – Removed figure: I2C interrupt mapping diagram. – Removed 'depending on the product implementation ..refer to section EXTI' paragraph. – Updated Section 26.7.2: I2C2 control register 2 (I2C_CR2) START bit 13 description and note. <p>Updated Debug support section:</p> <ul style="list-style-type: none"> – Updated Section 34.6.1: MCU device ID code by '0x1000 = Revision A and 1'.

Index

A

ADC_CCR	417
ADC_CDR	420
ADC_CR1	406
ADC_CR2	408
ADC_CSR	416
ADC_DR	416
ADC_HTR	411
ADC_JDRx	415
ADC_JOFRx	411
ADC_JSQR	415
ADC_LTR	412
ADC_SMPR1	410
ADC_SMPR2	411
ADC_SQR1	412
ADC_SQR2	413
ADC_SQR3	414
ADC_SR	405
AES_CR	497
AES_DINR	501
AES_DOUTR	502
AES_IVR	504
AES_KEYRx	502
AES_SR	500

C

CAN_BTR	1151
CAN_ESR	1150
CAN_FA1R	1161
CAN_FFA1R	1160
CAN_FiRx	1162
CAN_FM1R	1159
CAN_FMR	1159
CAN_FS1R	1160
CAN_IER	1149
CAN_MCR	1142
CAN_MSR	1144
CAN_RDHxR	1158
CAN_RDLxR	1158
CAN_RDTxR	1157
CAN_RF0R	1147
CAN_RF1R	1148
CAN_RIxR	1156
CAN_TDHzR	1155
CAN_TDLxR	1155
CAN_TDTxR	1154
CAN_TIxR	1153

CAN_TSR	1145
CRC_CR	268
CRC_DR	267
CRC_IDR	267
CRC_INIT	268
CRC_POL	269

D

DAC_CR	435
DAC_DHR12L1	439
DAC_DHR12L2	440
DAC_DHR12LD	441
DAC_DHR12R1	438
DAC_DHR12R2	440
DAC_DHR12RD	441
DAC_DHR8R1	439
DAC_DHR8R2	440
DAC_DHR8RD	442
DAC_DOR1	442
DAC_DOR2	442
DAC_SR	443
DAC_SWTRIGR	438
DBGMCU_APB2_FZ	1387, 1389
DBGMCU_CR	1386
DBGMCU_IDCODE	1372
DMA_HIFCR	240
DMA_HISR	239
DMA_LIFCR	240
DMA_LISR	238
DMA_SxCR	241
DMA_SxFCR	246
DMA_SxM0AR	245
DMA_SxM1AR	245
DMA_SxNDTR	244
DMA_SxPAR	245

E

EXTI_EMR	260
EXTI_FTSR	261
EXTI_IMR	260
EXTI_PR	262
EXTI_RTSR	261
EXTI_SWIER	262

F

FLITF_FCR	87
-----------------	----

FLITF_FKEYR	85
FLITF_FOPTCR	88, 91
FLITF_FOPTKEYR	85
FLITF_FSR	86
FMC_BCRx	308
FMC_BTRx	310
FMC_BWTR1..4	313
FMC_ECCR	326
FMC_PATT	324
FMC_PCR	321
FMC_PMEM	323
FMC_SDCMR	341
FMC_SDCR1,2	337
FMC_SDRTR	342
FMC_SDSR	343
FMC_SDTR1,2	339
FMC_SR	322

G

GPIOx_AFRH	207
GPIOx_AFRL	206
GPIOx_BSRR	204
GPIOx_IDR	204
GPIOx_LCKR	205
GPIOx_MODER	202
GPIOx_ODR	204
GPIOx_OSPEEDR	203
GPIOx_OTYPER	202
GPIOx_PUPDR	203

I

I2C_CR1	869
I2C_CR2	872
I2C_ICR	881
I2C_ISR	879
I2C_OAR1	875
I2C_OAR2	876
I2C_PECR	882
I2C_RXDR	883
I2C_TIMEOUTR	878
I2C_TIMINGR	877
I2C_TXDR	883
I2Cx_CR2	872
IWDG_KR	762
IWDG_PR	763
IWDG_RLR	764
IWDG_SR	765
IWDG_WINR	766

L

LPTIM_ARR	757
LPTIM_CFGR	752
LPTIM_CMP	756
LPTIM_CNT	757
LPTIM_CR	755
LPTIM_ICR	750
LPTIM_IER	751
LPTIM_ISR	748

O

OTG_CID	1232
OTG_DAINT	1260
OTG_DAINTMSK	1260
OTG_DCFG	1252
OTG_DCTL	1254
OTG_DEACHINT	1264
OTG_DEACHINTMSK	1264
OTG_DIEPCTL0	1267
OTG_DIEPCTLx	1269
OTG_DIEPDMAx	1274
OTG_DIEPEMPMSK	1263
OTG_DIEPINTx	1271
OTG_DIEPMISK	1257
OTG_DIEPTSIZ0	1273
OTG_DIEPTSIZx	1275
OTG_DIEPTXF0	1228
OTG_DIEPTXFx	1237
OTG_DOEPCTL0	1276
OTG_DOEPCTLx	1281
OTG_DOEPDMAx	1280
OTG_DOEPINTx	1277
OTG_DOEPMISK	1258
OTG_DOEPTSIZ0	1279
OTG_DOEPTSIZx	1283
OTG_DSTS	1256
OTG_DTHRCTL	1262
OTG_DTXFSTSx	1274
OTG_DVBUSDIS	1261
OTG_DVBUSPULSE	1261
OTG_GAHBCFG	1208
OTG_GCCFG	1230
OTG_GINTMSK	1222
OTG_GINTSTS	1217
OTG_GLPMCFG	1232
OTG_GOTGCTL	1203
OTG_GOTGINT	1206
OTG_GRSTCTL	1214
OTG_GRXFSIZ	1228
OTG_GRXSTSP	1225
OTG_GRXSTSRR	1225

OTG_GUSBCFG	1210	RCC_APB1ENR	162
OTG_HAINT	1241	RCC_APB1LPENR	171
OTG_HAINTMSK	1242	RCC_APB1RSTR	153
OTG_HCCHARx	1245	RCC_APB2ENR	165
OTG_HCDMAX	1251	RCC_APB2LPENR	175
OTG_HCFG	1238	RCC_APB2RSTR	157
OTG_HCINTMSKx	1249	RCC_BDCR	177
OTG_HCINTx	1247	RCC_CFGR	144
OTG_HCSPLTx	1246	RCC_CIR	146
OTG_HCTSIZx	1250	RCC_CR	139
OTG_HFIR	1239	RCC_CSR	178
OTG_HFNUM	1240	RCC_PLLCFGR	142, 181, 184
OTG_HNPTXFSIZ	1228	RCC_SSCGR	180
OTG_HNPTXSTS	1229	RNG_CR	453
OTG_HPRT	1243	RNG_DR	455
OTG_HPTXFSIZ	1237	RNG_SR	454
OTG_HPTXSTS	1240	RTC_ALRMAR	804
OTG_HS_DIEPEACHMSK1	1265	RTC_ALRMBR	805
OTG_HS_DOEPEACHMSK1	1266	RTC_ALRMBSSR	816
OTG_PCGCCTL	1284	RTC_BKPxR	817
P		RTC_CALR	811
PWR_CR	119	RTC_CR	796
PWR_CSR	121, 123, 125	RTC_DR	795
Q		RTC_ISR	799
QUADSPI_PIR	374	RTC_OR	817
QUADSPI_PSMAR	373	RTC_PRER	802
QUADSPI_PSMKR	373	RTC_SHIFTR	807
QUADSPI_ABR	372	RTC_SSR	806
QUADSPI_AR	371	RTC_TR	794
QUADSPI_CCR	369	RTC_TSDR	809
QUADSPI_CR	363	RTC_TSSSR	810
QUADSPI_DCR	366	RTC_TSTR	808
QUADSPI_DLR	368	RTC_WPR	806
QUADSPI_DR	372	RTC_WUTR	803
QUADSPI_FCR	368		
QUADSPI_LPTR	374		
QUADSPI_SR	367		
R		S	
RCC_AHB1ENR	159	SAI_ACLRFR	1059
RCC_AHB1LPENR	168	SAI_ACR1	1039
RCC_AHB1RSTR	149	SAI_ACR2	1044
RCC_AHB2ENR	160	SAI_ADR	1061
RCC_AHB2LPENR	170	SAI_AFRCR	1048
RCC_AHB2RSTR	152	SAI_AIM	1053
RCC_AHB3ENR	161	SAI_ASLOTR	1051
RCC_AHB3LPENR	171	SAI_ASRR	1055
RCC_AHB3RSTR	153	SAI_BCLRFR	1060

SAI_BSR	1057
SAI_GCR	1039
SDMMC_ARG	1108
SDMMC_CLKCR	1106
SDMMC_DCOUNT	1114
SDMMC_DCTRL	1111
SDMMC_DLEN	1111
SDMMC_DTIMER	1110
SDMMC_FIFO	1120
SDMMC_ICR	1115
SDMMC_MASK	1117
SDMMC_POWER	1106
SDMMC_RESPCMD	1109
SDMMC_RESPx	1109
SDMMC_STA	1114
SPIx_CR1	998
SPIx_CR2	1000
SPIx_CRCPR	1004
SPIx_DR	1004
SPIx_I2SCFGR	1005
SPIx_I2SPR	1007
SPIx_RXCRCR	1004
SPIx_SR	1002
SPIx_TXCRCR	1005
SYSCFG_EXTICR1	213
SYSCFG_EXTICR2	213
SYSCFG_EXTICR3	214
SYSCFG_EXTICR4	215
SYSCFG_MEMRMP	210

T

TIM2_OR	671
TIM5_OR	671
TIMx_ARR	590, 667, 710, 721, 735
TIMx_BDTR	593
TIMx_CCER	586, 665, 708, 719
TIMx_CCMR1	580, 659, 704, 717
TIMx_CCMR2	584, 663
TIMx_CCMR3	598
TIMx_CCR1	591, 668, 710, 721-722
TIMx_CCR2	592, 668, 710
TIMx_CCR3	592, 669
TIMx_CCR4	593, 669
TIMx_CCR5	599
TIMx_CCR6	600
TIMx_CNT	590, 666, 709, 720, 734
TIMx_CR1	569, 650, 699, 714, 731
TIMx_CR2	570, 651, 733
TIMx_DCR	596, 670
TIMx_DIER	575, 656, 702, 715, 733
TIMx_DMAR	597, 670

TIMx_EGR	579, 658, 703, 716, 734
TIMx_PSC	590, 667, 709, 721, 735
TIMx_RCR	591
TIMx_SMCR	573, 653, 700
TIMx_SR	577, 657, 702, 715, 734

U

USART_BRR	938
USART_CR1	929
USART_CR2	932
USART_CR3	935
USART_GTPR	939
USART_ICR	947
USART_ISR	942
USART_RDR	948
USART_RQR	941
USART_RTOR	940
USART_TDR	948

W

WWDG_CFR	772
WWDG_CR	772
WWDG_SR	773

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2018 STMicroelectronics – All rights reserved