



life.augmented

**RM0390**

## **Reference manual**

**STM32F446xx advanced ARM<sup>®</sup>-based 32-bit MCUs**

## **Introduction**

This reference manual targets application developers. It provides complete information on how to use the STM32F446xx microcontroller memory and peripherals.

The STM32F446xx constitute a family of microcontrollers with different memory sizes, packages and peripherals.

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

For information on the ARM<sup>®</sup> Cortex<sup>®</sup>-M4 with FPU core, refer to the Cortex<sup>®</sup>-M4 Technical Reference Manual.

## **Related documents**

Available from STMicroelectronics web site [www.st.com](http://www.st.com):

- STM32F446xx datasheets

For information on the Cortex<sup>®</sup>-M4 with FPU, refer to the STM32F3xx/F4xxx Cortex<sup>®</sup>-M4 with FPU programming manual (PM0214).

# Contents

<b>1</b>	<b>Documentation conventions</b>	<b>51</b>
1.1	List of abbreviations for registers	51
1.2	Glossary	52
1.3	Peripheral availability	52
<b>2</b>	<b>Memory and bus architecture</b>	<b>53</b>
2.1	System architecture	53
2.1.1	I-bus	54
2.1.2	D-bus	54
2.1.3	S-bus	54
2.1.4	DMA memory bus	54
2.1.5	DMA peripheral bus	55
2.1.6	USB OTG HS DMA bus	55
2.1.7	BusMatrix	55
2.1.8	AHB/APB bridges (APB)	55
2.2	Memory organization	56
2.2.1	Introduction	56
2.2.2	Memory map and register boundary addresses	56
2.2.3	Embedded SRAM	59
2.2.4	Flash memory overview	59
2.2.5	Bit banding	59
2.3	Boot configuration	60
<b>3</b>	<b>Embedded Flash memory interface</b>	<b>63</b>
3.1	Introduction	63
3.2	Main features	63
3.3	Embedded Flash memory	64
3.4	Read interface	65
3.4.1	Relation between CPU clock frequency and Flash memory read time	65
3.4.2	Adaptive real-time memory accelerator (ART Accelerator™)	66
3.5	Erase and program operations	68
3.5.1	Unlocking the Flash control register	68
3.5.2	Program/erase parallelism	69

3.5.3	Erase . . . . .	69
3.5.4	Programming . . . . .	70
3.5.5	Interrupts . . . . .	71
3.6	Option bytes . . . . .	71
3.6.1	Description of user option bytes . . . . .	71
3.6.2	Programming user option bytes . . . . .	73
3.6.3	Read protection (RDP) . . . . .	73
3.6.4	Write protections . . . . .	76
3.6.5	Proprietary code readout protection (PCROP) . . . . .	77
3.7	One-time programmable bytes . . . . .	78
3.8	Flash interface registers . . . . .	79
3.8.1	Flash access control register (FLASH_ACR) . . . . .	79
3.8.2	Flash key register (FLASH_KEYR) . . . . .	80
3.8.3	Flash option key register (FLASH_OPTKEYR) . . . . .	80
3.8.4	Flash status register (FLASH_SR) . . . . .	81
3.8.5	Flash control register (FLASH_CR) . . . . .	82
3.8.6	Flash option control register (FLASH_OPTCR) . . . . .	84
3.8.7	Flash interface register map . . . . .	86
<b>4</b>	<b>CRC calculation unit . . . . .</b>	<b>87</b>
4.1	CRC introduction . . . . .	87
4.2	CRC main features . . . . .	87
4.3	CRC functional description . . . . .	87
4.4	CRC registers . . . . .	88
4.4.1	Data register (CRC_DR) . . . . .	88
4.4.2	Independent data register (CRC_IDR) . . . . .	89
4.4.3	Control register (CRC_CR) . . . . .	89
4.4.4	CRC register map . . . . .	90
<b>5</b>	<b>Power controller (PWR) . . . . .</b>	<b>91</b>
5.1	Power supplies . . . . .	91
5.1.1	Independent A/D converter supply and reference voltage . . . . .	92
5.1.2	Battery backup domain . . . . .	92
5.1.3	Voltage regulator . . . . .	94
5.2	Power supply supervisor . . . . .	97
5.2.1	Power-on reset (POR)/power-down reset (PDR) . . . . .	97

---

5.2.2	Brownout reset (BOR) . . . . .	98
5.2.3	Programmable voltage detector (PVD) . . . . .	98
5.3	Low-power modes . . . . .	99
5.3.1	Slowing down system clocks . . . . .	100
5.3.2	Peripheral clock gating . . . . .	100
5.3.3	Low power mode . . . . .	101
5.3.4	Sleep mode . . . . .	101
5.3.5	Stop mode . . . . .	102
5.3.6	Standby mode . . . . .	105
5.3.7	Programming the RTC alternate functions to wake up the device from the Stop and Standby modes . . . . .	107
5.4	Power control registers . . . . .	110
5.4.1	PWR power control register (PWR_CR) . . . . .	110
5.4.2	PWR power control/status register (PWR_CSR) . . . . .	112
5.5	PWR register map . . . . .	114
<b>6</b>	<b>Reset and clock control (RCC) . . . . .</b>	<b>115</b>
6.1	Reset . . . . .	115
6.1.1	System reset . . . . .	115
6.1.2	Power reset . . . . .	115
6.1.3	Backup domain reset . . . . .	116
6.2	Clocks . . . . .	116
6.2.1	HSE clock . . . . .	119
6.2.2	HSI clock . . . . .	120
6.2.3	PLL configuration . . . . .	120
6.2.4	LSE clock . . . . .	121
6.2.5	LSI clock . . . . .	121
6.2.6	System clock (SYSCLK) selection . . . . .	121
6.2.7	Clock security system (CSS) . . . . .	122
6.2.8	RTC/AWU clock . . . . .	122
6.2.9	Watchdog clock . . . . .	123
6.2.10	Clock-out capability . . . . .	123
6.2.11	Internal/external clock measurement using TIM5/TIM11 . . . . .	124
6.3	RCC registers . . . . .	126
6.3.1	RCC clock control register (RCC_CR) . . . . .	126
6.3.2	RCC PLL configuration register (RCC_PLLCFGR) . . . . .	128
6.3.3	RCC clock configuration register (RCC_CFGR) . . . . .	130

6.3.4	RCC clock interrupt register (RCC_CIR) . . . . .	132
6.3.5	RCC AHB1 peripheral reset register (RCC_AHB1RSTR) . . . . .	135
6.3.6	RCC AHB2 peripheral reset register (RCC_AHB2RSTR) . . . . .	137
6.3.7	RCC AHB3 peripheral reset register (RCC_AHB3RSTR) . . . . .	137
6.3.8	RCC APB1 peripheral reset register (RCC_APB1RSTR) . . . . .	138
6.3.9	RCC APB2 peripheral reset register (RCC_APB2RSTR) . . . . .	141
6.3.10	RCC AHB1 peripheral clock enable register (RCC_AHB1ENR) . . . . .	143
6.3.11	RCC AHB2 peripheral clock enable register (RCC_AHB2ENR) . . . . .	144
6.3.12	RCC AHB3 peripheral clock enable register (RCC_AHB3ENR) . . . . .	145
6.3.13	RCC APB1 peripheral clock enable register (RCC_APB1ENR) . . . . .	145
6.3.14	RCC APB2 peripheral clock enable register (RCC_APB2ENR) . . . . .	148
6.3.15	RCC AHB1 peripheral clock enable in low power mode register (RCC_AHB1LPENR) . . . . .	150
6.3.16	RCC AHB2 peripheral clock enable in low power mode register (RCC_AHB2LPENR) . . . . .	152
6.3.17	RCC AHB3 peripheral clock enable in low power mode register (RCC_AHB3LPENR) . . . . .	153
6.3.18	RCC APB1 peripheral clock enable in low power mode register (RCC_APB1LPENR) . . . . .	153
6.3.19	RCC APB2 peripheral clock enabled in low power mode register (RCC_APB2LPENR) . . . . .	157
6.3.20	RCC Backup domain control register (RCC_BDCR) . . . . .	158
6.3.21	RCC clock control & status register (RCC_CSR) . . . . .	160
6.3.22	RCC spread spectrum clock generation register (RCC_SSCGR) . . . . .	161
6.3.23	RCC PLLI2S configuration register (RCC_PLLI2SCFGR) . . . . .	162
6.3.24	RCC PLL configuration register (RCC_PLLSAICFGR) . . . . .	165
6.3.25	RCC Dedicated Clock Configuration Register (RCC_DCKCFGR) . . . . .	166
6.3.26	RCC clocks gated enable register (CKGATENR) . . . . .	168
6.3.27	RCC dedicated clocks configuration register 2 (DCKCFG2) . . . . .	169
6.3.28	RCC register map . . . . .	171
<b>7</b>	<b>General-purpose I/Os (GPIO)</b> . . . . .	<b>175</b>
7.1	GPIO introduction . . . . .	175
7.2	GPIO main features . . . . .	175
7.3	GPIO functional description . . . . .	175
7.3.1	General-purpose I/O (GPIO) . . . . .	177
7.3.2	I/O pin multiplexer and mapping . . . . .	177
7.3.3	I/O port control registers . . . . .	179

---

7.3.4	I/O port data registers . . . . .	180
7.3.5	I/O data bitwise handling . . . . .	180
7.3.6	GPIO locking mechanism . . . . .	180
7.3.7	I/O alternate function input/output . . . . .	181
7.3.8	External interrupt/wakeup lines . . . . .	181
7.3.9	Input configuration . . . . .	181
7.3.10	Output configuration . . . . .	182
7.3.11	Alternate function configuration . . . . .	183
7.3.12	Analog configuration . . . . .	184
7.3.13	Using the OSC32_IN/OSC32_OUT pins as GPIO PC14/PC15 port pins . . . . .	184
7.3.14	Using the OSC_IN/OSC_OUT pins as GPIO PH0/PH1 port pins . . . . .	184
7.3.15	Selection of RTC additional_AF1 and RTC_AF2 alternate functions . . . . .	185
7.4	GPIO registers . . . . .	186
7.4.1	GPIO port mode register (GPIOx_MODER) (x = A..H) . . . . .	186
7.4.2	GPIO port output type register (GPIOx_OTYPER) (x = A..H) . . . . .	187
7.4.3	GPIO port output speed register (GPIOx_OSPEEDR) (x = A..H) . . . . .	187
7.4.4	GPIO port pull-up/pull-down register (GPIOx_PUPDR) (x = A..H) . . . . .	188
7.4.5	GPIO port input data register (GPIOx_IDR) (x = A..H) . . . . .	188
7.4.6	GPIO port output data register (GPIOx_ODR) (x = A..H) . . . . .	189
7.4.7	GPIO port bit set/reset register (GPIOx_BSRR) (x = A..H) . . . . .	189
7.4.8	GPIO port configuration lock register (GPIOx_LCKR) (x = A..H) . . . . .	189
7.4.9	GPIO alternate function low register (GPIOx_AFRL) (x = A..H) . . . . .	191
7.4.10	GPIO alternate function high register (GPIOx_AFRH) (x = A..H) . . . . .	191
7.4.11	GPIO register map . . . . .	192
8	System configuration controller (SYSCFG) . . . . .	194
8.1	I/O compensation cell . . . . .	194
8.2	SYSCFG registers . . . . .	194
8.2.1	SYSCFG memory remap register (SYSCFG_MEMRMP) . . . . .	194
8.2.2	SYSCFG peripheral mode configuration register (SYSCFG_PMC) . . . . .	196
8.2.3	SYSCFG external interrupt configuration register 1 (SYSCFG_EXTICR1) . . . . .	196
8.2.4	SYSCFG external interrupt configuration register 2 (SYSCFG_EXTICR2) . . . . .	197
8.2.5	SYSCFG external interrupt configuration register 3 (SYSCFG_EXTICR3) . . . . .	198

8.2.6	SYSCFG external interrupt configuration register 4 (SYSCFG_EXTICR4) .....	198
8.2.7	Compensation cell control register (SYSCFG_CMPCR) .....	199
8.2.8	SYSCFG configuration register (SYSCFG_CFGR) .....	199
8.2.9	SYSCFG register maps .....	201
<b>9</b>	<b>Direct memory access controller (DMA) .....</b>	<b>202</b>
9.1	DMA introduction .....	202
9.2	DMA main features .....	202
9.3	DMA functional description .....	204
9.3.1	DMA block diagram .....	204
9.3.2	DMA overview .....	204
9.3.3	DMA transactions .....	205
9.3.4	Channel selection .....	205
9.3.5	Arbiter .....	207
9.3.6	DMA streams .....	207
9.3.7	Source, destination and transfer modes .....	207
9.3.8	Pointer incrementation .....	210
9.3.9	Circular mode .....	211
9.3.10	Double buffer mode .....	211
9.3.11	Programmable data width, packing/unpacking, endianness .....	212
9.3.12	Single and burst transfers .....	214
9.3.13	FIFO .....	214
9.3.14	DMA transfer completion .....	217
9.3.15	DMA transfer suspension .....	218
9.3.16	Flow controller .....	218
9.3.17	Summary of the possible DMA configurations .....	219
9.3.18	Stream configuration procedure .....	220
9.3.19	Error management .....	221
9.4	DMA interrupts .....	222
9.5	DMA registers .....	223
9.5.1	DMA low interrupt status register (DMA_LISR) .....	223
9.5.2	DMA high interrupt status register (DMA_HISR) .....	224
9.5.3	DMA low interrupt flag clear register (DMA_LIFCR) .....	225
9.5.4	DMA high interrupt flag clear register (DMA_HIFCR) .....	225
9.5.5	DMA stream x configuration register (DMA_SxCR) (x = 0..7) .....	226
9.5.6	DMA stream x number of data register (DMA_SxNDTR) (x = 0..7) .....	229

---

9.5.7	DMA stream x peripheral address register (DMA_SxPAR) (x = 0..7)	230
9.5.8	DMA stream x memory 0 address register (DMA_SxM0AR) (x = 0..7)	230
9.5.9	DMA stream x memory 1 address register (DMA_SxM1AR) (x = 0..7)	230
9.5.10	DMA stream x FIFO control register (DMA_SxFCR) (x = 0..7)	231
9.5.11	DMA register map	233
<b>10</b>	<b>Interrupts and events</b>	<b>237</b>
10.1	Nested vectored interrupt controller (NVIC)	237
10.1.1	NVIC features	237
10.1.2	SysTick calibration value register	237
10.1.3	Interrupt and exception vectors	237
10.2	External interrupt/event controller (EXTI)	237
10.2.1	EXTI main features	241
10.2.2	EXTI block diagram	242
10.2.3	Wakeup event management	242
10.2.4	Functional description	242
10.2.5	External interrupt/event line mapping	244
10.3	EXTI registers	245
10.3.1	Interrupt mask register (EXTI_IMR)	245
10.3.2	Event mask register (EXTI_EMR)	245
10.3.3	Rising trigger selection register (EXTI_RTSR)	246
10.3.4	Falling trigger selection register (EXTI_FTSR)	246
10.3.5	Software interrupt event register (EXTI_SWIER)	247
10.3.6	Pending register (EXTI_PR)	247
10.3.7	EXTI register map	248
<b>11</b>	<b>Flexible memory controller (FMC)</b>	<b>249</b>
11.1	FMC main features	249
11.2	Block diagram	250
11.3	AHB interface	251
11.3.1	Supported memories and transactions	251
11.4	External device address mapping	253
11.4.1	NOR/PSRAM address mapping	254
11.4.2	NAND Flash memory address mapping	254
11.4.3	SDRAM address mapping	255
11.5	NOR Flash/PSRAM controller	257

11.5.1	External memory interface signals . . . . .	259
11.5.2	Supported memories and transactions . . . . .	260
11.5.3	General timing rules . . . . .	262
11.5.4	NOR Flash/PSRAM controller asynchronous transactions . . . . .	262
11.5.5	Synchronous transactions . . . . .	279
11.5.6	NOR/PSRAM controller registers . . . . .	286
11.6	NAND Flash controller . . . . .	293
11.6.1	External memory interface signals . . . . .	293
11.6.2	NAND Flash supported memories and transactions . . . . .	295
11.6.3	Timing diagrams for NAND Flash memory . . . . .	295
11.6.4	NAND Flash operations . . . . .	296
11.6.5	NAND Flash prewait functionality . . . . .	297
11.6.6	Computation of the error correction code (ECC) in NAND Flash memory . . . . .	298
11.6.7	NAND Flashcontroller registers . . . . .	299
11.7	SDRAM controller . . . . .	305
11.7.1	SDRAM controller main features . . . . .	305
11.7.2	SDRAM External memory interface signals . . . . .	305
11.7.3	SDRAM controller functional description . . . . .	306
11.7.4	Low-power modes . . . . .	312
11.7.5	SDRAM controller registers . . . . .	314
11.8	FMC register map . . . . .	322
<b>12</b>	<b>Quad-SPI interface (QUADSPI) . . . . .</b>	<b>324</b>
12.1	Introduction . . . . .	324
12.2	QUADSPI main features . . . . .	324
12.3	QUADSPI functional description . . . . .	324
12.3.1	QUADSPI block diagram . . . . .	324
12.3.2	QUADSPI pins . . . . .	325
12.3.3	QUADSPI Command sequence . . . . .	326
12.3.4	QUADSPI signal interface protocol modes . . . . .	328
12.3.5	QUADSPI indirect mode . . . . .	330
12.3.6	QUADSPI status flag polling mode . . . . .	332
12.3.7	QUADSPI memory-mapped mode . . . . .	332
12.3.8	QUADSPI Flash memory configuration . . . . .	333
12.3.9	QUADSPI delayed data sampling . . . . .	333
12.3.10	QUADSPI configuration . . . . .	333

---

12.3.11	QUADSPI usage . . . . .	334
12.3.12	Sending the instruction only once . . . . .	336
12.3.13	QUADSPI error management . . . . .	336
12.3.14	QUADSPI busy bit and abort functionality . . . . .	337
12.3.15	nCS behavior . . . . .	337
12.4	QUADSPI interrupts . . . . .	339
12.5	QUADSPI registers . . . . .	340
12.5.1	QUADSPI control register (QUADSPI_CR) . . . . .	340
12.5.2	QUADSPI device configuration register (QUADSPI_DCR) . . . . .	343
12.5.3	QUADSPI status register (QUADSPI_SR) . . . . .	344
12.5.4	QUADSPI flag clear register (QUADSPI_FCR) . . . . .	345
12.5.5	QUADSPI data length register (QUADSPI_DLR) . . . . .	345
12.5.6	QUADSPI communication configuration register (QUADSPI_CCR) . . . . .	346
12.5.7	QUADSPI address register (QUADSPI_AR) . . . . .	348
12.5.8	QUADSPI alternate bytes registers (QUADSPI_ABR) . . . . .	349
12.5.9	QUADSPI data register (QUADSPI_DR) . . . . .	349
12.5.10	QUADSPI polling status mask register (QUADSPI_PSMKR) . . . . .	350
12.5.11	QUADSPI polling status match register (QUADSPI_PSMAR) . . . . .	350
12.5.12	QUADSPI polling interval register (QUADSPI_PIR) . . . . .	351
12.5.13	QUADSPI low-power timeout register (QUADSPI_LPTR) . . . . .	351
12.5.14	QUADSPI register map . . . . .	352
<b>13</b>	<b>Analog-to-digital converter (ADC) . . . . .</b>	<b>353</b>
13.1	ADC introduction . . . . .	353
13.2	ADC main features . . . . .	353
13.3	ADC functional description . . . . .	353
13.3.1	ADC on-off control . . . . .	355
13.3.2	ADC1/2 and ADC3 connectivity . . . . .	356
13.3.3	ADC clock . . . . .	359
13.3.4	Channel selection . . . . .	359
13.3.5	Single conversion mode . . . . .	360
13.3.6	Continuous conversion mode . . . . .	360
13.3.7	Timing diagram . . . . .	360
13.3.8	Analog watchdog . . . . .	361
13.3.9	Scan mode . . . . .	362
13.3.10	Injected channel management . . . . .	362
13.3.11	Discontinuous mode . . . . .	363

13.4	Data alignment . . . . .	364
13.5	Channel-wise programmable sampling time . . . . .	365
13.6	Conversion on external trigger and trigger polarity . . . . .	366
13.7	Fast conversion mode . . . . .	367
13.8	Data management . . . . .	368
13.8.1	Using the DMA . . . . .	368
13.8.2	Managing a sequence of conversions without using the DMA . . . . .	368
13.8.3	Conversions without DMA and without overrun detection . . . . .	369
13.9	Multi ADC mode . . . . .	369
13.9.1	Injected simultaneous mode . . . . .	372
13.9.2	Regular simultaneous mode . . . . .	373
13.9.3	Interleaved mode . . . . .	374
13.9.4	Alternate trigger mode . . . . .	376
13.9.5	Combined regular/injected simultaneous mode . . . . .	378
13.9.6	Combined regular simultaneous + alternate trigger mode . . . . .	378
13.10	Temperature sensor . . . . .	379
13.11	Battery charge monitoring . . . . .	381
13.12	ADC interrupts . . . . .	381
13.13	ADC registers . . . . .	382
13.13.1	ADC status register (ADC_SR) . . . . .	382
13.13.2	ADC control register 1 (ADC_CR1) . . . . .	383
13.13.3	ADC control register 2 (ADC_CR2) . . . . .	385
13.13.4	ADC sample time register 1 (ADC_SMPR1) . . . . .	387
13.13.5	ADC sample time register 2 (ADC_SMPR2) . . . . .	387
13.13.6	ADC injected channel data offset register x (ADC_JOFR $x$ ) ( $x=1..4$ ) . . . . .	388
13.13.7	ADC watchdog higher threshold register (ADC_HTR) . . . . .	388
13.13.8	ADC watchdog lower threshold register (ADC_LTR) . . . . .	389
13.13.9	ADC regular sequence register 1 (ADC_SQR1) . . . . .	389
13.13.10	ADC regular sequence register 2 (ADC_SQR2) . . . . .	390
13.13.11	ADC regular sequence register 3 (ADC_SQR3) . . . . .	391
13.13.12	ADC injected sequence register (ADC_JSQR) . . . . .	392
13.13.13	ADC injected data register x (ADC_JDR $x$ ) ( $x=1..4$ ) . . . . .	392
13.13.14	ADC regular data register (ADC_DR) . . . . .	393
13.13.15	ADC Common status register (ADC_CSR) . . . . .	393
13.13.16	ADC common control register (ADC_CCR) . . . . .	394

---

13.13.17	ADC common regular data register for dual and triple modes (ADC_CDR) . . . . .	397
13.13.18	ADC register map . . . . .	397
<b>14</b>	<b>Digital-to-analog converter (DAC) . . . . .</b>	<b>400</b>
14.1	DAC introduction . . . . .	400
14.2	DAC main features . . . . .	400
14.3	DAC functional description . . . . .	401
14.3.1	DAC channel enable . . . . .	401
14.3.2	DAC output buffer enable . . . . .	402
14.3.3	DAC data format . . . . .	402
14.3.4	DAC conversion . . . . .	403
14.3.5	DAC output voltage . . . . .	404
14.3.6	DAC trigger selection . . . . .	404
14.3.7	DMA request . . . . .	405
14.3.8	Noise generation . . . . .	405
14.3.9	Triangle-wave generation . . . . .	406
14.4	Dual DAC channel conversion . . . . .	407
14.4.1	Independent trigger without wave generation . . . . .	408
14.4.2	Independent trigger with single LFSR generation . . . . .	408
14.4.3	Independent trigger with different LFSR generation . . . . .	408
14.4.4	Independent trigger with single triangle generation . . . . .	409
14.4.5	Independent trigger with different triangle generation . . . . .	409
14.4.6	Simultaneous software start . . . . .	409
14.4.7	Simultaneous trigger without wave generation . . . . .	410
14.4.8	Simultaneous trigger with single LFSR generation . . . . .	410
14.4.9	Simultaneous trigger with different LFSR generation . . . . .	410
14.4.10	Simultaneous trigger with single triangle generation . . . . .	411
14.4.11	Simultaneous trigger with different triangle generation . . . . .	411
14.5	DAC registers . . . . .	412
14.5.1	DAC control register (DAC_CR) . . . . .	412
14.5.2	DAC software trigger register (DAC_SWTRIGR) . . . . .	415
14.5.3	DAC channel1 12-bit right-aligned data holding register (DAC_DHR12R1) . . . . .	415
14.5.4	DAC channel1 12-bit left aligned data holding register (DAC_DHR12L1) . . . . .	416
14.5.5	DAC channel1 8-bit right aligned data holding register (DAC_DHR8R1) . . . . .	416

14.5.6	DAC channel2 12-bit right aligned data holding register (DAC_DHR12R2) .....	417
14.5.7	DAC channel2 12-bit left aligned data holding register (DAC_DHR12L2) .....	417
14.5.8	DAC channel2 8-bit right-aligned data holding register (DAC_DHR8R2) .....	417
14.5.9	Dual DAC 12-bit right-aligned data holding register (DAC_DHR12RD) .....	418
14.5.10	DUAL DAC 12-bit left aligned data holding register (DAC_DHR12LD) .....	418
14.5.11	DUAL DAC 8-bit right aligned data holding register (DAC_DHR8RD) .....	419
14.5.12	DAC channel1 data output register (DAC_DOR1) .....	419
14.5.13	DAC channel2 data output register (DAC_DOR2) .....	419
14.5.14	DAC status register (DAC_SR) .....	420
14.5.15	DAC register map .....	421
<b>15</b>	<b>Digital camera interface (DCMI) .....</b>	<b>422</b>
15.1	DCMI introduction .....	422
15.2	DCMI main features .....	422
15.3	DCMI clocks .....	422
15.4	DCMI functional overview .....	423
15.4.1	DCMI block diagram .....	423
15.4.2	DMA interface .....	424
15.4.3	DCMI physical interface .....	424
15.4.4	Synchronization .....	426
15.4.5	Capture modes .....	429
15.4.6	Crop feature .....	430
15.4.7	JPEG format .....	431
15.4.8	FIFO .....	431
15.5	Data format description .....	432
15.5.1	Data formats .....	432
15.5.2	Monochrome format .....	432
15.5.3	RGB format .....	433
15.5.4	YCbCr format .....	433
15.5.5	YCbCr format - Y only .....	433
15.5.6	Half resolution image extraction .....	434
15.6	DCMI interrupts .....	434

---

15.7	DCMI register description . . . . .	434
15.7.1	DCMI control register (DCMI_CR) . . . . .	434
15.7.2	DCMI status register (DCMI_SR) . . . . .	438
15.7.3	DCMI raw interrupt status register (DCMI_RIS) . . . . .	439
15.7.4	DCMI interrupt enable register (DCMI_IER) . . . . .	440
15.7.5	DCMI masked interrupt status register (DCMI_MIS) . . . . .	441
15.7.6	DCMI interrupt clear register (DCMI_ICR) . . . . .	442
15.7.7	DCMI embedded synchronization code register (DCMI_ESCR) . . . . .	443
15.7.8	DCMI embedded synchronization unmask register (DCMI_ESUR) . . . . .	444
15.7.9	DCMI crop window start (DCMI_CWSTRT) . . . . .	445
15.7.10	DCMI crop window size (DCMI_CWSIZE) . . . . .	445
15.7.11	DCMI data register (DCMI_DR) . . . . .	446
15.7.12	DCMI register map . . . . .	447
<b>16</b>	<b>Advanced-control timers (TIM1&amp;TIM8) . . . . .</b>	<b>448</b>
16.1	TIM1&TIM8 introduction . . . . .	448
16.2	TIM1&TIM8 main features . . . . .	448
16.3	TIM1&TIM8 functional description . . . . .	450
16.3.1	Time-base unit . . . . .	450
16.3.2	Counter modes . . . . .	452
16.3.3	Repetition counter . . . . .	461
16.3.4	Clock selection . . . . .	464
16.3.5	Capture/compare channels . . . . .	467
16.3.6	Input capture mode . . . . .	470
16.3.7	PWM input mode . . . . .	471
16.3.8	Forced output mode . . . . .	471
16.3.9	Output compare mode . . . . .	472
16.3.10	PWM mode . . . . .	473
16.3.11	Complementary outputs and dead-time insertion . . . . .	476
16.3.12	Using the break function . . . . .	478
16.3.13	Clearing the OCxREF signal on an external event . . . . .	481
16.3.14	6-step PWM generation . . . . .	482
16.3.15	One-pulse mode . . . . .	483
16.3.16	Encoder interface mode . . . . .	484
16.3.17	Timer input XOR function . . . . .	487
16.3.18	Interfacing with Hall sensors . . . . .	487
16.3.19	TIMx and external trigger synchronization . . . . .	489

16.3.20	Timer synchronization . . . . .	492
16.3.21	Debug mode . . . . .	492
16.4	TIM1&TIM8 registers . . . . .	493
16.4.1	TIM1&TIM8 control register 1 (TIMx_CR1) . . . . .	493
16.4.2	TIM1&TIM8 control register 2 (TIMx_CR2) . . . . .	494
16.4.3	TIM1&TIM8 slave mode control register (TIMx_SMCR) . . . . .	496
16.4.4	TIM1&TIM8 DMA/interrupt enable register (TIMx_DIER) . . . . .	498
16.4.5	TIM1&TIM8 status register (TIMx_SR) . . . . .	500
16.4.6	TIM1&TIM8 event generation register (TIMx_EGR) . . . . .	501
16.4.7	TIM1&TIM8 capture/compare mode register 1 (TIMx_CCMR1) . . . . .	503
16.4.8	TIM1&TIM8 capture/compare mode register 2 (TIMx_CCMR2) . . . . .	506
16.4.9	TIM1&TIM8 capture/compare enable register (TIMx_CCER) . . . . .	507
16.4.10	TIM1&TIM8 counter (TIMx_CNT) . . . . .	511
16.4.11	TIM1&TIM8 prescaler (TIMx_PSC) . . . . .	511
16.4.12	TIM1&TIM8 auto-reload register (TIMx_ARR) . . . . .	511
16.4.13	TIM1&TIM8 repetition counter register (TIMx_RCR) . . . . .	512
16.4.14	TIM1&TIM8 capture/compare register 1 (TIMx_CCR1) . . . . .	512
16.4.15	TIM1&TIM8 capture/compare register 2 (TIMx_CCR2) . . . . .	513
16.4.16	TIM1&TIM8 capture/compare register 3 (TIMx_CCR3) . . . . .	513
16.4.17	TIM1&TIM8 capture/compare register 4 (TIMx_CCR4) . . . . .	514
16.4.18	TIM1&TIM8 break and dead-time register (TIMx_BDTR) . . . . .	514
16.4.19	TIM1&TIM8 DMA control register (TIMx_DCR) . . . . .	516
16.4.20	TIM1&TIM8 DMA address for full transfer (TIMx_DMAR) . . . . .	517
16.4.21	TIM1&TIM8 register map . . . . .	518
17	<b>General-purpose timers (TIM2 to TIM5) . . . . .</b>	<b>520</b>
17.1	TIM2 to TIM5 introduction . . . . .	520
17.2	TIM2 to TIM5 main features . . . . .	520
17.3	TIM2 to TIM5 functional description . . . . .	521
17.3.1	Time-base unit . . . . .	521
17.3.2	Counter modes . . . . .	523
17.3.3	Clock selection . . . . .	532
17.3.4	Capture/compare channels . . . . .	535
17.3.5	Input capture mode . . . . .	537
17.3.6	PWM input mode . . . . .	538
17.3.7	Forced output mode . . . . .	539
17.3.8	Output compare mode . . . . .	540

---

17.3.9	PWM mode .....	541
17.3.10	One-pulse mode .....	544
17.3.11	Clearing the OC <sub>x</sub> REF signal on an external event .....	545
17.3.12	Encoder interface mode .....	546
17.3.13	Timer input XOR function .....	549
17.3.14	Timers and external trigger synchronization .....	549
17.3.15	Timer synchronization .....	552
17.3.16	Debug mode .....	557
17.4	TIM2 to TIM5 registers .....	558
17.4.1	TIMx control register 1 (TIMx_CR1) .....	558
17.4.2	TIMx control register 2 (TIMx_CR2) .....	560
17.4.3	TIMx slave mode control register (TIMx_SMCR) .....	561
17.4.4	TIMx DMA/Interrupt enable register (TIMx_DIER) .....	563
17.4.5	TIMx status register (TIMx_SR) .....	564
17.4.6	TIMx event generation register (TIMx_EGR) .....	566
17.4.7	TIMx capture/compare mode register 1 (TIMx_CCMR1) .....	567
17.4.8	TIMx capture/compare mode register 2 (TIMx_CCMR2) .....	570
17.4.9	TIMx capture/compare enable register (TIMx_CCER) .....	571
17.4.10	TIMx counter (TIMx_CNT) .....	573
17.4.11	TIMx prescaler (TIMx_PSC) .....	573
17.4.12	TIMx auto-reload register (TIMx_ARR) .....	573
17.4.13	TIMx capture/compare register 1 (TIMx_CCR1) .....	574
17.4.14	TIMx capture/compare register 2 (TIMx_CCR2) .....	574
17.4.15	TIMx capture/compare register 3 (TIMx_CCR3) .....	575
17.4.16	TIMx capture/compare register 4 (TIMx_CCR4) .....	575
17.4.17	TIMx DMA control register (TIMx_DCR) .....	576
17.4.18	TIMx DMA address for full transfer (TIMx_DMAR) .....	576
17.4.19	TIM2 option register (TIM2_OR) .....	577
17.4.20	TIM5 option register (TIM5_OR) .....	578
17.4.21	TIMx register map .....	579
<b>18</b>	<b>General-purpose timers (TIM9 to TIM14) .....</b>	<b>581</b>
18.1	TIM9 to TIM14 introduction .....	581
18.2	TIM9 to TIM14 main features .....	581
18.2.1	TIM9/TIM12 main features .....	581
18.2.2	TIM10/TIM11 and TIM13/TIM14 main features .....	582
18.3	TIM9 to TIM14 functional description .....	584

18.3.1	Time-base unit . . . . .	584
18.3.2	Counter modes . . . . .	586
18.3.3	Clock selection . . . . .	589
18.3.4	Capture/compare channels . . . . .	591
18.3.5	Input capture mode . . . . .	592
18.3.6	PWM input mode (only for TIM9/12) . . . . .	593
18.3.7	Forced output mode . . . . .	594
18.3.8	Output compare mode . . . . .	595
18.3.9	PWM mode . . . . .	596
18.3.10	One-pulse mode . . . . .	597
18.3.11	TIM9/12 external trigger synchronization . . . . .	599
18.3.12	Timer synchronization (TIM9/12) . . . . .	602
18.3.13	Debug mode . . . . .	602
18.4	TIM9 and TIM12 registers . . . . .	602
18.4.1	TIM9/12 control register 1 (TIMx_CR1) . . . . .	602
18.4.2	TIM9/12 slave mode control register (TIMx_SMCR) . . . . .	604
18.4.3	TIM9/12 Interrupt enable register (TIMx_DIER) . . . . .	605
18.4.4	TIM9/12 status register (TIMx_SR) . . . . .	606
18.4.5	TIM9/12 event generation register (TIMx_EGR) . . . . .	608
18.4.6	TIM9/12 capture/compare mode register 1 (TIMx_CCMR1) . . . . .	608
18.4.7	TIM9/12 capture/compare enable register (TIMx_CCER) . . . . .	612
18.4.8	TIM9/12 counter (TIMx_CNT) . . . . .	613
18.4.9	TIM9/12 prescaler (TIMx_PSC) . . . . .	613
18.4.10	TIM9/12 auto-reload register (TIMx_ARR) . . . . .	613
18.4.11	TIM9/12 capture/compare register 1 (TIMx_CCR1) . . . . .	614
18.4.12	TIM9/12 capture/compare register 2 (TIMx_CCR2) . . . . .	614
18.4.13	TIM9/12 register map . . . . .	615
18.5	TIM10/11/13/14 registers . . . . .	617
18.5.1	TIM10/11/13/14 control register 1 (TIMx_CR1) . . . . .	617
18.5.2	TIM10/11/13/14 Interrupt enable register (TIMx_DIER) . . . . .	618
18.5.3	TIM10/11/13/14 status register (TIMx_SR) . . . . .	618
18.5.4	TIM10/11/13/14 event generation register (TIMx_EGR) . . . . .	619
18.5.5	TIM10/11/13/14 capture/compare mode register 1 (TIMx_CCMR1) . . . . .	620
18.5.6	TIM10/11/13/14 capture/compare enable register (TIMx_CCER) . . . . .	623
18.5.7	TIM10/11/13/14 counter (TIMx_CNT) . . . . .	624

---

18.5.8	TIM10/11/13/14 prescaler (TIMx_PSC) . . . . .	624
18.5.9	TIM10/11/13/14 auto-reload register (TIMx_ARR) . . . . .	624
18.5.10	TIM10/11/13/14 capture/compare register 1 (TIMx_CCR1) . . . . .	625
18.5.11	TIM11 option register 1 (TIM11_OR) . . . . .	625
18.5.12	TIM10/11/13/14 register map . . . . .	626
<b>19</b>	<b>Basic timers (TIM6&amp;TIM7) . . . . .</b>	<b>628</b>
19.1	TIM6&TIM7 introduction . . . . .	628
19.2	TIM6&TIM7 main features . . . . .	628
19.3	TIM6&TIM7 functional description . . . . .	629
19.3.1	Time-base unit . . . . .	629
19.3.2	Counting mode . . . . .	631
19.3.3	Clock source . . . . .	633
19.3.4	Debug mode . . . . .	634
19.4	TIM6&TIM7 registers . . . . .	634
19.4.1	TIM6&TIM7 control register 1 (TIMx_CR1) . . . . .	634
19.4.2	TIM6&TIM7 control register 2 (TIMx_CR2) . . . . .	636
19.4.3	TIM6&TIM7 DMA/Interrupt enable register (TIMx_DIER) . . . . .	636
19.4.4	TIM6&TIM7 status register (TIMx_SR) . . . . .	637
19.4.5	TIM6&TIM7 event generation register (TIMx_EGR) . . . . .	637
19.4.6	TIM6&TIM7 counter (TIMx_CNT) . . . . .	637
19.4.7	TIM6&TIM7 prescaler (TIMx_PSC) . . . . .	638
19.4.8	TIM6&TIM7 auto-reload register (TIMx_ARR) . . . . .	638
19.4.9	TIM6&TIM7 register map . . . . .	639
<b>20</b>	<b>Independent watchdog (IWDG) . . . . .</b>	<b>640</b>
20.1	IWDG introduction . . . . .	640
20.2	IWDG main features . . . . .	640
20.3	IWDG functional description . . . . .	640
20.3.1	Hardware watchdog . . . . .	640
20.3.2	Register access protection . . . . .	640
20.3.3	Debug mode . . . . .	641
20.4	IWDG registers . . . . .	642
20.4.1	Key register (IWDG_KR) . . . . .	642
20.4.2	Prescaler register (IWDG_PR) . . . . .	643
20.4.3	Reload register (IWDG_RLR) . . . . .	644

20.4.4	Status register (IWDG_SR) . . . . .	644
20.4.5	IWDG register map . . . . .	645
<b>21</b>	<b>Window watchdog (WWDG) . . . . .</b>	<b>646</b>
21.1	WWDG introduction . . . . .	646
21.2	WWDG main features . . . . .	646
21.3	WWDG functional description . . . . .	646
21.4	How to program the watchdog timeout . . . . .	648
21.5	Debug mode . . . . .	649
21.6	WWDG registers . . . . .	650
21.6.1	Control register (WWDG_CR) . . . . .	650
21.6.2	Configuration register (WWDG_CFR) . . . . .	651
21.6.3	Status register (WWDG_SR) . . . . .	651
21.6.4	WWDG register map . . . . .	652
<b>22</b>	<b>Real-time clock (RTC) . . . . .</b>	<b>653</b>
22.1	Introduction . . . . .	653
22.2	RTC main features . . . . .	653
22.3	RTC functional description . . . . .	655
22.3.1	Clock and prescalers . . . . .	655
22.3.2	Real-time clock and calendar . . . . .	655
22.3.3	Programmable alarms . . . . .	656
22.3.4	Periodic auto-wakeup . . . . .	656
22.3.5	RTC initialization and configuration . . . . .	657
22.3.6	Reading the calendar . . . . .	659
22.3.7	Resetting the RTC . . . . .	660
22.3.8	RTC synchronization . . . . .	660
22.3.9	RTC reference clock detection . . . . .	661
22.3.10	RTC coarse digital calibration . . . . .	661
22.3.11	RTC smooth digital calibration . . . . .	662
22.3.12	Timestamp function . . . . .	664
22.3.13	Tamper detection . . . . .	665
22.3.14	Calibration clock output . . . . .	667
22.3.15	Alarm output . . . . .	667
22.4	RTC and low power modes . . . . .	668
22.5	RTC interrupts . . . . .	668

---

22.6	RTC registers . . . . .	670
22.6.1	RTC time register (RTC_TR) . . . . .	670
22.6.2	RTC date register (RTC_DR) . . . . .	671
22.6.3	RTC control register (RTC_CR) . . . . .	672
22.6.4	RTC initialization and status register (RTC_ISR) . . . . .	674
22.6.5	RTC prescaler register (RTC_PRER) . . . . .	676
22.6.6	RTC wakeup timer register (RTC_WUTR) . . . . .	677
22.6.7	RTC calibration register (RTC_CALIBR) . . . . .	677
22.6.8	RTC alarm A register (RTC_ALRMAR) . . . . .	679
22.6.9	RTC alarm B register (RTC_ALRMBR) . . . . .	680
22.6.10	RTC write protection register (RTC_WPR) . . . . .	681
22.6.11	RTC sub second register (RTC_SSR) . . . . .	681
22.6.12	RTC shift control register (RTC_SHIFTR) . . . . .	682
22.6.13	RTC time stamp time register (RTC_TSTR) . . . . .	683
22.6.14	RTC time stamp date register (RTC_TSDR) . . . . .	683
22.6.15	RTC timestamp sub second register (RTC_TSSSR) . . . . .	684
22.6.16	RTC calibration register (RTC_CALR) . . . . .	684
22.6.17	RTC tamper and alternate function configuration register (RTC_TAFCR) . . . . .	685
22.6.18	RTC alarm A sub second register (RTC_ALRMASSR) . . . . .	687
22.6.19	RTC alarm B sub second register (RTC_ALRMBSSR) . . . . .	688
22.6.20	RTC backup registers (RTC_BKPxR) . . . . .	689
22.6.21	RTC register map . . . . .	690
23	<b>Fast-mode Plus Inter-integrated circuit (FMPI2C) interface . . . . .</b>	<b>692</b>
23.1	Introduction . . . . .	692
23.2	FMPi2C main features . . . . .	692
23.3	FMPi2C implementation . . . . .	693
23.4	FMPi2C functional description . . . . .	693
23.4.1	FMPi2C block diagram . . . . .	694
23.4.2	FMPi2C clock requirements . . . . .	695
23.4.3	Mode selection . . . . .	695
23.4.4	FMPi2C initialization . . . . .	697
23.4.5	Software reset . . . . .	701
23.4.6	Data transfer . . . . .	702
23.4.7	FMPi2C slave mode . . . . .	704
23.4.8	FMPi2C master mode . . . . .	713

23.4.9	FMPI2C_TIMINGR register configuration examples . . . . .	725
23.4.10	SMBus specific features . . . . .	725
23.4.11	SMBus initialization . . . . .	728
23.4.12	SMBus: FMPI2C_TIMEOUTTR register configuration examples . . . . .	730
23.4.13	SMBus slave mode . . . . .	731
23.4.14	Error conditions . . . . .	737
23.4.15	DMA requests . . . . .	739
23.4.16	Debug mode . . . . .	740
23.5	FMPI2C low-power modes . . . . .	740
23.6	FMPI2C interrupts . . . . .	741
23.7	FMPI2C registers . . . . .	742
23.7.1	Control register 1 (FMPI2C_CR1) . . . . .	742
23.7.2	Control register 2 (FMPI2C_CR2) . . . . .	745
23.7.3	Own address 1 register (FMPI2C_OAR1) . . . . .	748
23.7.4	Own address 2 register (FMPI2C_OAR2) . . . . .	749
23.7.5	Timing register (FMPI2C_TIMINGR) . . . . .	750
23.7.6	Timeout register (FMPI2C_TIMEOUTTR) . . . . .	751
23.7.7	Interrupt and status register (FMPI2C_ISR) . . . . .	752
23.7.8	Interrupt clear register (FMPI2C_ICR) . . . . .	754
23.7.9	PEC register (FMPI2C_PECR) . . . . .	755
23.7.10	Receive data register (FMPI2C_RXDR) . . . . .	756
23.7.11	Transmit data register (FMPI2C_TXDR) . . . . .	756
23.7.12	FMPI2C register map . . . . .	757
<b>24</b>	<b>Inter-integrated circuit (I<sup>2</sup>C) interface . . . . .</b>	<b>759</b>
24.1	I <sup>2</sup> C introduction . . . . .	759
24.2	I <sup>2</sup> C main features . . . . .	760
24.3	I <sup>2</sup> C functional description . . . . .	761
24.3.1	Mode selection . . . . .	761
24.3.2	I <sup>2</sup> C slave mode . . . . .	762
24.3.3	I <sup>2</sup> C master mode . . . . .	765
24.3.4	Error conditions . . . . .	771
24.3.5	Programmable noise filter . . . . .	772
24.3.6	SDA/SCL line control . . . . .	773
24.3.7	SMBus . . . . .	773
24.3.8	DMA requests . . . . .	776

---

24.3.9	Packet error checking .....	777
24.4	I <sup>2</sup> C interrupts .....	778
24.5	I <sup>2</sup> C debug mode .....	780
24.6	I <sup>2</sup> C registers .....	780
24.6.1	I <sup>2</sup> C Control register 1 (I2C_CR1) .....	780
24.6.2	I <sup>2</sup> C Control register 2 (I2C_CR2) .....	782
24.6.3	I <sup>2</sup> C Own address register 1 (I2C_OAR1) .....	784
24.6.4	I <sup>2</sup> C Own address register 2 (I2C_OAR2) .....	784
24.6.5	I <sup>2</sup> C Data register (I2C_DR) .....	785
24.6.6	I <sup>2</sup> C Status register 1 (I2C_SR1) .....	785
24.6.7	I <sup>2</sup> C Status register 2 (I2C_SR2) .....	789
24.6.8	I <sup>2</sup> C Clock control register (I2C_CCR) .....	790
24.6.9	I <sup>2</sup> C TRISE register (I2C_TRISE) .....	791
24.6.10	I <sup>2</sup> C FLTR register (I2C_FLTR) .....	792
24.6.11	I <sup>2</sup> C register map .....	793
<b>25</b>	<b>Universal synchronous asynchronous receiver transmitter (USART) .....</b>	<b>794</b>
25.1	USART introduction .....	794
25.2	USART main features .....	795
25.3	USART implementation .....	796
25.4	USART functional description .....	796
25.4.1	USART character description .....	799
25.4.2	Transmitter .....	800
25.4.3	Receiver .....	803
25.4.4	Fractional baud rate generation .....	808
25.4.5	USART receiver tolerance to clock deviation .....	817
25.4.6	Multiprocessor communication .....	818
25.4.7	Parity control .....	820
25.4.8	LIN (local interconnection network) mode .....	821
25.4.9	USART synchronous mode .....	823
25.4.10	Single-wire half-duplex communication .....	825
25.4.11	Smartcard .....	826
25.4.12	IrDA SIR ENDEC block .....	828
25.4.13	Continuous communication using DMA .....	830
25.4.14	Hardware flow control .....	832

25.5	USART interrupts .....	834
25.6	USART registers .....	835
25.6.1	Status register (USART_SR) .....	835
25.6.2	Data register (USART_DR) .....	838
25.6.3	Baud rate register (USART_BRR) .....	838
25.6.4	Control register 1 (USART_CR1) .....	839
25.6.5	Control register 2 (USART_CR2) .....	841
25.6.6	Control register 3 (USART_CR3) .....	842
25.6.7	Guard time and prescaler register (USART_GTPR) .....	844
25.6.8	USART register map .....	845
<b>26</b>	<b>Serial peripheral interface/ inter-IC sound (SPI/I2S) .....</b>	<b>846</b>
26.1	Introduction .....	846
26.1.1	SPI main features .....	846
26.1.2	SPI extended features .....	847
26.1.3	I2S features .....	847
26.2	SPI/I2S implementation .....	847
26.3	SPI functional description .....	848
26.3.1	General description .....	848
26.3.2	Communications between one master and one slave .....	849
26.3.3	Standard multi-slave communication .....	852
26.3.4	Multi-master communication .....	853
26.3.5	Slave select (NSS) pin management .....	853
26.3.6	Communication formats .....	855
26.3.7	SPI configuration .....	857
26.3.8	Procedure for enabling SPI .....	857
26.3.9	Data transmission and reception procedures .....	858
26.3.10	Procedure for disabling the SPI .....	860
26.3.11	Communication using DMA (direct memory addressing) .....	861
26.3.12	SPI status flags .....	863
26.3.13	SPI error flags .....	864
26.4	SPI special features .....	865
26.4.1	TI mode .....	865
26.4.2	CRC calculation .....	866
26.5	SPI interrupts .....	868
26.6	I <sup>2</sup> S functional description .....	869

---

26.6.1	I <sup>2</sup> S general description . . . . .	869
26.6.2	I <sup>2</sup> S full-duplex . . . . .	870
26.6.3	Supported audio protocols . . . . .	871
26.6.4	Clock generator . . . . .	878
26.6.5	I <sup>2</sup> S master mode . . . . .	880
26.6.6	I <sup>2</sup> S slave mode . . . . .	882
26.6.7	I <sup>2</sup> S status flags . . . . .	883
26.6.8	I <sup>2</sup> S error flags . . . . .	884
26.6.9	I <sup>2</sup> S interrupts . . . . .	885
26.6.10	DMA features . . . . .	885
26.7	SPI and I <sup>2</sup> S registers . . . . .	886
26.7.1	SPI control register 1 (SPI_CR1) (not used in I <sup>2</sup> S mode) . . . . .	886
26.7.2	SPI control register 2 (SPI_CR2) . . . . .	888
26.7.3	SPI status register (SPI_SR) . . . . .	889
26.7.4	SPI data register (SPI_DR) . . . . .	891
26.7.5	SPI CRC polynomial register (SPI_CRCPR) (not used in I <sup>2</sup> S mode) . . . . .	891
26.7.6	SPI RX CRC register (SPI_RXCRCR) (not used in I <sup>2</sup> S mode) . . . . .	892
26.7.7	SPI TX CRC register (SPI_TXCRCR) (not used in I <sup>2</sup> S mode) . . . . .	892
26.7.8	SPI_I <sup>2</sup> S configuration register (SPI_I2SCFGR) . . . . .	893
26.7.9	SPI_I <sup>2</sup> S prescaler register (SPI_I2SPR) . . . . .	894
26.7.10	SPI register map . . . . .	896
27	<b>SPDIF receiver interface (SPDIFRX) . . . . .</b>	<b>897</b>
27.1	SPDIFRX interface introduction . . . . .	897
27.2	SPDIFRX main features . . . . .	897
27.3	SPDIFRX functional description . . . . .	897
27.3.1	S/PDIF protocol (IEC-60958) . . . . .	898
27.3.2	SPDIFRX decoder (SPDIFRX_DC) . . . . .	900
27.3.3	SPDIFRX tolerance to clock deviation . . . . .	904
27.3.4	SPDIFRX synchronization . . . . .	904
27.3.5	SPDIFRX handling . . . . .	906
27.3.6	Data reception management . . . . .	908
27.3.7	Dedicated control flow . . . . .	910
27.3.8	Reception errors . . . . .	910
27.3.9	Clocking strategy . . . . .	913
27.3.10	Symbol clock generation . . . . .	914

27.3.11	DMA Interface . . . . .	915
27.3.12	Interrupt Generation . . . . .	916
27.3.13	Register protection . . . . .	917
27.4	Programming procedures . . . . .	917
27.4.1	Initialization phase . . . . .	918
27.4.2	Handling of interrupts coming from SPDIFRX . . . . .	919
27.4.3	Handling of interrupts coming from DMA . . . . .	919
27.5	SPDIFRX interface registers . . . . .	920
27.5.1	Control register (SPDIFRX_CR) . . . . .	920
27.5.2	Interrupt mask register (SPDIFRX_IMR) . . . . .	923
27.5.3	Status register (SPDIFRX_SR) . . . . .	924
27.5.4	Interrupt flag clear register (SPDIFRX_IFCR) . . . . .	926
27.5.5	Data input register (SPDIFRX_DR) . . . . .	927
27.5.6	Data input register (SPDIFRX_DR) . . . . .	928
27.5.7	Data input register (SPDIFRX_DR) . . . . .	929
27.5.8	Channel status register (SPDIFRX_CSR) . . . . .	930
27.5.9	Debug Information register (SPDIFRX_DIR) . . . . .	931
27.5.10	SPDIFRX version register (SPDIFRX_VERR) . . . . .	932
27.5.11	SPDIFRX identification register (SPDIFRX_IDR) . . . . .	932
27.5.12	SPDIFRX size identification register (SPDIFRX_SIDR) . . . . .	933
27.5.13	SPDIFRX interface register map . . . . .	934
<b>28</b>	<b>Serial audio interface (SAI) . . . . .</b>	<b>935</b>
28.1	Introduction . . . . .	935
28.2	SAI main features . . . . .	936
28.3	SAI functional description . . . . .	937
28.3.1	SAI block diagram . . . . .	937
28.3.2	SAI pins and internal signals . . . . .	938
28.3.3	Main SAI modes . . . . .	938
28.3.4	SAI synchronization mode . . . . .	939
28.3.5	Audio data size . . . . .	940
28.3.6	Frame synchronization . . . . .	941
28.3.7	Slot configuration . . . . .	944
28.3.8	SAI clock generator . . . . .	946
28.3.9	Internal FIFOs . . . . .	948
28.3.10	AC'97 link controller . . . . .	950
28.3.11	SPDIF output . . . . .	952

---

28.3.12	Specific features . . . . .	954
28.3.13	Error flags . . . . .	958
28.3.14	Disabling the SAI . . . . .	961
28.3.15	SAI DMA interface . . . . .	962
28.4	SAI interrupts . . . . .	962
28.5	SAI registers . . . . .	964
28.5.1	Global configuration register (SAI_GCR) . . . . .	964
28.5.2	Configuration register 1 (SAI_ACR1 / SAI_BCR1) . . . . .	964
28.5.3	Configuration register 2 (SAI_ACR2 / SAI_BCR2) . . . . .	967
28.5.4	Frame configuration register (SAI_AFRCR / SAI_BFRCR) . . . . .	969
28.5.5	Slot register (SAI_ASLOTR / SAI_BSLOTR) . . . . .	971
28.5.6	Interrupt mask register 2 (SAI_AIM / SAI_BIM) . . . . .	972
28.5.7	Status register (SAI_ASR / SAI_BSR) . . . . .	973
28.5.8	Clear flag register (SAI_ACLRFR / SAI_BCLRFR) . . . . .	975
28.5.9	Data register (SAI_ADR / SAI_BDR) . . . . .	976
28.5.10	SAI register map . . . . .	977
<b>29</b>	<b>Secure digital input/output interface (SDIO) . . . . .</b>	<b>979</b>
29.1	SDIO main features . . . . .	979
29.2	SDIO bus topology . . . . .	979
29.3	SDIO functional description . . . . .	981
29.3.1	SDIO adapter . . . . .	983
29.3.2	SDIO APB2 interface . . . . .	994
29.4	Card functional description . . . . .	995
29.4.1	Card identification mode . . . . .	995
29.4.2	Card reset . . . . .	996
29.4.3	Operating voltage range validation . . . . .	996
29.4.4	Card identification process . . . . .	996
29.4.5	Block write . . . . .	997
29.4.6	Block read . . . . .	998
29.4.7	Stream access, stream write and stream read (MultiMediaCard only) . . . . .	998
29.4.8	Erase: group erase and sector erase . . . . .	1000
29.4.9	Wide bus selection or deselection . . . . .	1000
29.4.10	Protection management . . . . .	1000
29.4.11	Card status register . . . . .	1004
29.4.12	SD status register . . . . .	1007

29.4.13	SD I/O mode . . . . .	1011
29.4.14	Commands and responses . . . . .	1012
29.5	Response formats . . . . .	1015
29.5.1	R1 (normal response command) . . . . .	1016
29.5.2	R1b . . . . .	1016
29.5.3	R2 (CID, CSD register) . . . . .	1016
29.5.4	R3 (OCR register) . . . . .	1017
29.5.5	R4 (Fast I/O) . . . . .	1017
29.5.6	R4b . . . . .	1017
29.5.7	R5 (interrupt request) . . . . .	1018
29.5.8	R6 . . . . .	1018
29.6	SDIO I/O card-specific operations . . . . .	1019
29.6.1	SDIO I/O read wait operation by SDIO_D2 signalling . . . . .	1019
29.6.2	SDIO read wait operation by stopping SDIO_CK . . . . .	1020
29.6.3	SDIO suspend/resume operation . . . . .	1020
29.6.4	SDIO interrupts . . . . .	1020
29.7	HW flow control . . . . .	1020
29.8	SDIO registers . . . . .	1021
29.8.1	SDIO power control register (SDIO_POWER) . . . . .	1021
29.8.2	SDIO clock control register (SDIO_CLKCR) . . . . .	1021
29.8.3	SDIO argument register (SDIO_ARG) . . . . .	1023
29.8.4	SDIO command register (SDIO_CMD) . . . . .	1023
29.8.5	SDIO command response register (SDIO_RESPCMD) . . . . .	1024
29.8.6	SDIO response 1..4 register (SDIO_RESPx) . . . . .	1024
29.8.7	SDIO data timer register (SDIO_DTIMER) . . . . .	1025
29.8.8	SDIO data length register (SDIO_DLEN) . . . . .	1026
29.8.9	SDIO data control register (SDIO_DCTRL) . . . . .	1026
29.8.10	SDIO data counter register (SDIO_DCOUNT) . . . . .	1028
29.8.11	SDIO status register (SDIO_STA) . . . . .	1028
29.8.12	SDIO interrupt clear register (SDIO_ICR) . . . . .	1029
29.8.13	SDIO mask register (SDIO_MASK) . . . . .	1031
29.8.14	SDIO FIFO counter register (SDIO_FIFOCNT) . . . . .	1033
29.8.15	SDIO data FIFO register (SDIO_FIFO) . . . . .	1034
29.8.16	SDIO register map . . . . .	1035
<b>30</b>	<b>Controller area network (bxCAN) . . . . .</b>	<b>1037</b>
30.1	Introduction . . . . .	1037

---

30.2	bxCAN main features . . . . .	1037
30.3	bxCAN general description . . . . .	1038
30.3.1	CAN 2.0B active core . . . . .	1038
30.3.2	Control, status and configuration registers . . . . .	1038
30.3.3	Tx mailboxes . . . . .	1038
30.3.4	Acceptance filters . . . . .	1039
30.4	bxCAN operating modes . . . . .	1039
30.4.1	Initialization mode . . . . .	1040
30.4.2	Normal mode . . . . .	1040
30.4.3	Sleep mode (low-power) . . . . .	1040
30.5	Test mode . . . . .	1041
30.5.1	Silent mode . . . . .	1042
30.5.2	Loop back mode . . . . .	1042
30.5.3	Loop back combined with silent mode . . . . .	1042
30.6	Behavior in debug mode . . . . .	1043
30.7	bxCAN functional description . . . . .	1043
30.7.1	Transmission handling . . . . .	1043
30.7.2	Time triggered communication mode . . . . .	1045
30.7.3	Reception handling . . . . .	1045
30.7.4	Identifier filtering . . . . .	1047
30.7.5	Message storage . . . . .	1051
30.7.6	Error management . . . . .	1053
30.7.7	Bit timing . . . . .	1053
30.8	bxCAN interrupts . . . . .	1056
30.9	CAN registers . . . . .	1057
30.9.1	Register access protection . . . . .	1057
30.9.2	CAN control and status registers . . . . .	1057
30.9.3	CAN mailbox registers . . . . .	1067
30.9.4	CAN filter registers . . . . .	1074
30.9.5	bxCAN register map . . . . .	1078
31	<b>USB on-the-go full-speed/high-speed (OTG_FS/OTG_HS) . . . . .</b>	<b>1082</b>
31.1	Introduction . . . . .	1082
31.2	USB_OTG main features . . . . .	1084
31.2.1	General features . . . . .	1084
31.2.2	Host-mode features . . . . .	1085

31.2.3	Peripheral-mode features . . . . .	1085
31.3	USB_OTG Implementation . . . . .	1086
31.4	USB OTG functional description . . . . .	1086
31.4.1	USB OTG block diagram . . . . .	1086
31.4.2	OTG core . . . . .	1087
31.4.3	Full-speed OTG PHY . . . . .	1088
31.4.4	Embedded full speed OTG PHY . . . . .	1088
31.4.5	High-speed OTG PHY . . . . .	1089
31.4.6	External Full-speed OTG PHY using the I <sup>2</sup> C interface . . . . .	1089
31.5	OTG dual role device (DRD) . . . . .	1089
31.5.1	ID line detection . . . . .	1089
31.5.2	HNP dual role device . . . . .	1090
31.5.3	SRP dual role device . . . . .	1090
31.6	USB peripheral . . . . .	1090
31.6.1	SRP-capable peripheral . . . . .	1091
31.6.2	Peripheral states . . . . .	1091
31.6.3	Peripheral endpoints . . . . .	1092
31.7	USB host . . . . .	1095
31.7.1	SRP-capable host . . . . .	1095
31.7.2	USB host states . . . . .	1096
31.7.3	Host channels . . . . .	1097
31.7.4	Host scheduler . . . . .	1098
31.8	SOF trigger . . . . .	1099
31.8.1	Host SOFs . . . . .	1100
31.8.2	Peripheral SOFs . . . . .	1100
31.9	Power options . . . . .	1100
31.10	Dynamic update of the OTG_HFIR register . . . . .	1101
31.11	USB data FIFOs . . . . .	1101
31.11.1	Peripheral FIFO architecture . . . . .	1102
31.11.2	Host FIFO architecture . . . . .	1103
31.11.3	FIFO RAM allocation . . . . .	1104
31.12	OTG_FS system performance . . . . .	1106
31.13	OTG_FS/OTG_HS interrupts . . . . .	1106
31.14	OTG_FS/OTG_HS control and status registers . . . . .	1108
31.14.1	CSR memory map . . . . .	1108

31.15 OTG_FS/OTG_HS registers .....	1114
31.15.1 OTG control and status register (OTG_GOTGCTL) .....	1114
31.15.2 OTG interrupt register (OTG_GOTGINT) .....	1117
31.15.3 OTG AHB configuration register (OTG_GAHBCFG) .....	1118
31.15.4 OTG USB configuration register (OTG_GUSBCFG) .....	1120
31.15.5 OTG reset register (OTG_GRSTCTL) .....	1124
31.15.6 OTG core interrupt register (OTG_GINTSTS) .....	1126
31.15.7 OTG interrupt mask register (OTG_GINTMSK) .....	1131
31.15.8 OTG_FS Receive status debug read/OTG status read and pop registers (OTG_GRXSTSR/OTG_GRXSTSP) .....	1135
31.15.9 OTG Receive FIFO size register (OTG_GRXFSIZ) .....	1136
31.15.10 OTG Host non-periodic transmit FIFO size register (OTG_HNPTXFSIZ)/Endpoint 0 Transmit FIFO size (OTG_DIEPTXF0) .....	1137
31.15.11 OTG non-periodic transmit FIFO/queue status register (OTG_HNPTXSTS) .....	1138
31.15.12 OTG I <sup>2</sup> C access register (OTG_GI2CCTL) .....	1139
31.15.13 OTG general core configuration register (OTG_GCCFG) .....	1140
31.15.14 OTG core ID register (OTG_CID) .....	1141
31.15.15 OTG core LPM configuration register (OTG_GLPMCFG) .....	1141
31.15.16 OTG Host periodic transmit FIFO size register (OTG_HPTXFSIZ) .....	1145
31.15.17 OTG device IN endpoint transmit FIFO size register (OTG_DIEPTXF <sub>x</sub> ) ( $x = 1..5[\text{FS}] / 8[\text{HS}]$ , where $x$ is the FIFO_number) .....	1145
31.15.18 Host-mode registers .....	1146
31.15.19 OTG Host configuration register (OTG_HCFG) .....	1146
31.15.20 OTG Host frame interval register (OTG_HFIR) .....	1147
31.15.21 OTG Host frame number/frame time remaining register (OTG_HFNUM) .....	1148
31.15.22 OTG_Host periodic transmit FIFO/queue status register (OTG_HPTXSTS) .....	1148
31.15.23 OTG Host all channels interrupt register (OTG_HAINT) .....	1149
31.15.24 OTG Host all channels interrupt mask register (OTG_HAINTMSK) .....	1150
31.15.25 OTG Host port control and status register (OTG_HPRT) .....	1150
31.15.26 OTG Host channel-x characteristics register (OTG_HCCHAR <sub>x</sub> ) ( $x = 0..15[\text{HS}] / 11[\text{FS}]$ , where $x = \text{Channel\_number}$ ) .....	1153
31.15.27 OTG Host channel-x split control register (OTG_HCSPLTx) ( $x = 0..15$ , where $x = \text{Channel\_number}$ ) .....	1154

31.15.28 OTG Host channel-x interrupt register (OTG_HCINTx) (x = 0..15[HS] / 11[FS], where x = Channel_number) .....	1155
31.15.29 OTG Host channel-x interrupt mask register (OTG_HCINTMSKx) (x = 0..15[HS] / 11[FS], where x = Channel_number) .....	1157
31.15.30 OTG Host channel-x transfer size register (OTG_HCTSIZx) (x = 0..15[HS] / 11[FS], where x = Channel_number) .....	1158
31.15.31 OTG Host channel-x DMA address register (OTG_HCDMAX) (x = 0..15, where x = Channel_number) .....	1160
31.15.32 Device-mode registers .....	1160
31.15.33 OTG device configuration register (OTG_DCFG) .....	1160
31.15.34 OTG device control register (OTG_DCTL) .....	1162
31.15.35 OTG device status register (OTG_DSTS) .....	1165
31.15.36 OTG device IN endpoint common interrupt mask register (OTG_DIEPMSK) .....	1166
31.15.37 OTG device OUT endpoint common interrupt mask register (OTG_DOEPMSK) .....	1167
31.15.38 OTG device all endpoints interrupt register (OTG_DAINT) .....	1168
31.15.39 OTG all endpoints interrupt mask register (OTG_DAINTMSK) .....	1169
31.15.40 OTG device V <sub>BUS</sub> discharge time register (OTG_DVBUSDIS) .....	1170
31.15.41 OTG device V <sub>BUS</sub> pulsing time register (OTG_DVBUSPULSE) .....	1170
31.15.42 OTG Device threshold control register (OTG_DTHRCTL) .....	1171
31.15.43 OTG device each endpoint interrupt register (OTG_DEACHINT) ...	1172
31.15.44 OTG device IN endpoint FIFO empty interrupt mask register (OTG_DIEPEMPMSK) .....	1172
31.15.45 OTG device each endpoint interrupt register mask (OTG_DEACHINTMSK) .....	1173
31.15.46 OTG device control IN endpoint 0 control register (OTG_DIEPCTL0) .....	1173
31.15.47 OTG device endpoint-x control register (OTG_DIEPCTLx) (x = 1..5[FS] / 0..8[HS], where x = Endpoint_number) .....	1175
31.15.48 OTG device control OUT endpoint 0 control register (OTG_DOEPCTL0) .....	1177
31.15.49 OTG device endpoint-x control register (OTG_DOEPCTLx) (x = 1..5[FS] / 0..8[HS], where x = Endpoint_number) .....	1179
31.15.50 OTG device endpoint-x interrupt register (OTG_DIEPINTx) (x = 0..5[FS] / 0..8[HS], where x = Endpoint_number) .....	1181
31.15.51 OTG device endpoint-x interrupt register (OTG_DOEPINTx) (x = 0..5[FS] / 0..8[HS], where x = Endpoint_number) .....	1183

---

31.15.52	OTG device IN endpoint 0 transfer size register (OTG_DIEPTSIZ0) . . . . .	1184
31.15.53	OTG Device channel-x DMA address register (OTG_DIEPDMAx) ( $x = 0..15$ , where $x = \text{Channel\_number}$ ) . . . . .	1184
31.15.54	OTG Device channel-x DMA address register (OTG_DOEPDMAx) ( $x = 0..15$ , where $x = \text{Channel\_number}$ ) . . . . .	1185
31.15.55	OTG device OUT endpoint 0 transfer size register (OTG_DOEPTSIZ0) . . . . .	1185
31.15.56	OTG device IN endpoint-x transfer size register (OTG_DIEPTSIZx) ( $x = 1..5[\text{FS}] / 8[\text{HS}]$ , where $x = \text{Endpoint\_number}$ ) . . . . .	1186
31.15.57	OTG device IN endpoint transmit FIFO status register (OTG_DTXFSTSx) ( $x = 0..5[\text{FS}] / 8[\text{HS}]$ , where $x = \text{Endpoint\_number}$ ) . . . . .	1187
31.15.58	OTG device OUT endpoint-x transfer size register (OTG_DOEPTSIZx) ( $x = 1..5[\text{FS}] / 8[\text{HS}]$ , where $x = \text{Endpoint\_number}$ ) . . . . .	1188
31.15.59	OTG power and clock gating control register (OTG_PCGCCTL) . . . . .	1189
31.15.60	OTG_FS/OTG_HS register map . . . . .	1190
31.16	OTG_FS/OTG_HS programming model . . . . .	1202
31.16.1	Core initialization . . . . .	1202
31.16.2	Host initialization . . . . .	1203
31.16.3	Device initialization . . . . .	1203
31.16.4	DMA mode . . . . .	1204
31.16.5	Host programming model . . . . .	1204
31.16.6	Device programming model . . . . .	1237
31.16.7	Worst case response time . . . . .	1257
31.16.8	OTG programming model . . . . .	1259
<b>32</b>	<b>HDMI-CEC controller (HDMI-CEC) . . . . .</b>	<b>1266</b>
32.1	Introduction . . . . .	1266
32.2	HDMI-CEC controller main features . . . . .	1266
32.3	HDMI-CEC functional description . . . . .	1267
32.3.1	HDMI-CEC pin . . . . .	1267
32.3.2	HDMI-CEC block diagram . . . . .	1267
32.3.3	Message description . . . . .	1268
32.3.4	Bit timing . . . . .	1268
32.4	Arbitration . . . . .	1269
32.4.1	SFT option bit . . . . .	1270
32.5	Error handling . . . . .	1271

32.5.1	Bit error . . . . .	1271
32.5.2	Message error . . . . .	1271
32.5.3	Bit Rising Error (BRE) . . . . .	1272
32.5.4	Short Bit Period Error (SBPE) . . . . .	1272
32.5.5	Long Bit Period Error (LBPE) . . . . .	1272
32.5.6	Transmission Error Detection (TXERR) . . . . .	1274
32.6	HDMI-CEC interrupts . . . . .	1275
32.7	HDMI-CEC registers . . . . .	1276
32.7.1	CEC control register (CEC_CR) . . . . .	1276
32.7.2	CEC configuration register (CEC_CFGR) . . . . .	1277
32.7.3	CEC Tx data register (CEC_TXDR) . . . . .	1280
32.7.4	CEC Rx Data Register (CEC_RXDR) . . . . .	1280
32.7.5	CEC Interrupt and Status Register (CEC_ISR) . . . . .	1280
32.7.6	CEC interrupt enable register (CEC_IER) . . . . .	1282
32.7.7	HDMI-CEC register map . . . . .	1284
<b>33</b>	<b>Debug support (DBG)</b> . . . . .	<b>1285</b>
33.1	Overview . . . . .	1285
33.2	Reference ARM® documentation . . . . .	1286
33.3	SWJ debug port (serial wire and JTAG) . . . . .	1286
33.3.1	Mechanism to select the JTAG-DP or the SW-DP . . . . .	1287
33.4	Pinout and debug port pins . . . . .	1287
33.4.1	SWJ debug port pins . . . . .	1288
33.4.2	Flexible SWJ-DP pin assignment . . . . .	1288
33.4.3	Internal pull-up and pull-down on JTAG pins . . . . .	1289
33.4.4	Using serial wire and releasing the unused debug pins as GPIOs . . . . .	1289
33.5	STM32F446xx JTAG TAP connection . . . . .	1290
33.6	ID codes and locking mechanism . . . . .	1291
33.6.1	MCU device ID code . . . . .	1291
33.6.2	Boundary scan TAP . . . . .	1291
33.6.3	Cortex®-M4 with FPU TAP . . . . .	1291
33.6.4	Cortex®-M4 with FPU JEDEC-106 ID code . . . . .	1292
33.7	JTAG debug port . . . . .	1292
33.8	SW debug port . . . . .	1294
33.8.1	SW protocol introduction . . . . .	1294
33.8.2	SW protocol sequence . . . . .	1294

---

33.8.3	SW-DP state machine (reset, idle states, ID code) . . . . .	1295
33.8.4	DP and AP read/write accesses . . . . .	1296
33.8.5	SW-DP registers . . . . .	1296
33.8.6	SW-AP registers . . . . .	1297
33.9	AHB-AP (AHB access port) - valid for both JTAG-DP and SW-DP . . . . .	1298
33.10	Core debug . . . . .	1299
33.11	Capability of the debugger host to connect under system reset . . . . .	1300
33.12	FPB (Flash patch breakpoint) . . . . .	1300
33.13	DWT (data watchpoint trigger) . . . . .	1301
33.14	ITM (instrumentation trace macrocell) . . . . .	1301
33.14.1	General description . . . . .	1301
33.14.2	Time stamp packets, synchronization and overflow packets . . . . .	1301
33.15	ETM (Embedded trace macrocell) . . . . .	1303
33.15.1	General description . . . . .	1303
33.15.2	Signal protocol, packet types . . . . .	1303
33.15.3	Main ETM registers . . . . .	1304
33.15.4	Configuration example . . . . .	1304
33.16	MCU debug component (DBGMCU) . . . . .	1304
33.16.1	Debug support for low-power modes . . . . .	1305
33.16.2	Debug support for timers, watchdog, bxCAN and I <sup>2</sup> C . . . . .	1305
33.16.3	Debug MCU configuration register . . . . .	1305
33.16.4	Debug MCU APB1 freeze register (DBGMCU_APB1_FZ) . . . . .	1308
33.16.5	Debug MCU APB2 Freeze register (DBGMCU_APB2_FZ) . . . . .	1310
33.17	TPIU (trace port interface unit) . . . . .	1311
33.17.1	Introduction . . . . .	1311
33.17.2	TRACE pin assignment . . . . .	1312
33.17.3	TPUI formatter . . . . .	1314
33.17.4	TPUI frame synchronization packets . . . . .	1314
33.17.5	Transmission of the synchronization frame packet . . . . .	1314
33.17.6	Synchronous mode . . . . .	1315
33.17.7	Asynchronous mode . . . . .	1315
33.17.8	TRACECLKIN connection in STM32F446xx . . . . .	1315
33.17.9	TPIU registers . . . . .	1315
33.17.10	Example of configuration . . . . .	1316
33.18	DBG register map . . . . .	1317

<b>34</b>	<b>Device electronic signature</b>	<b>1318</b>
34.1	Unique device ID register (96 bits)	1318
34.2	Flash size	1319
34.3	Package data register	1319
<b>35</b>	<b>Revision history</b>	<b>1320</b>

## List of tables

Table 1.	STM32F446xx register boundary addresses . . . . .	56
Table 2.	Boot modes. . . . .	60
Table 3.	Memory mapping vs. Boot mode/physical remap in STM32F446xx . . . . .	61
Table 4.	Flash module organization . . . . .	64
Table 5.	Number of wait states according to CPU clock (HCLK) frequency. . . . .	65
Table 6.	Program/erase parallelism . . . . .	69
Table 7.	Flash interrupt request . . . . .	71
Table 8.	Option byte organization. . . . .	71
Table 9.	Description of the option bytes . . . . .	72
Table 10.	Access versus read protection level . . . . .	75
Table 11.	OTP area organization . . . . .	78
Table 12.	Flash register map and reset value . . . . .	86
Table 13.	CRC calculation unit register map and reset values. . . . .	90
Table 14.	Voltage regulator configuration mode versus device operating mode . . . . .	95
Table 15.	Low-power mode summary . . . . .	100
Table 16.	Sleep-now entry and exit . . . . .	102
Table 17.	Stop operating modes. . . . .	103
Table 18.	Stop mode entry and exit for STM32F446xx . . . . .	105
Table 19.	Standby mode entry and exit . . . . .	106
Table 20.	PWR register map and reset values . . . . .	114
Table 21.	RCC register map and reset values . . . . .	171
Table 22.	Port bit configuration table . . . . .	176
Table 23.	Flexible SWJ-DP pin assignment . . . . .	178
Table 24.	RTC_AF1 pin . . . . .	185
Table 25.	RTC_AF2 pin . . . . .	186
Table 26.	GPIO register map and reset values . . . . .	192
Table 27.	SYSCFG register map and reset values. . . . .	201
Table 28.	DMA1 request mapping . . . . .	206
Table 29.	DMA2 request mapping . . . . .	206
Table 30.	Source and destination address . . . . .	207
Table 31.	Source and destination address registers in double buffer mode (DBM=1) . . . . .	212
Table 32.	Packing/unpacking & endian behavior (bit PINC = MINC = 1) . . . . .	213
Table 33.	Restriction on NDT versus PSIZE and MSIZE . . . . .	213
Table 34.	FIFO threshold configurations . . . . .	215
Table 35.	Possible DMA configurations . . . . .	219
Table 36.	DMA interrupt requests. . . . .	222
Table 37.	DMA register map and reset values . . . . .	233
Table 38.	Vector table for STM32F446xx . . . . .	237
Table 39.	External interrupt/event controller register map and reset values. . . . .	248
Table 40.	NOR/PSRAM bank selection . . . . .	254
Table 41.	NOR/PSRAM External memory address . . . . .	254
Table 42.	NAND memory mapping and timing registers. . . . .	254
Table 43.	NAND bank selection . . . . .	255
Table 44.	SDRAM bank selection. . . . .	255
Table 45.	SDRAM address mapping . . . . .	255
Table 46.	SDRAM address mapping with 8-bit data bus width. . . . .	256
Table 47.	SDRAM address mapping with 16-bit data bus width. . . . .	257
Table 48.	Programmable NOR/PSRAM access parameters . . . . .	258

Table 49.	Non-multiplexed I/O NOR Flash memory . . . . .	259
Table 50.	16-bit multiplexed I/O NOR Flash memory . . . . .	259
Table 51.	Non-multiplexed I/Os PSRAM/SRAM . . . . .	260
Table 52.	16-Bit multiplexed I/O PSRAM . . . . .	260
Table 53.	NOR Flash/PSRAM: example of supported memories and transactions . . . . .	261
Table 54.	FMC_BCRx bit fields . . . . .	264
Table 55.	FMC_BTRx bit fields . . . . .	264
Table 56.	FMC_BCRx bit fields . . . . .	266
Table 57.	FMC_BTRx bit fields . . . . .	266
Table 58.	FMC_BWTRx bit fields . . . . .	267
Table 59.	FMC_BCRx bit fields . . . . .	269
Table 60.	FMC_BTRx bit fields . . . . .	269
Table 61.	FMC_BWTRx bit fields . . . . .	270
Table 62.	FMC_BCRx bit fields . . . . .	271
Table 63.	FMC_BTRx bit fields . . . . .	272
Table 64.	FMC_BWTRx bit fields . . . . .	272
Table 65.	FMC_BCRx bit fields . . . . .	274
Table 66.	FMC_BTRx bit fields . . . . .	274
Table 67.	FMC_BWTRx bit fields . . . . .	275
Table 68.	FMC_BCRx bit fields . . . . .	276
Table 69.	FMC_BTRx bit fields . . . . .	277
Table 70.	FMC_BCRx bit fields . . . . .	282
Table 71.	FMC_BTRx bit fields . . . . .	283
Table 72.	FMC_BCRx bit fields . . . . .	284
Table 73.	FMC_BTRx bit fields . . . . .	285
Table 74.	Programmable NAND Flash access parameters . . . . .	293
Table 75.	8-bit NAND Flash . . . . .	293
Table 76.	16-bit NAND Flash . . . . .	294
Table 77.	Supported memories and transactions . . . . .	295
Table 78.	ECC result relevant bits . . . . .	304
Table 79.	SDRAM signals . . . . .	305
Table 80.	FMC register map . . . . .	322
Table 81.	QUADSPI pins . . . . .	325
Table 82.	QUADSPI interrupt requests . . . . .	339
Table 83.	QUADSPI register map and reset values . . . . .	352
Table 84.	ADC pins . . . . .	355
Table 85.	Analog watchdog channel selection . . . . .	361
Table 86.	Configuring the trigger polarity . . . . .	366
Table 87.	External trigger for regular channels . . . . .	366
Table 88.	External trigger for injected channels . . . . .	367
Table 89.	ADC interrupts . . . . .	381
Table 90.	ADC global register map . . . . .	397
Table 91.	ADC register map and reset values for each ADC . . . . .	397
Table 92.	ADC register map and reset values (common ADC registers) . . . . .	399
Table 93.	DAC pins . . . . .	401
Table 94.	External triggers . . . . .	404
Table 95.	DAC register map . . . . .	421
Table 96.	DCMI external signals . . . . .	424
Table 97.	Positioning of captured data bytes in 32-bit words (8-bit width) . . . . .	425
Table 98.	Positioning of captured data bytes in 32-bit words (10-bit width) . . . . .	425
Table 99.	Positioning of captured data bytes in 32-bit words (12-bit width) . . . . .	426
Table 100.	Positioning of captured data bytes in 32-bit words (14-bit width) . . . . .	426

Table 101.	Data storage in monochrome progressive video format . . . . .	432
Table 102.	Data storage in RGB progressive video format . . . . .	433
Table 103.	Data storage in YCbCr progressive video format . . . . .	433
Table 104.	Data storage in YCbCr progressive video format - Y extraction mode . . . . .	434
Table 105.	DCMI interrupts . . . . .	434
Table 106.	DCMI register map and reset values . . . . .	447
Table 107.	Counting direction versus encoder signals . . . . .	485
Table 108.	TIMx Internal trigger connection . . . . .	498
Table 109.	Output control bits for complementary OC <sub>x</sub> and OC <sub>xN</sub> channels with break feature . . . . .	510
Table 110.	TIM1&TIM8 register map and reset values . . . . .	518
Table 111.	Counting direction versus encoder signals . . . . .	547
Table 112.	TIMx internal trigger connections . . . . .	562
Table 113.	Output control bit for standard OC <sub>x</sub> channels . . . . .	572
Table 114.	TIM2 to TIM5 register map and reset values . . . . .	579
Table 115.	TIMx internal trigger connections . . . . .	605
Table 116.	Output control bit for standard OC <sub>x</sub> channels . . . . .	613
Table 117.	TIM9/12 register map and reset values . . . . .	615
Table 118.	Output control bit for standard OC <sub>x</sub> channels . . . . .	623
Table 119.	TIM10/11/13/14 register map and reset values . . . . .	626
Table 120.	TIM6&TIM7 register map and reset values . . . . .	639
Table 121.	Min/max IWDG timeout period at 32 kHz (LSI) . . . . .	641
Table 122.	IWDG register map and reset values . . . . .	645
Table 123.	WWDG register map and reset values . . . . .	652
Table 124.	Effect of low power modes on RTC . . . . .	668
Table 125.	Interrupt control bits . . . . .	669
Table 126.	RTC register map and reset values . . . . .	690
Table 127.	STM32F446xx FMPI2C implementation . . . . .	693
Table 128.	I2C-SMBUS specification data setup and hold times . . . . .	700
Table 129.	FMPI2C configuration table . . . . .	704
Table 130.	I2C-SMBUS specification clock timings . . . . .	715
Table 131.	SMBus timeout specifications . . . . .	727
Table 132.	SMBUS with PEC configuration . . . . .	729
Table 133.	Examples of TIMEOUTA settings for various FMPI2CCLK frequencies (max t <sub>TIMEOUT</sub> = 25 ms) . . . . .	730
Table 134.	Examples of TIMEOUTB settings for various FMPI2CCLK frequencies . . . . .	730
Table 135.	Examples of TIMEOUTA settings for various FMPI2CCLK frequencies (max t <sub>IDLE</sub> = 50 µs) . . . . .	730
Table 136.	low-power modes . . . . .	740
Table 137.	FMPI2C Interrupt requests . . . . .	741
Table 138.	FMPI2C register map and reset values . . . . .	757
Table 139.	Maximum DNF[3:0] value to be compliant with Thd:dat(max) . . . . .	772
Table 140.	SMBus vs. I2C . . . . .	774
Table 141.	I2C Interrupt requests . . . . .	778
Table 142.	I2C register map and reset values . . . . .	793
Table 143.	USART features . . . . .	796
Table 144.	Noise detection from sampled data . . . . .	807
Table 145.	Error calculation for programmed baud rates at f <sub>PCLK</sub> = 8 MHz or f <sub>PCLK</sub> = 12 MHz, oversampling by 16 . . . . .	810
Table 146.	Error calculation for programmed baud rates at f <sub>PCLK</sub> = 8 MHz or f <sub>PCLK</sub> = 12 MHz, oversampling by 8 . . . . .	810
Table 147.	Error calculation for programmed baud rates at f <sub>PCLK</sub> = 16 MHz or f <sub>PCLK</sub> = 24 MHz, . . . . .	

oversampling by 16 .....	811
Table 148. Error calculation for programmed baud rates at $f_{PCLK} = 16$ MHz or $f_{PCLK} = 24$ MHz, oversampling by 8 .....	812
Table 149. Error calculation for programmed baud rates at $f_{PCLK} = 8$ MHz or $f_{PCLK} = 16$ MHz, oversampling by 16 .....	812
Table 150. Error calculation for programmed baud rates at $f_{PCLK} = 8$ MHz or $f_{PCLK} = 16$ MHz, oversampling by 8 .....	813
Table 151. Error calculation for programmed baud rates at $f_{PCLK} = 30$ MHz or $f_{PCLK} = 60$ MHz, oversampling by 16 .....	814
Table 152. Error calculation for programmed baud rates at $f_{PCLK} = 30$ MHz or $f_{PCLK} = 60$ MHz, oversampling by 8 .....	814
Table 153. Error calculation for programmed baud rates at $f_{PCLK} = 42$ MHz or $f_{PCLK} = 84$ Hz, oversampling by 16 .....	815
Table 154. Error calculation for programmed baud rates at $f_{PCLK} = 42$ MHz or $f_{PCLK} = 84$ MHz, oversampling by 8 .....	816
Table 155. USART receiver tolerance when DIV fraction is 0 .....	818
Table 156. USART receiver tolerance when DIV_Fraction is different from 0 .....	818
Table 157. Frame formats .....	820
Table 158. USART interrupt requests .....	834
Table 159. USART register map and reset values .....	845
Table 160. SPI interrupt requests .....	868
Table 161. Audio-frequency precision using standard 8 MHz HSE .....	879
Table 162. I <sup>2</sup> S interrupt requests .....	885
Table 163. SPI register map and reset values .....	896
Table 164. Transition sequence for preamble .....	903
Table 165. Minimum SPDIFRX_CLK frequency versus audio sampling rate .....	913
Table 166. Conditions of spdifrx_symb_ck generation .....	915
Table 167. Bit field property versus SPDIFRX state .....	917
Table 168. SPDIFRX interface register map and reset values .....	934
Table 169. SAI internal signals .....	938
Table 170. SAI pins .....	938
Table 171. External synchronization selection .....	940
Table 172. Example of possible audio frequency sampling range .....	947
Table 173. SOPD pattern .....	953
Table 174. Parity bit calculation .....	953
Table 175. Audio sampling frequency versus symbol rates .....	954
Table 176. SAI interrupt sources .....	962
Table 177. SAI register map and reset values .....	977
Table 178. SDIO I/O definitions .....	982
Table 179. Command format .....	987
Table 180. Short response format .....	988
Table 181. Long response format .....	988
Table 182. Command path status flags .....	988
Table 183. Data token format .....	991
Table 184. DPSM flags .....	992
Table 185. Transmit FIFO status flags .....	993
Table 186. Receive FIFO status flags .....	993
Table 187. Card status .....	1004
Table 188. SD status .....	1007
Table 189. Speed class code field .....	1008
Table 190. Performance move field .....	1009
Table 191. AU_SIZE field .....	1009

Table 192. Maximum AU size . . . . .	1009
Table 193. Erase size field . . . . .	1010
Table 194. Erase timeout field . . . . .	1010
Table 195. Erase offset field . . . . .	1010
Table 196. Block-oriented write commands . . . . .	1013
Table 197. Block-oriented write protection commands . . . . .	1014
Table 198. Erase commands . . . . .	1014
Table 199. I/O mode commands . . . . .	1014
Table 200. Lock card . . . . .	1015
Table 201. Application-specific commands . . . . .	1015
Table 202. R1 response . . . . .	1016
Table 203. R2 response . . . . .	1016
Table 204. R3 response . . . . .	1017
Table 205. R4 response . . . . .	1017
Table 206. R4b response . . . . .	1017
Table 207. R5 response . . . . .	1018
Table 208. R6 response . . . . .	1019
Table 209. Response type and SDIO_RESPx registers . . . . .	1025
Table 210. SDIO register map . . . . .	1035
Table 211. Transmit mailbox mapping . . . . .	1052
Table 212. Receive mailbox mapping . . . . .	1052
Table 213. bxCAN register map and reset values . . . . .	1078
Table 214. OTG_HS speeds supported . . . . .	1083
Table 215. OTG_FS speeds supported . . . . .	1083
Table 216. Core global control and status registers (CSRs) . . . . .	1108
Table 217. Host-mode control and status registers (CSRs) . . . . .	1109
Table 218. Device-mode control and status registers . . . . .	1111
Table 219. Data FIFO (DFIFO) access register map . . . . .	1114
Table 220. Power and clock gating control and status registers . . . . .	1114
Table 221. TRDT values (FS) . . . . .	1123
Table 222. TRDT values (HS) . . . . .	1123
Table 223. Minimum duration for soft disconnect . . . . .	1164
Table 224. OTG_FS/OTG_HS register map and reset values . . . . .	1190
Table 225. HDMI pin . . . . .	1267
Table 226. Error handling timing parameters . . . . .	1273
Table 227. TXERR timing parameters . . . . .	1274
Table 228. HDMI-CEC interrupts . . . . .	1275
Table 229. HDMI-CEC register map and reset values . . . . .	1284
Table 230. SWJ debug port pins . . . . .	1288
Table 231. Flexible SWJ-DP pin assignment . . . . .	1288
Table 232. JTAG debug port data registers . . . . .	1292
Table 233. 32-bit debug port registers addressed through the shifted value A[3:2] . . . . .	1294
Table 234. Packet request (8-bits) . . . . .	1295
Table 235. ACK response (3 bits) . . . . .	1295
Table 236. DATA transfer (33 bits) . . . . .	1295
Table 237. SW-DP registers . . . . .	1296
Table 238. Cortex®-M4 with FPU AHB-AP registers . . . . .	1298
Table 239. Core debug registers . . . . .	1299
Table 240. Main ITM registers . . . . .	1302
Table 241. Main ETM registers . . . . .	1304
Table 242. Asynchronous TRACE pin assignment . . . . .	1312
Table 243. Synchronous TRACE pin assignment . . . . .	1312

Table 244. Flexible TRACE pin assignment . . . . .	1313
Table 245. Important TPIU registers . . . . .	1316
Table 246. DBG register map and reset values . . . . .	1317
Table 247. Document revision history . . . . .	1320

## List of figures

Figure 1.	System architecture for STM32F446xx devices .....	54
Figure 2.	Flash memory interface connection inside system architecture .....	63
Figure 3.	Sequential 32-bit instruction execution .....	67
Figure 4.	RDP levels .....	76
Figure 5.	PCROP levels .....	78
Figure 6.	CRC calculation unit block diagram .....	87
Figure 7.	Power supply overview for STM32F446xx .....	91
Figure 8.	Backup domain .....	94
Figure 9.	Power-on reset/power-down reset waveform .....	97
Figure 10.	BOR thresholds .....	98
Figure 11.	PVD thresholds .....	99
Figure 12.	Simplified diagram of the reset circuit .....	116
Figure 13.	Clock tree .....	117
Figure 14.	HSE/ LSE clock sources (hardware configuration) .....	119
Figure 15.	Frequency measurement with TIM5 in Input capture mode .....	124
Figure 16.	Frequency measurement with TIM11 in Input capture mode .....	125
Figure 17.	Basic structure of a 5 V tolerant I/O port bit .....	176
Figure 18.	Selecting an alternate function on STM32F446xx .....	179
Figure 19.	Input floating/pull up/pull down configurations .....	182
Figure 20.	Output configuration .....	183
Figure 21.	Alternate function configuration .....	183
Figure 22.	High impedance-analog configuration .....	184
Figure 23.	DMA block diagram .....	204
Figure 24.	Channel selection .....	205
Figure 25.	Peripheral-to-memory mode .....	208
Figure 26.	Memory-to-peripheral mode .....	209
Figure 27.	Memory-to-memory mode .....	210
Figure 28.	FIFO structure .....	215
Figure 29.	External interrupt/event controller block diagram .....	242
Figure 30.	External interrupt/event GPIO mapping .....	244
Figure 31.	FMC block diagram .....	250
Figure 32.	FMC memory banks .....	253
Figure 33.	Mode1 read access waveforms .....	263
Figure 34.	Mode1 write access waveforms .....	263
Figure 35.	ModeA read access waveforms .....	265
Figure 36.	ModeA write access waveforms .....	265
Figure 37.	Mode2 and mode B read access waveforms .....	267
Figure 38.	Mode2 write access waveforms .....	268
Figure 39.	ModeB write access waveforms .....	268
Figure 40.	ModeC read access waveforms .....	270
Figure 41.	ModeC write access waveforms .....	271
Figure 42.	ModeD read access waveforms .....	273
Figure 43.	ModeD write access waveforms .....	273
Figure 44.	Muxed read access waveforms .....	275
Figure 45.	Muxed write access waveforms .....	276
Figure 46.	Asynchronous wait during a read access waveforms .....	278
Figure 47.	Asynchronous wait during a write access waveforms .....	279
Figure 48.	Wait configuration waveforms .....	281

Figure 49.	Synchronous multiplexed read mode waveforms - NOR, PSRAM (CRAM) . . . . .	282
Figure 50.	Synchronous multiplexed write mode waveforms - PSRAM (CRAM) . . . . .	284
Figure 51.	NAND Flash controller waveforms for common memory access . . . . .	296
Figure 52.	Access to non 'CE don't care' NAND-Flash . . . . .	297
Figure 53.	Burst write SDRAM access waveforms . . . . .	307
Figure 54.	Burst read SDRAM access . . . . .	308
Figure 55.	Logic diagram of Read access with RBURST bit set (CAS=1, RPIPE=0) . . . . .	309
Figure 56.	Read access crossing row boundary . . . . .	311
Figure 57.	Write access crossing row boundary . . . . .	311
Figure 58.	Self-refresh mode . . . . .	313
Figure 59.	Power-down mode . . . . .	314
Figure 60.	QUADSPI block diagram when dual-flash mode is disabled . . . . .	324
Figure 61.	QUADSPI block diagram when dual-flash mode is enabled . . . . .	325
Figure 62.	An example of a read command in quad mode . . . . .	326
Figure 63.	An example of a DDR command in quad mode . . . . .	329
Figure 64.	nCS when CKMODE = 0 (T = CLK period) . . . . .	337
Figure 65.	nCS when CKMODE = 1 in SDR mode (T = CLK period) . . . . .	337
Figure 66.	nCS when CKMODE = 1 in DDR mode (T = CLK period) . . . . .	338
Figure 67.	nCS when CKMODE = 1 with an abort (T = CLK period) . . . . .	338
Figure 68.	Single ADC block diagram . . . . .	354
Figure 69.	ADC1 connectivity . . . . .	356
Figure 70.	ADC2 connectivity . . . . .	357
Figure 71.	ADC3 connectivity . . . . .	358
Figure 72.	Timing diagram . . . . .	361
Figure 73.	Analog watchdog's guarded area . . . . .	361
Figure 74.	Injected conversion latency . . . . .	363
Figure 75.	Right alignment of 12-bit data . . . . .	365
Figure 76.	Left alignment of 12-bit data . . . . .	365
Figure 77.	Left alignment of 6-bit data . . . . .	365
Figure 78.	Multi ADC block diagram <sup>(1)</sup> . . . . .	370
Figure 79.	Injected simultaneous mode on 4 channels: dual ADC mode . . . . .	373
Figure 80.	Injected simultaneous mode on 4 channels: triple ADC mode . . . . .	373
Figure 81.	Regular simultaneous mode on 16 channels: dual ADC mode . . . . .	374
Figure 82.	Regular simultaneous mode on 16 channels: triple ADC mode . . . . .	374
Figure 83.	Interleaved mode on 1 channel in continuous conversion mode: dual ADC mode . . . . .	375
Figure 84.	Interleaved mode on 1 channel in continuous conversion mode: triple ADC mode . . . . .	376
Figure 85.	Alternate trigger: injected group of each ADC . . . . .	377
Figure 86.	Alternate trigger: 4 injected channels (each ADC) in discontinuous mode . . . . .	377
Figure 87.	Alternate trigger: injected group of each ADC . . . . .	378
Figure 88.	Alternate + regular simultaneous . . . . .	379
Figure 89.	Case of trigger occurring during injected conversion . . . . .	379
Figure 90.	Temperature sensor and VREFINT channel block diagram . . . . .	380
Figure 91.	DAC channel block diagram . . . . .	401
Figure 92.	DAC output buffer connection . . . . .	402
Figure 93.	Data registers in single DAC channel mode . . . . .	403
Figure 94.	Data registers in dual DAC channel mode . . . . .	403
Figure 95.	Timing diagram for conversion with trigger disabled TEN = 0 . . . . .	404
Figure 96.	DAC LFSR register calculation algorithm . . . . .	406
Figure 97.	DAC conversion (SW trigger enabled) with LFSR wave generation . . . . .	406
Figure 98.	DAC triangle wave generation . . . . .	407
Figure 99.	DAC conversion (SW trigger enabled) with triangle wave generation . . . . .	407
Figure 100.	DCMI block diagram . . . . .	423

---

Figure 101. Top-level block diagram .....	424
Figure 102. DCMI signal waveforms .....	425
Figure 103. Timing diagram .....	427
Figure 104. Frame capture waveforms in snapshot mode .....	429
Figure 105. Frame capture waveforms in continuous grab mode .....	430
Figure 106. Coordinates and size of the window after cropping .....	430
Figure 107. Data capture waveforms .....	431
Figure 108. Pixel raster scan order .....	432
Figure 109. Advanced-control timer block diagram .....	449
Figure 110. Counter timing diagram with prescaler division change from 1 to 2 .....	451
Figure 111. Counter timing diagram with prescaler division change from 1 to 4 .....	451
Figure 112. Counter timing diagram, internal clock divided by 1 .....	452
Figure 113. Counter timing diagram, internal clock divided by 2 .....	453
Figure 114. Counter timing diagram, internal clock divided by 4 .....	453
Figure 115. Counter timing diagram, internal clock divided by N .....	453
Figure 116. Counter timing diagram, update event when ARPE=0 (TIMx_ARR not preloaded) .....	454
Figure 117. Counter timing diagram, update event when ARPE=1 (TIMx_ARR preloaded) .....	454
Figure 118. Counter timing diagram, internal clock divided by 1 .....	456
Figure 119. Counter timing diagram, internal clock divided by 2 .....	456
Figure 120. Counter timing diagram, internal clock divided by 4 .....	457
Figure 121. Counter timing diagram, internal clock divided by N .....	457
Figure 122. Counter timing diagram, update event when repetition counter is not used .....	458
Figure 123. Counter timing diagram, internal clock divided by 1, TIMx_ARR = 0x6 .....	459
Figure 124. Counter timing diagram, internal clock divided by 2 .....	459
Figure 125. Counter timing diagram, internal clock divided by 4, TIMx_ARR=0x36 .....	460
Figure 126. Counter timing diagram, internal clock divided by N .....	460
Figure 127. Counter timing diagram, update event with ARPE=1 (counter underflow) .....	461
Figure 128. Counter timing diagram, update event with ARPE=1 (counter overflow) .....	461
Figure 129. Update rate examples depending on mode and TIMx_RCR register settings .....	463
Figure 130. Control circuit in normal mode, internal clock divided by 1 .....	464
Figure 131. TI2 external clock connection example .....	465
Figure 132. Control circuit in external clock mode 1 .....	466
Figure 133. External trigger input block .....	466
Figure 134. Control circuit in external clock mode 2 .....	467
Figure 135. Capture/compare channel (example: channel 1 input stage) .....	468
Figure 136. Capture/compare channel 1 main circuit .....	468
Figure 137. Output stage of capture/compare channel (channels 1 to 3) .....	469
Figure 138. Output stage of capture/compare channel (channel 4) .....	469
Figure 139. PWM input mode timing .....	471
Figure 140. Output compare mode, toggle on OC1 .....	473
Figure 141. Edge-aligned PWM waveforms (ARR=8) .....	474
Figure 142. Center-aligned PWM waveforms (ARR=8) .....	475
Figure 143. Complementary output with dead-time insertion .....	477
Figure 144. Dead-time waveforms with delay greater than the negative pulse .....	477
Figure 145. Dead-time waveforms with delay greater than the positive pulse .....	477
Figure 146. Output behavior in response to a break .....	480
Figure 147. Clearing TIMx OCxREF .....	481
Figure 148. 6-step generation, COM example (OSSR=1) .....	482
Figure 149. Example of one pulse mode .....	483
Figure 150. Example of counter operation in encoder interface mode .....	486

---

Figure 151. Example of encoder interface mode with TI1FP1 polarity inverted.....	486
Figure 152. Example of Hall sensor interface .....	488
Figure 153. Control circuit in reset mode .....	489
Figure 154. Control circuit in gated mode .....	490
Figure 155. Control circuit in trigger mode.....	491
Figure 156. Control circuit in external clock mode 2 + trigger mode .....	492
Figure 157. General-purpose timer block diagram .....	521
Figure 158. Counter timing diagram with prescaler division change from 1 to 2 .....	522
Figure 159. Counter timing diagram with prescaler division change from 1 to 4 .....	523
Figure 160. Counter timing diagram, internal clock divided by 1 .....	524
Figure 161. Counter timing diagram, internal clock divided by 2 .....	524
Figure 162. Counter timing diagram, internal clock divided by 4 .....	524
Figure 163. Counter timing diagram, internal clock divided by N.....	525
Figure 164. Counter timing diagram, Update event when ARPE=0 (TIMx_ARR not preloaded).....	525
Figure 165. Counter timing diagram, Update event when ARPE=1 (TIMx_ARR preloaded).....	526
Figure 166. Counter timing diagram, internal clock divided by 1 .....	527
Figure 167. Counter timing diagram, internal clock divided by 2 .....	527
Figure 168. Counter timing diagram, internal clock divided by 4 .....	527
Figure 169. Counter timing diagram, internal clock divided by N.....	528
Figure 170. Counter timing diagram, Update event.....	528
Figure 171. Counter timing diagram, internal clock divided by 1, TIMx_ARR=0x6 .....	529
Figure 172. Counter timing diagram, internal clock divided by 2 .....	530
Figure 173. Counter timing diagram, internal clock divided by 4, TIMx_ARR=0x36 .....	530
Figure 174. Counter timing diagram, internal clock divided by N.....	530
Figure 175. Counter timing diagram, Update event with ARPE=1 (counter underflow).....	531
Figure 176. Counter timing diagram, Update event with ARPE=1 (counter overflow).....	531
Figure 177. Control circuit in normal mode, internal clock divided by 1.....	532
Figure 178. TI2 external clock connection example.....	533
Figure 179. Control circuit in external clock mode 1 .....	534
Figure 180. External trigger input block .....	534
Figure 181. Control circuit in external clock mode 2 .....	535
Figure 182. Capture/compare channel (example: channel 1 input stage) .....	536
Figure 183. Capture/compare channel 1 main circuit .....	536
Figure 184. Output stage of capture/compare channel (channel 1).....	537
Figure 185. PWM input mode timing .....	539
Figure 186. Output compare mode, toggle on OC1.....	541
Figure 187. Edge-aligned PWM waveforms (ARR=8).....	542
Figure 188. Center-aligned PWM waveforms (ARR=8).....	543
Figure 189. Example of one-pulse mode .....	544
Figure 190. Clearing TIMx OCxREF .....	546
Figure 191. Example of counter operation in encoder interface mode .....	548
Figure 192. Example of encoder interface mode with TI1FP1 polarity inverted .....	548
Figure 193. Control circuit in reset mode .....	549
Figure 194. Control circuit in gated mode .....	550
Figure 195. Control circuit in trigger mode.....	551
Figure 196. Control circuit in external clock mode 2 + trigger mode .....	552
Figure 197. Master/Slave timer example .....	552
Figure 198. Gating timer 2 with OC1REF of timer 1 .....	553
Figure 199. Gating timer 2 with Enable of timer 1 .....	554
Figure 200. Triggering timer 2 with update of timer 1 .....	555
Figure 201. Triggering timer 2 with Enable of timer 1 .....	556
Figure 202. Triggering timer 1 and 2 with timer 1 TI1 input .....	557

---

Figure 203. General-purpose timer block diagram (TIM9 and TIM12) . . . . .	582
Figure 204. General-purpose timer block diagram (TIM10/11/13/14) . . . . .	583
Figure 205. Counter timing diagram with prescaler division change from 1 to 2 . . . . .	585
Figure 206. Counter timing diagram with prescaler division change from 1 to 4 . . . . .	585
Figure 207. Counter timing diagram, internal clock divided by 1 . . . . .	586
Figure 208. Counter timing diagram, internal clock divided by 2 . . . . .	587
Figure 209. Counter timing diagram, internal clock divided by 4 . . . . .	587
Figure 210. Counter timing diagram, internal clock divided by N . . . . .	587
Figure 211. Counter timing diagram, update event when ARPE=0 (TIMx_ARR not preloaded) . . . . .	588
Figure 212. Counter timing diagram, update event when ARPE=1 (TIMx_ARR preloaded) . . . . .	588
Figure 213. Control circuit in normal mode, internal clock divided by 1 . . . . .	589
Figure 214. TI2 external clock connection example . . . . .	590
Figure 215. Control circuit in external clock mode 1 . . . . .	590
Figure 216. Capture/compare channel (example: channel 1 input stage) . . . . .	591
Figure 217. Capture/compare channel 1 main circuit . . . . .	592
Figure 218. Output stage of capture/compare channel (channel 1) . . . . .	592
Figure 219. PWM input mode timing . . . . .	594
Figure 220. Output compare mode, toggle on OC1 . . . . .	596
Figure 221. Edge-aligned PWM waveforms (ARR=8) . . . . .	597
Figure 222. Example of one pulse mode . . . . .	598
Figure 223. Control circuit in reset mode . . . . .	600
Figure 224. Control circuit in gated mode . . . . .	601
Figure 225. Control circuit in trigger mode . . . . .	601
Figure 226. Basic timer block diagram . . . . .	628
Figure 227. Counter timing diagram with prescaler division change from 1 to 2 . . . . .	630
Figure 228. Counter timing diagram with prescaler division change from 1 to 4 . . . . .	630
Figure 229. Counter timing diagram, internal clock divided by 1 . . . . .	631
Figure 230. Counter timing diagram, internal clock divided by 2 . . . . .	632
Figure 231. Counter timing diagram, internal clock divided by 4 . . . . .	632
Figure 232. Counter timing diagram, internal clock divided by N . . . . .	632
Figure 233. Counter timing diagram, update event when ARPE = 0 (TIMx_ARR not preloaded) . . . . .	633
Figure 234. Counter timing diagram, update event when ARPE=1 (TIMx_ARR preloaded) . . . . .	633
Figure 235. Control circuit in normal mode, internal clock divided by 1 . . . . .	634
Figure 236. Independent watchdog block diagram . . . . .	641
Figure 237. Watchdog block diagram . . . . .	647
Figure 238. Window watchdog timing diagram . . . . .	648
Figure 239. RTC block diagram . . . . .	654
Figure 240. FMPI2C block diagram . . . . .	694
Figure 241. I2C bus protocol . . . . .	696
Figure 242. Setup and hold timings . . . . .	698
Figure 243. FMPI2C initialization flowchart . . . . .	701
Figure 244. Data reception . . . . .	702
Figure 245. Data transmission . . . . .	703
Figure 246. Slave initialization flowchart . . . . .	706
Figure 247. Transfer sequence flowchart for FMPI2C slave transmitter, NOSTRETCH=0 . . . . .	708
Figure 248. Transfer sequence flowchart for FMPI2C slave transmitter, NOSTRETCH=1 . . . . .	709
Figure 249. Transfer bus diagrams for FMPI2C slave transmitter . . . . .	710
Figure 250. Transfer sequence flowchart for slave receiver with NOSTRETCH=0 . . . . .	711

---

Figure 251. Transfer sequence flowchart for slave receiver with NOSTRETCH=1 .....	712
Figure 252. Transfer bus diagrams for FMPI2C slave receiver .....	712
Figure 253. Master clock generation .....	714
Figure 254. Master initialization flowchart .....	716
Figure 255. 10-bit address read access with HEAD10R=0 .....	716
Figure 256. 10-bit address read access with HEAD10R=1 .....	717
Figure 257. Transfer sequence flowchart for FMPI2C master transmitter for $N \leq 255$ bytes .....	718
Figure 258. Transfer sequence flowchart for FMPI2C master transmitter for $N > 255$ bytes .....	719
Figure 259. Transfer bus diagrams for FMPI2C master transmitter .....	720
Figure 260. Transfer sequence flowchart for FMPI2C master receiver for $N \leq 255$ bytes .....	722
Figure 261. Transfer sequence flowchart for FMPI2C master receiver for $N > 255$ bytes .....	723
Figure 262. Transfer bus diagrams for FMPI2C master receiver .....	724
Figure 263. Timeout intervals for $t_{LOW:SEXT}$ , $t_{LOW:MEXT}$ .....	727
Figure 264. Transfer sequence flowchart for SMBus slave transmitter $N$ bytes + PEC .....	731
Figure 265. Transfer bus diagrams for SMBus slave transmitter ( $SBC=1$ ) .....	732
Figure 266. Transfer sequence flowchart for SMBus slave receiver $N$ Bytes + PEC .....	733
Figure 267. Bus transfer diagrams for SMBus slave receiver ( $SBC=1$ ) .....	734
Figure 268. Bus transfer diagrams for SMBus master transmitter .....	735
Figure 269. Bus transfer diagrams for SMBus master receiver .....	737
Figure 270. FMPI2C interrupt mapping diagram .....	742
Figure 271. I2C bus protocol .....	761
Figure 272. I2C block diagram .....	762
Figure 273. Transfer sequence diagram for slave transmitter .....	764
Figure 274. Transfer sequence diagram for slave receiver .....	765
Figure 275. Transfer sequence diagram for master transmitter .....	768
Figure 276. Transfer sequence diagram for master receiver .....	770
Figure 277. I2C interrupt mapping diagram .....	779
Figure 278. USART block diagram .....	798
Figure 279. Word length programming .....	799
Figure 280. Configurable stop bits .....	801
Figure 281. TC/TXE behavior when transmitting .....	802
Figure 282. Start bit detection when oversampling by 16 or 8 .....	803
Figure 283. Data sampling when oversampling by 16 .....	806
Figure 284. Data sampling when oversampling by 8 .....	807
Figure 285. Mute mode using Idle line detection .....	819
Figure 286. Mute mode using address mark detection .....	819
Figure 287. Break detection in LIN mode (11-bit break length - LBDL bit is set) .....	822
Figure 288. Break detection in LIN mode vs. Framing error detection .....	823
Figure 289. USART example of synchronous transmission .....	824
Figure 290. USART data clock timing diagram ( $M=0$ ) .....	824
Figure 291. USART data clock timing diagram ( $M=1$ ) .....	825
Figure 292. RX data setup/hold time .....	825
Figure 293. ISO 7816-3 asynchronous protocol .....	826
Figure 294. Parity error detection using the 1.5 stop bits .....	827
Figure 295. IrDA SIR ENDEC- block diagram .....	829
Figure 296. IrDA data modulation (3/16) -Normal mode .....	829
Figure 297. Transmission using DMA .....	831
Figure 298. Reception using DMA .....	832
Figure 299. Hardware flow control between 2 USARTs .....	832
Figure 300. RTS flow control .....	833
Figure 301. CTS flow control .....	833
Figure 302. USART interrupt mapping diagram .....	835

---

Figure 303. SPI block diagram.....	848
Figure 304. Full-duplex single master/ single slave application.....	849
Figure 305. Half-duplex single master/ single slave application .....	850
Figure 306. Simplex single master/single slave application (master in transmit-only/ slave in receive-only mode) .....	851
Figure 307. Master and three independent slaves.....	852
Figure 308. Multi-master application .....	853
Figure 309. Hardware/software slave select management .....	854
Figure 310. Data clock timing diagram .....	856
Figure 311. TXE/RXNE/BSY behavior in master / full-duplex mode (BIDIMODE=0, RXONLY=0) in the case of continuous transfers .....	859
Figure 312. TXE/RXNE/BSY behavior in slave / full-duplex mode (BIDIMODE=0, RXONLY=0) in the case of continuous transfers .....	860
Figure 313. Transmission using DMA .....	862
Figure 314. Reception using DMA.....	863
Figure 315. TI mode transfer .....	866
Figure 316. I <sup>2</sup> S block diagram .....	869
Figure 317. Full-duplex communication .....	871
Figure 318. I <sup>2</sup> S Philips protocol waveforms (16/32-bit full accuracy, CPOL = 0).....	872
Figure 319. I <sup>2</sup> S Philips standard waveforms (24-bit frame with CPOL = 0).....	872
Figure 320. Transmitting 0x8EAA33 .....	873
Figure 321. Receiving 0x8EAA33 .....	873
Figure 322. I <sup>2</sup> S Philips standard (16-bit extended to 32-bit packet frame with CPOL = 0).....	873
Figure 323. Example of 16-bit data frame extended to 32-bit channel frame .....	874
Figure 324. MSB Justified 16-bit or 32-bit full-accuracy length with CPOL = 0 .....	874
Figure 325. MSB justified 24-bit frame length with CPOL = 0 .....	874
Figure 326. MSB justified 16-bit extended to 32-bit packet frame with CPOL = 0 .....	875
Figure 327. LSB justified 16-bit or 32-bit full-accuracy with CPOL = 0 .....	875
Figure 328. LSB justified 24-bit frame length with CPOL = 0 .....	875
Figure 329. Operations required to transmit 0x3478AE.....	876
Figure 330. Operations required to receive 0x3478AE .....	876
Figure 331. LSB justified 16-bit extended to 32-bit packet frame with CPOL = 0 .....	876
Figure 332. Example of 16-bit data frame extended to 32-bit channel frame .....	877
Figure 333. PCM standard waveforms (16-bit) .....	877
Figure 334. PCM standard waveforms (16-bit extended to 32-bit packet frame).....	877
Figure 335. Audio sampling frequency definition .....	878
Figure 336. I <sup>2</sup> S clock generator architecture .....	878
Figure 337. SPDIFRX block diagram.....	898
Figure 338. S/PDIF Sub-Frame Format.....	898
Figure 339. S/PDIF block format .....	899
Figure 340. S/PDIF Preambles .....	899
Figure 341. Channel coding example .....	900
Figure 342. SPDIFRX decoder .....	901
Figure 343. Noise filtering and edge detection .....	901
Figure 344. Thresholds .....	903
Figure 345. Synchronization flowchart.....	905
Figure 346. Synchronization process scheduling .....	906
Figure 347. SPDIFRX States .....	907
Figure 348. SPDIFRX_DR register format.....	909
Figure 349. Channel/user data format .....	910
Figure 350. S/PDIF overrun error when RXSTEO = 0 .....	912
Figure 351. S/PDIF overrun error when RXSTEO = 1 .....	913

---

Figure 352. SPDIFRX interface interrupt mapping diagram .....	916
Figure 353. Functional block diagram .....	937
Figure 354. Audio frame .....	941
Figure 355. FS role is start of frame + channel side identification (FSDEF = TRIS = 1) .....	943
Figure 356. FS role is start of frame (FSDEF = 0) .....	944
Figure 357. Slot size configuration with FBOFF = 0 in SAI_xSLOTR .....	945
Figure 358. First bit offset .....	945
Figure 359. Audio block clock generator overview .....	946
Figure 360. AC'97 audio frame .....	950
Figure 361. Example of typical AC'97 configuration on devices featuring at least 2 embedded SAIs (three external AC'97 decoders) .....	951
Figure 362. SPDIF format .....	952
Figure 363. SAI_xDR register ordering .....	953
Figure 364. Data companding hardware in an audio block in the SAI .....	956
Figure 365. Tristate strategy on SD output line on an inactive slot .....	957
Figure 366. Tristate on output data line in a protocol like I2S .....	958
Figure 367. Overrun detection error .....	959
Figure 368. FIFO underrun event .....	960
Figure 369. "No response" and "no data" operations .....	980
Figure 370. (Multiple) block read operation .....	980
Figure 371. (Multiple) block write operation .....	980
Figure 372. Sequential read operation .....	981
Figure 373. Sequential write operation .....	981
Figure 374. SDIO block diagram .....	981
Figure 375. SDIO adapter .....	983
Figure 376. Control unit .....	984
Figure 377. SDIO_CK clock dephasing (BYPASS = 0) .....	984
Figure 378. SDIO adapter command path .....	985
Figure 379. Command path state machine (SDIO) .....	986
Figure 380. SDIO command transfer .....	987
Figure 381. Data path .....	989
Figure 382. Data path state machine (DPSM) .....	990
Figure 383. CAN network topology .....	1038
Figure 384. Dual CAN block diagram .....	1039
Figure 385. bxCAN operating modes .....	1041
Figure 386. bxCAN in silent mode .....	1042
Figure 387. bxCAN in loop back mode .....	1042
Figure 388. bxCAN in combined mode .....	1043
Figure 389. Transmit mailbox states .....	1045
Figure 390. Receive FIFO states .....	1046
Figure 391. Filter bank scale configuration - register organization .....	1049
Figure 392. Example of filter numbering .....	1050
Figure 393. Filtering mechanism - example .....	1051
Figure 394. CAN error state diagram .....	1052
Figure 395. Bit timing .....	1054
Figure 396. CAN frames .....	1055
Figure 397. Event flags and interrupt generation .....	1056
Figure 398. Can mailbox registers .....	1068
Figure 399. OTG full-speed block diagram .....	1086
Figure 400. OTG high-speed block diagram .....	1087
Figure 401. OTG_FS A-B device connection .....	1089
Figure 402. USB_FS peripheral-only connection .....	1091

---

Figure 403. USB_FS host-only connection .....	1095
Figure 404. SOF connectivity (SOF trigger output to TIM and ITR1 connection) .....	1099
Figure 405. Updating OTG_HFIR dynamically .....	1101
Figure 406. Device-mode FIFO address mapping and AHB FIFO access mapping .....	1102
Figure 407. Host-mode FIFO address mapping and AHB FIFO access mapping .....	1103
Figure 408. Interrupt hierarchy .....	1107
Figure 409. Transmit FIFO write task .....	1206
Figure 410. Receive FIFO read task .....	1207
Figure 411. Normal bulk/control OUT/SETUP .....	1209
Figure 412. Bulk/control IN transactions .....	1213
Figure 413. Normal interrupt OUT .....	1216
Figure 414. Normal interrupt IN .....	1221
Figure 415. Isochronous OUT transactions .....	1223
Figure 416. Isochronous IN transactions .....	1226
Figure 417. Normal bulk/control OUT/SETUP transactions - DMA .....	1228
Figure 418. Normal bulk/control IN transaction - DMA .....	1230
Figure 419. Normal interrupt OUT transactions - DMA mode .....	1231
Figure 420. Normal interrupt IN transactions - DMA mode .....	1232
Figure 421. Normal isochronous OUT transaction - DMA mode .....	1233
Figure 422. Normal isochronous IN transactions - DMA mode .....	1234
Figure 423. Receive FIFO packet read .....	1240
Figure 424. Processing a SETUP packet .....	1242
Figure 425. Bulk OUT transaction .....	1248
Figure 426. TRDT max timing case .....	1259
Figure 427. A-device SRP .....	1260
Figure 428. B-device SRP .....	1261
Figure 429. A-device HNP .....	1262
Figure 430. B-device HNP .....	1264
Figure 431. HDMI-CEC block diagram .....	1267
Figure 432. Message structure .....	1268
Figure 433. Blocks .....	1268
Figure 434. Bit timings .....	1269
Figure 435. Signal free time .....	1269
Figure 436. Arbitration phase .....	1270
Figure 437. SFT of three nominal bit periods .....	1270
Figure 438. Error bit timing .....	1271
Figure 439. Error handling .....	1273
Figure 440. TXERR detection .....	1274
Figure 441. Block diagram of STM32 MCU and Cortex®-M4 with FPU-level debug support .....	1285
Figure 442. SWJ debug port .....	1287
Figure 443. JTAG TAP connections .....	1290
Figure 444. TPIU block diagram .....	1311

# 1 Documentation conventions

## 1.1 List of abbreviations for registers

The following abbreviations are used in register descriptions:

read/write (rw)	Software can read and write to 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 (rc_w1)	Software can read as well as clear this bit by writing 1. Writing '0' has no effect on the bit value.
read/clear (rc_w0)	Software can read as well as clear this bit by writing 0. Writing '1' has no effect on the bit value.
read/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 (rs)	Software can read as well as set this bit. Writing '0' has no effect on the bit value.
Reserved (Res.)	Reserved bit, must be kept at reset value.

## 1.2 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®-M4 with FPU 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.
- **I-Code**: this bus connects the Instruction bus of the CPU core to the Flash instruction interface. Prefetch is performed on this bus.
- **D-Code**: this bus connects the D-Code bus (literal load and debug access) of the CPU to the Flash data interface.
- **Option bytes**: product configuration bits stored in the Flash memory.
- **OBL**: option byte loader.
- **AHB**: advanced high-performance bus.
- **CPU**: refers to the Cortex®-M4 with FPUcore.

## 1.3 Peripheral availability

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

## 2 Memory and bus architecture

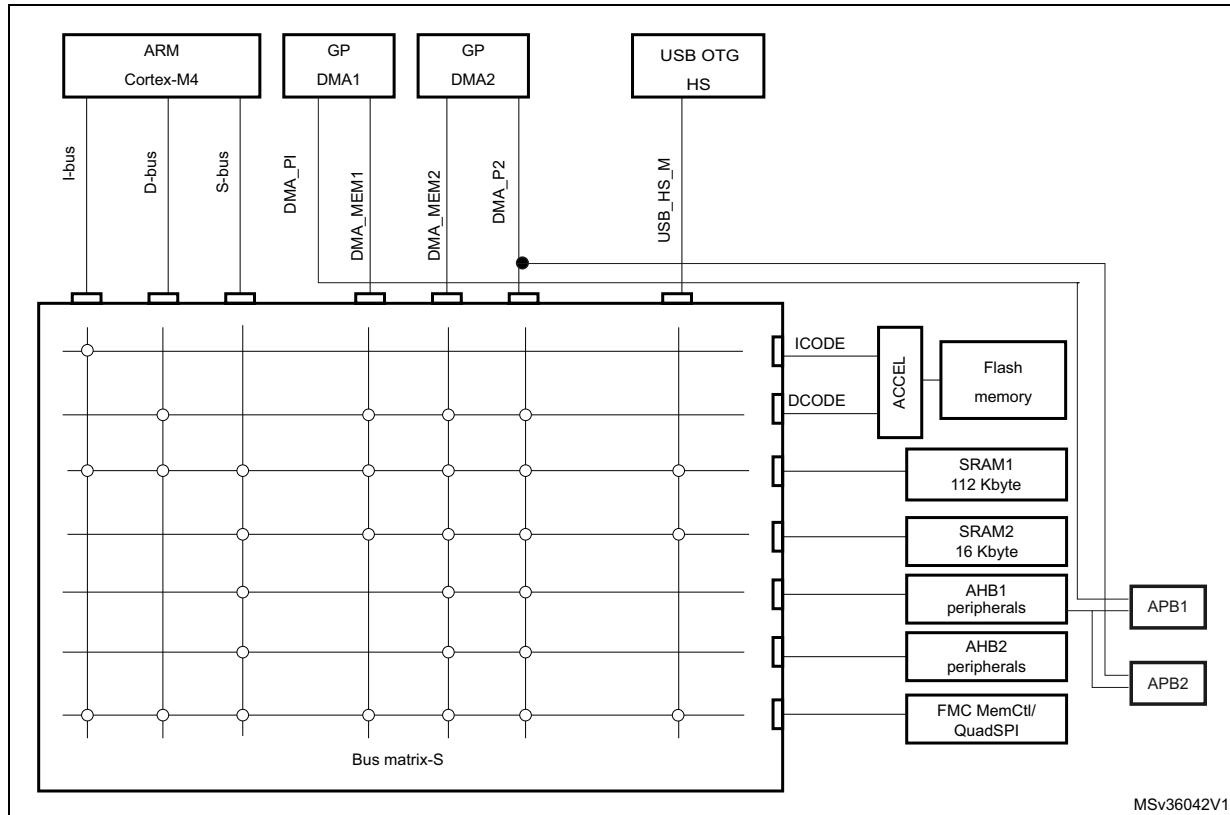
### 2.1 System architecture

In STM32F446xx, the main system consists of 32-bit multilayer AHB bus matrix that interconnects:

- Seven masters:
  - Cortex®-M4 with FPU core I-bus, D-bus and S-bus
  - DMA1 memory bus
  - DMA2 memory bus
  - DMA2 peripheral bus
  - USB OTG HS DMA bus
- Seven slaves:
  - Internal Flash memory ICode bus
  - Internal Flash memory DCode bus
  - Main internal SRAM1 (112 KB)
  - Auxiliary internal SRAM2 (16 KB)
  - AHB1 peripherals including AHB to APB bridges and APB peripherals
  - AHB2 peripherals
  - FMC / QUADSPI

The bus matrix provides access from a master to a slave, enabling concurrent access and efficient operation even when several high-speed peripherals work simultaneously. This architecture is shown in [Figure 1](#).

Figure 1. System architecture for STM32F446xx devices



## 2.1.1 I-bus

This bus connects the Instruction bus of the Cortex®-M4 with FPU core to the BusMatrix. This bus is used by the core to fetch instructions. The target of this bus is a memory containing code (internal Flash memory/SRAM or external memories through the FMC).

## 2.1.2 D-bus

This bus connects the databus of the Cortex®-M4 with FPU to the BusMatrix. This bus is used by the core for literal load and debug access. The target of this bus is a memory containing code or data (internal Flash memory or external memories through the FMC).

## 2.1.3 S-bus

This bus connects the system bus of the Cortex®-M4 with FPU core to a BusMatrix. This bus is used to access data located in a peripheral or in SRAM. Instructions may also be fetched on this bus (less efficient than ICode). The targets of this bus are the internal SRAM, SRAM2, the AHB1 peripherals including the APB peripherals, the AHB2 peripherals and the external memories through the FMC and QUADSPI.

## 2.1.4 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 Flash, internal SRAMs (SRAM1, SRAM2) and external memories through the FMC and QUADSPI.

### 2.1.5 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 Flash, internal SRAMs (SRAM1, SRAM2) and external memories through the FMC and the QUADSPI.

### 2.1.6 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 SRAMs (SRAM1, SRAM2), internal Flash memory, and external memories through the FMC and QUADSPI.

### 2.1.7 BusMatrix

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

### 2.1.8 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 and Flash memory interface). Before using a peripheral you have 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.

## 2.2 Memory organization

### 2.2.1 Introduction

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

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

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

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 [Memory map and register boundary addresses](#) and peripheral sections.

### 2.2.2 Memory map and register boundary addresses

See the datasheet corresponding to your device for a comprehensive diagram of the memory map.

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

**Table 1. STM32F446xx register boundary addresses**

Boundary address	Peripheral	Bus	Register map
0xA000 0000 - 0xA000 0FFF	FMC control register	AHB3	<a href="#">Section 11.8: FMC register map on page 322</a>
0xA000 1000 - 0xA000 1FFF	QUADSPI register	AHB3	<a href="#">Section 12.5.14: QUADSPI register map on page 352</a>
0x5005 0000 - 0x5005 03FF	DCMI	AHB2	<a href="#">Section 15.7.12: DCMI register map on page 447</a>
0x5000 0000 - 0x5003 FFFF	USB OTG FS	AHB2	<a href="#">Section 31.15.60: OTG_FS/OTG_HS register map on page 1190</a>
0x4004 0000 - 0x4007 FFFF	USB OTG HS	AHB1	<a href="#">Section 31.15.60: OTG_FS/OTG_HS register map on page 1190</a>
0x4002 6400 - 0x4002 67FF	DMA2		<a href="#">Section 9.5.11: DMA register map on page 233</a>
0x4002 6000 - 0x4002 63FF	DMA1		
0x4002 4000 - 0x4002 4FFF	BKPSRAM		<a href="#">Section 13.13.18: ADC register map on page 397</a>
0x4002 3C00 - 0x4002 3FFF	Flash interface register		<a href="#">Section 3.8: Flash interface registers on page 79</a>
0x4002 3800 - 0x4002 3BFF	RCC		<a href="#">Section 6.3.28: RCC register map on page 171</a>
0x4002 3000 - 0x4002 33FF	CRC		<a href="#">Section 4.4.4: CRC register map on page 90</a>

**Table 1. STM32F446xx register boundary addresses (continued)**

<b>Boundary address</b>	<b>Peripheral</b>	<b>Bus</b>	<b>Register map</b>
0x4002 1C00 - 0x4002 1FFF	GPIOH	AHB1	<a href="#">Section 7.4.11: GPIO register map on page 192</a>
0x4002 1800 - 0x4002 1BFF	GPIOG		
0x4002 1400 - 0x4002 17FF	GPIOF		
0x4002 1000 - 0x4002 13FF	GPIOE		
0x4002 0C00 - 0x4002 0FFF	GPIOD		
0x4002 0800 - 0x4002 0BFF	GPIOC		
0x4002 0400 - 0x4002 07FF	GPIOB		
0x4002 0000 - 0x4002 03FF	GPIOA		
0x4001 5C00 - 0x4001 5FFF	SAI2	APB2	<a href="#">Section 28.5.10: SAI register map on page 977</a>
0x4001 5800 - 0x4001 5BFF	SAI1		
0x4001 4800 - 0x4001 4BFF	TIM11	APB2	<a href="#">Section 18.5.12: TIM10/11/13/14 register map on page 626</a>
0x4001 4400 - 0x4001 47FF	TIM10		
0x4001 4000 - 0x4001 43FF	TIM9		
0x4001 3C00 - 0x4001 3FFF	EXTI		
0x4001 3800 - 0x4001 3BFF	SYSCFG		
0x4001 3400 - 0x4001 37FF	SPI4	APB2	<a href="#">Section 26.7.10: SPI register map on page 896</a>
0x4001 3000 - 0x4001 33FF	SPI1	APB2	<a href="#">Section 26.7.10: SPI register map on page 896</a>
0x4001 2C00 - 0x4001 2FFF	SDMMC		
0x4001 2000 - 0x4001 23FF	ADC1 - ADC2 - ADC3		
0x4001 1400 - 0x4001 17FF	USART6		
0x4001 1000 - 0x4001 13FF	USART1		
0x4001 0400 - 0x4001 07FF	TIM8		<a href="#">Section 16.4.21: TIM1&amp;TIM8 register map on page 518</a>
0x4001 0000 - 0x4001 03FF	TIM1		

**Table 1. STM32F446xx register boundary addresses (continued)**

<b>Boundary address</b>	<b>Peripheral</b>	<b>Bus</b>	<b>Register map</b>
0x4000 7400 - 0x4000 77FF	DAC	APB1	<a href="#">Section 14.5.15: DAC register map on page 421</a>
0x4000 7000 - 0x4000 73FF	PWR		<a href="#">Section 5.5: PWR register map on page 114</a>
0x4000 6C00 - 0x4000 6FFF	HDMI-CEC		<a href="#">Section 32.7.7: HDMI-CEC register map on page 1284</a>
0x4000 6800 - 0x4000 6BFF	CAN2		<a href="#">Section 30.9.5: bxCAN register map on page 1078</a>
0x4000 6400 - 0x4000 67FF	CAN1		
0x4000 5C00 - 0x4000 5FFF	I2C3		
0x4000 5800 - 0x4000 5BFF	I2C2		<a href="#">Section 24.6.11: I2C register map on page 793</a>
0x4000 5400 - 0x4000 57FF	I2C1		
0x4000 5000 - 0x4000 53FF	UART5		
0x4000 4C00 - 0x4000 4FFF	UART4		<a href="#">Section 25.6.8: USART register map on page 845</a>
0x4000 4800 - 0x4000 4BFF	USART3		
0x4000 4400 - 0x4000 47FF	USART2		
0x4000 4000 - 0x4000 43FF	SPDIF-RX		<a href="#">Section 27.5.13: SPDIFRX interface register map on page 934</a>
0x4000 3C00 - 0x4000 3FFF	SPI3 / I2S3		<a href="#">Section 26.7.10: SPI register map on page 896</a>
0x4000 3800 - 0x4000 3BFF	SPI2 / I2S2		
0x4000 3000 - 0x4000 33FF	IWDG		<a href="#">Section 20.4.5: IWDG register map on page 645</a>
0x4000 2C00 - 0x4000 2FFF	WWDG		<a href="#">Section 21.6.4: WWDG register map on page 652</a>
0x4000 2800 - 0x4000 2BFF	RTC & BKP Registers		<a href="#">Section 22.6.21: RTC register map on page 690</a>
0x4000 2000 - 0x4000 23FF	TIM14		
0x4000 1C00 - 0x4000 1FFF	TIM13		<a href="#">Section 18.5.12: TIM10/11/13/14 register map on page 626</a>
0x4000 1800 - 0x4000 1BFF	TIM12		
0x4000 1400 - 0x4000 17FF	TIM7		<a href="#">Section 19.4.9: TIM6&amp;TIM7 register map on page 639</a>
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		<a href="#">Section 17.4.21: TIMx register map on page 579</a>

### 2.2.3 Embedded SRAM

The STM32F446xx feature 4 Kbytes of backup SRAM (see [Section 5.1.2: Battery backup domain](#)) plus 128 Kbytes of system SRAM.

The embedded SRAM can be accessed as bytes, half-words (16 bits) or full words (32 bits). Read and write operations are performed at CPU speed with 0 wait state. The embedded SRAM is divided into up to two blocks:

- SRAM1 and SRAM2 mapped at address 0x2000 0000 and accessible by all AHB masters.

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.

The CPU can access the SRAM1 and SRAM2 through the System Bus or through the I-Code/D-Code buses when boot from SRAM is selected or when physical remap is selected ([Section 8.2.1: SYSCFG memory remap register \(SYSCFG\\_MEMRMP\)](#) in the SYSCFG controller). To get the max performance on SRAM execution, physical remap should be selected (boot or software selection).

### 2.2.4 Flash memory overview

The Flash memory interface manages CPU AHB I-Code and D-Code 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 a system of instruction prefetch and cache lines.

The Flash memory is organized as follows:

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

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

### 2.2.5 Bit banding

The Cortex®-M4 with FPU memory map includes two bit-band regions. These regions map each word in an alias region of memory to a bit in a bit-band region of memory. Writing to a word in the alias region has the same effect as a read-modify-write operation on the targeted bit in the bit-band region.

In the STM32F446xx devices both the peripheral registers and the SRAM are mapped to a bit-band region, so that single bit-band write and read operations are allowed. The operations are only available for Cortex®-M4 with FPU accesses, and not from other bus masters (e.g. DMA).

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

$$\text{bit\_word\_addr} = \text{bit\_band\_base} + (\text{byte\_offset} \times 32) + (\text{bit\_number} \times 4)$$

where:

- *bit\_word\_addr* is the address of the word in the alias memory region that maps to the targeted bit
- *bit\_band\_base* is the starting address of the alias region
- *byte\_offset* is the number of the byte in the bit-band region that contains the targeted bit
- *bit\_number* is the bit position (0-7) of the targeted bit

### Example

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

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

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

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

For more information on bit-banding, refer to the *Cortex®-M4 with FPU programming manual* (see [Related documents on page 1](#)).

## 2.3 Boot configuration

Due to its fixed memory map, the code area starts from address 0x0000 0000 (accessed through the ICode/DCode buses) while the data area (SRAM) starts from address 0x2000 0000 (accessed through the system bus). The Cortex®-M4 with FPU CPU always fetches the reset vector on the ICode bus, which implies to have the boot area available only in the code area (typically, Flash memory). STM32F446xx microcontrollers implement a special mechanism to be able to boot from other memories (like the internal SRAM).

In the STM32F446xx, three different boot modes can be selected through the BOOT[1:0] pins as shown in [Table 2](#).

**Table 2. Boot modes**

Boot mode selection pins		Boot mode	Aliasing
BOOT1	BOOT0		
x	0	Main Flash memory	Main Flash memory is selected as the boot area
0	1	System memory	System memory is selected as the boot area
1	1	Embedded SRAM	Embedded SRAM is selected as the boot area

The values on the BOOT pins are latched on the 4th rising edge of SYSCLK after a reset. It is up to the user to set the BOOT1 and BOOT0 pins after reset to select the required boot mode.

BOOT0 is a dedicated pin while BOOT1 is shared with a GPIO pin. Once BOOT1 has been sampled, the corresponding GPIO pin is free and can be used for other purposes.

The BOOT pins are 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 this startup delay is over, the CPU fetches the top-of-stack value from address 0x0000 0000, then starts code execution from the boot memory starting from 0x0000 0004.

**Note:** *When the device boots from SRAM, in the application initialization code, you have to relocate the vector table in SRAM using the NVIC exception table and the offset register.*

### Embedded bootloader

The embedded bootloader mode is used to reprogram the Flash memory using one of the following serial interfaces:

- USART
- CAN2
- I2C
- SPI
- USB OTG FS in Device mode (DFU: device firmware upgrade).

The USART peripherals operate at the internal 16 MHz oscillator (HSI) frequency, while the CAN and USB OTG FS require an external clock (HSE) multiple of 1 MHz (ranging from 4 to 26 MHz).

The embedded bootloader code is located in system memory. It is programmed by ST during production. For additional information, refer to application note AN2606.

### Physical remap in STM32F446xx

Once the boot pins are selected, the application software can modify the memory accessible in the code area (in this way the code can be executed through the ICode bus in place of the System bus). This modification is performed by programming the [Section 8.2.1: SYSCFG memory remap register \(SYSCFG\\_MEMRMP\)](#) in the SYSCFG controller.

The following memories can thus be remapped:

- Main Flash memory
- System memory
- Embedded SRAM1 (112 KB)
- FMC bank 1 (NOR/PSRAM 1 and 2)
- FMC SDRAM bank 1

**Table 3. Memory mapping vs. Boot mode/physical remap in STM32F446xx**

Addresses	Boot/Remap in main Flash memory	Boot/Remap in embedded SRAM	Boot/Remap in System memory	Remap in FMC
0x2001 C000 - 0x2001 FFFF	SRAM2 (16 KB)	SRAM2 (16 KB)	SRAM2 (16 KB)	SRAM2 (16 KB)
0x2000 0000 - 0x2001 BFFF	SRAM1 (112 KB)	SRAM1 (112 KB)	SRAM1 (112 KB)	SRAM1 (112 KB)
0x1FFF 0000 - 0x1FFF 77FF	System memory	System memory	System memory	System memory
0x0810 0000 - 0x0FFF FFFF	Reserved	Reserved	Reserved	Reserved
0x0800 0000 - 0x081F FFFF	Flash memory	Flash memory	Flash memory	Flash memory

**Table 3. Memory mapping vs. Boot mode/physical remap in STM32F446xx (continued)**

Addresses	Boot/Remap in main Flash memory	Boot/Remap in embedded SRAM	Boot/Remap in System memory	Remap in FMC
0x0400 0000 - 0x07FF FFFF	Reserved	Reserved	Reserved	FMC bank 1 NOR/PSRAM 2 (128 MB Aliased)
0x0000 0000 - 0x001F FFFF <sup>(1)(2)</sup>	Flash (512 KB) Aliased	SRAM1 (112 KB) Aliased	System memory (30 KB) Aliased	FMC bank 1 NOR/PSRAM 1 (128 MB Aliased) or FMC SDRAM bank 1 (128 MB Aliased)

1. When the FMC is remapped at address 0x0000 0000, only the first two regions of bank 1 memory controller (bank 1 NOR/PSRAM 1 and NOR/PSRAM 2) or SDRAM bank 1 can be remapped. In remap mode, the CPU can access the external memory via ICode bus instead of System bus which boosts up the performance.
2. Even when aliased in the boot memory space, the related memory is still accessible at its original memory space.

## 3 Embedded Flash memory interface

### 3.1 Introduction

The Flash memory interface manages CPU AHB I-Code and D-Code accesses to the Flash memory. It implements the erase and program Flash memory operations and the read and write protection mechanisms.

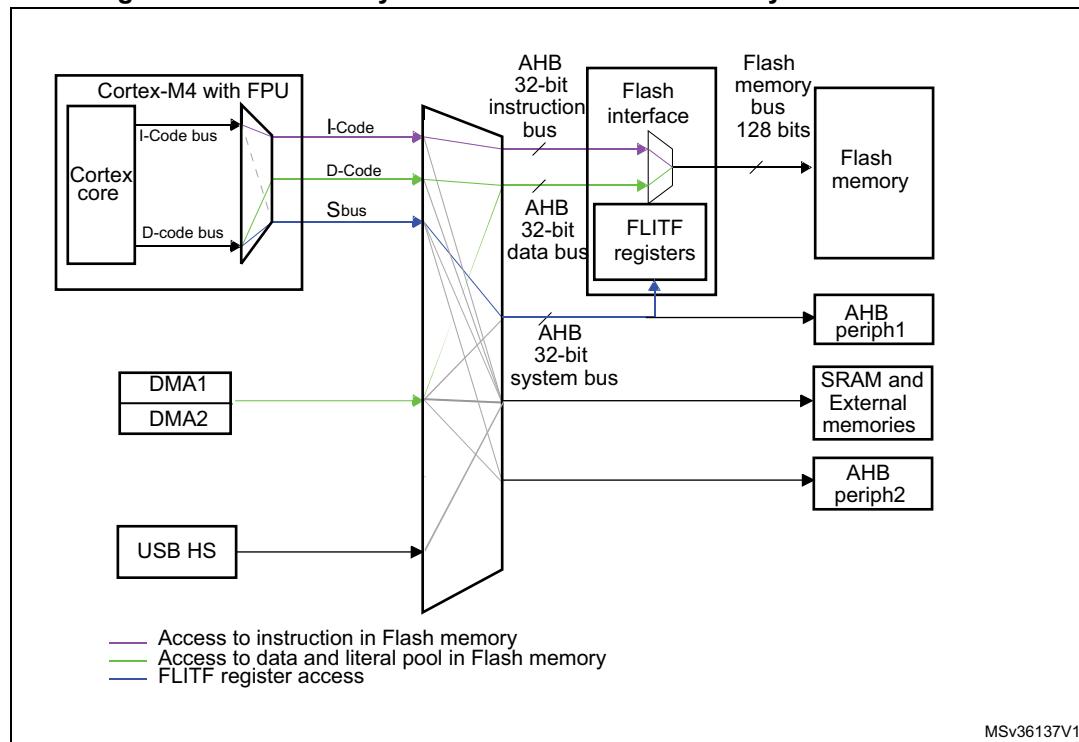
The Flash memory interface accelerates code execution with a system of instruction prefetch and cache lines.

### 3.2 Main features

- Flash memory read operations
- Flash memory program/erase operations
- Read / write protections
- Prefetch on I-Code
- 64 cache lines of 128 bits on I-Code
- 8 cache lines of 128 bits on D-Code

*Figure 2* shows the Flash memory interface connection inside the system architecture.

**Figure 2. Flash memory interface connection inside system architecture**



### 3.3 Embedded Flash memory

The Flash memory has the following main features:

- Capacity up to 512 KBytes
- 128 bits wide data read
- Byte, half-word, word and double word write
- Sector and mass erase
- Memory organization

The Flash memory is organized as follows:

- A main memory block divided into 4 sectors of 16 KBytes, 1 sector of 64 KBytes, and 3 sectors of 128 Kbytes
- System memory from which the device boots in System memory boot mode
- 512 OTP (one-time programmable) bytes 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, watchdog software/hardware and reset when the device is in Standby or Stop mode.
- Low-power modes (for details refer to the Power control (PWR) section of the reference manual)

**Table 4. Flash module organization**

Block	Name	Block base addresses	Size
Main memory	Sector 0	0x0800 0000 - 0x0800 3FFF	16 Kbytes
	Sector 1	0x0800 4000 - 0x0800 7FFF	16 Kbytes
	Sector 2	0x0800 8000 - 0x0800 BFFF	16 Kbytes
	Sector 3	0x0800 C000 - 0x0800 FFFF	16 Kbytes
	Sector 4	0x0801 0000 - 0x0801 FFFF	64 Kbytes
	Sector 5	0x0802 0000 - 0x0803 FFFF	128 Kbytes
	Sector 6	0x0804 0000 - 0x0805 FFFF	128 Kbytes
	Sector 7	0x0806 0000 - 0x0807 FFFF	128 Kbytes
System memory		0x1FFF 0000 - 0x1FFF 77FF	30 Kbytes
OTP area		0x1FFF 7800 - 0x1FFF 7A0F	528 bytes
Option bytes		0x1FFF C000 - 0x1FFF C00F	16 bytes

## 3.4 Read interface

### 3.4.1 Relation between CPU clock frequency and Flash memory read time

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 prefetch buffer must be disabled when the supply voltage is below 2.1 V. The correspondence between wait states and CPU clock frequency is given in [Table 5](#).

**Note:** On STM32F446xx devices:

- when  $VOS[1:0] = '0x01'$ , the maximum value of  $f_{HCLK}$  is 120 MHz.
- when  $VOS[1:0] = '0x10'$ , the maximum value of  $f_{HCLK}$  is 144 MHz. It can be extended to 168 MHz by activating the over-drive mode.
- when  $VOS[1:0] = '0x11'$ , the maximum value of  $f_{HCLK}$  is 168 MHz. It can be extended to 180 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 5.1.3: 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 Prefetch OFF
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)		$144 < HCLK \leq 168$	$132 < HCLK \leq 154$	$120 < HCLK \leq 140$
7 WS (8 CPU cycles)		$168 < HCLK \leq 180$	$154 < HCLK \leq 176$	$140 < HCLK \leq 160$
8 WS (9 CPU cycles)			$176 < HCLK \leq 180$	$160 < HCLK \leq 168$

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 needed 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 you have configured, you can check the AHB prescaler factor and clock source status values. To make sure that the number of WS you have programmed is effective, you can read the FLASH\_ACR register.*

### 3.4.2

### Adaptive real-time memory accelerator (ART Accelerator™)

The proprietary Adaptive real-time (ART) memory accelerator is optimized for STM32 industry-standard ARM® Cortex®-M4 with FPU processors. It balances the inherent performance advantage of the ARM® Cortex®-M4 with FPU over Flash memory technologies, which normally requires the processor to wait for the Flash memory at higher operating frequencies.

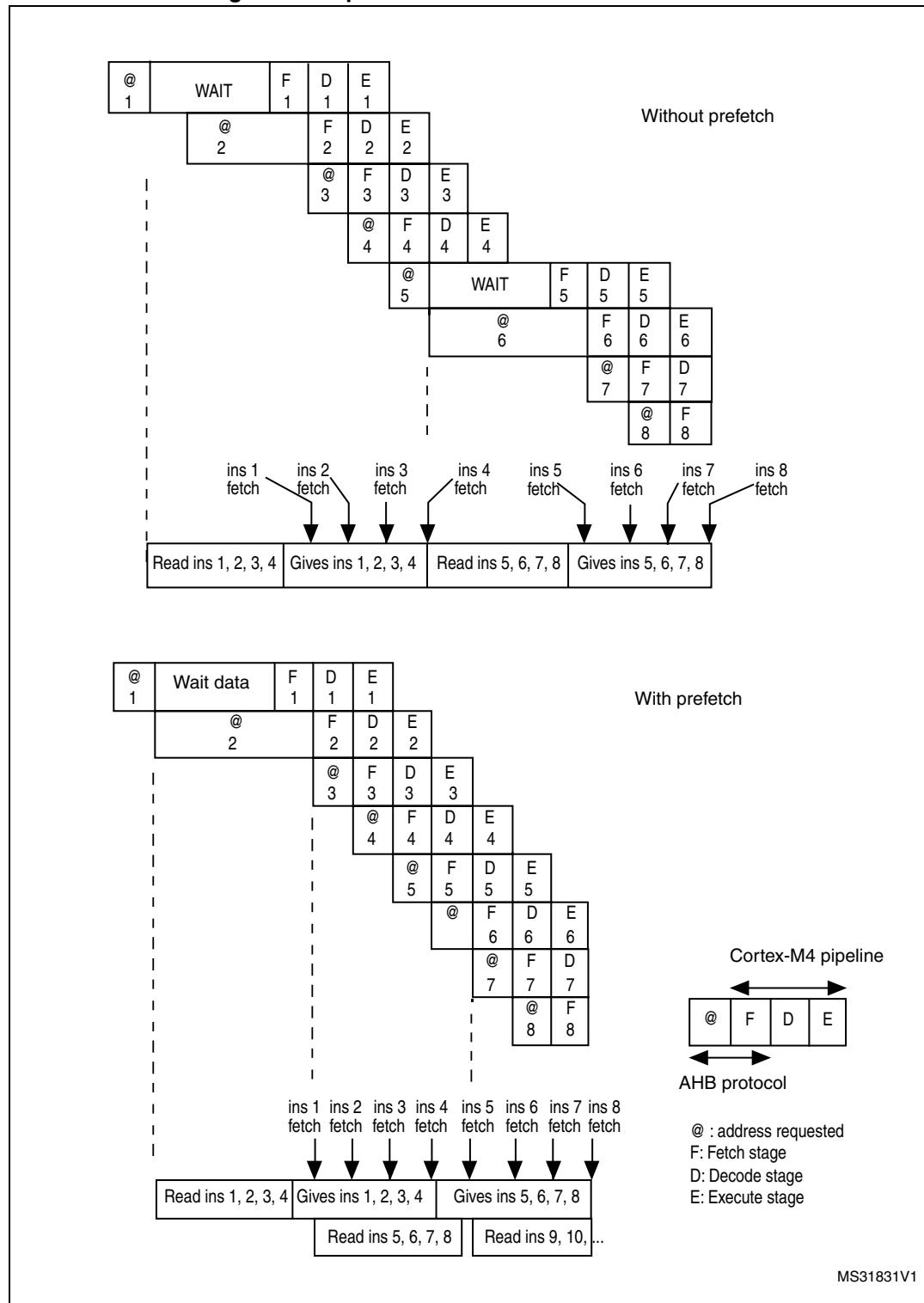
To release the processor full performance, the accelerator implements an instruction prefetch queue and branch cache which increases program execution speed from the 128-bit Flash memory. Based on CoreMark benchmark, the performance achieved thanks to the ART accelerator is equivalent to 0 wait state program execution from Flash memory at a CPU frequency up to 180 MHz.

#### Instruction prefetch

Each Flash memory read operation provides 128 bits from either four instructions of 32 bits or 8 instructions of 16 bits according to the program launched. So, in case of sequential code, at least four CPU cycles are needed to execute the previous read instruction line. Prefetch on the I-Code bus can be used to read the next sequential instruction line from the Flash memory while the current instruction line is being requested by the CPU. Prefetch is enabled by setting the PRFTEN bit in the FLASH\_ACR register. This feature is useful if at least one wait state is needed to access the Flash memory.

**Figure 3** shows the execution of sequential 32-bit instructions with and without prefetch when 3 WSs are needed to access the Flash memory.

**Figure 3. Sequential 32-bit instruction execution**



When the code is not sequential (branch), the instruction may not be present in the currently used instruction line or in the prefetched instruction line. In this case (miss), the penalty in terms of number of cycles is at least equal to the number of wait states.

### Instruction cache memory

To limit the time lost due to jumps, it is possible to retain 64 lines of 128 bits in an instruction cache memory. This feature can be enabled by setting the instruction cache enable (ICEN) bit in the FLASH\_ACR register. Each time a miss occurs (requested data not present in the currently used instruction line, in the prefetched instruction line or in the instruction cache memory), the line read is copied into the instruction cache memory. If some data contained in the instruction cache memory are requested by the CPU, they are provided without inserting any delay. Once all the instruction cache memory lines have been filled, the LRU (least recently used) policy is used to determine the line to replace in the instruction memory cache. This feature is particularly useful in case of code containing loops.

### Data management

Literal pools are fetched from Flash memory through the D-Code bus during the execution stage of the CPU pipeline. The CPU pipeline is consequently stalled until the requested literal pool is provided. To limit the time lost due to literal pools, accesses through the AHB databus D-Code have priority over accesses through the AHB instruction bus I-Code.

If some literal pools are frequently used, the data cache memory can be enabled by setting the data cache enable (DCEN) bit in the FLASH\_ACR register. This feature works like the instruction cache memory, but the retained data size is limited to 8 rows of 128 bits.

*Note:* *Data in user configuration sector are not cacheable.*

## 3.5 Erase and program 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 on STM32F4xx 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.

### 3.5.1 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.

### 3.5.2 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<sub>PP</sub> 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 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 <sub>PP</sub>	Voltage range 2.7 - 3.6 V	Voltage range 2.4 - 2.7 V	Voltage range 2.1 - 2.4 V	Voltage range 1.7 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<sub>PP</sub> an external high-voltage supply (between 8 and 9 V) must be applied to the V<sub>PP</sub> 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<sub>PP</sub> to initial programming on the factory line. The V<sub>PP</sub> supply must not be applied for more than an hour, otherwise the Flash memory might be damaged.

### 3.5.3 Erase

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 7 sectors in the main memory block you wish 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 MER and SER bits are both set in the FLASH\_CR register, mass erase is performed.*

*If both MER 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.*

### 3.5.4 Programming

#### 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.*

#### Programming errors

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 a program alignment error flag (PGAERR) is set in the FLASH\_SR register.

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 a program parallelism error flag (PGPERR) is set in the FLASH\_SR register.

If the standard programming sequence is not respected (for example, if there is an attempt to write to a Flash memory address when the PG bit is not set), the operation is aborted and a program sequence error flag (PGSERR) is set in the FLASH\_SR register.

#### Programming and caches

If a Flash memory write access concerns some data in the data cache, the Flash write access modifies the data in the Flash memory and the data in the cache.

If an erase operation in Flash memory also concerns data in the data or instruction cache, you have 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 the caches by setting the DCRST and ICRST bits in the FLASH\_CR register.

*Note:* *The I/D cache should be flushed only when it is disabled (I/DCEN = 0).*

### 3.5.5 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, PGSERR (Program error flags)
- WRPERR (Protection error flag)

In this case, if the error interrupt enable bit (ERRIE) is set in the FLASH\_SR 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, PGSERR	ERRIE

## 3.6 Option bytes

### 3.6.1 Description of user option bytes

The option bytes are configured by the end user depending on the application requirements. *Table 8* shows the organization of these bytes inside the user configuration sector.

**Table 8. Option byte organization**

Address	[63:16]	[15:0]
0x1FFF C000	Reserved	ROP & user option bytes (RDP & USER)
0x1FFF C008	Reserved	Write protection nWRP bits for sectors 0 to 7

**Table 9. Description of the option bytes**

<b>Option bytes (word, address 0x1FFF C000)</b>	
<b>RDP:</b> Read protection option byte.	
The read protection is used to protect the software code stored in Flash memory.	
Bits 15:8	0xAA: Level 0, no protection 0xCC: Level 2, chip protection (debug and boot from RAM features disabled) Others: Level 1, read protection of memories (debug features limited)
<b>USER:</b> User option byte	
This byte is used to configure the following features: Select the watchdog event: Hardware or software Reset event when entering the Stop mode Reset event when entering the Standby mode	
Bit 7	<b>nRST_STDBY</b> 0: Reset generated when entering the Standby mode 1: No reset generated
Bit 6	<b>nRST_STOP</b> 0: Reset generated when entering the Stop mode 1: No reset generated
Bit 5	<b>WDG_SW</b> 0: Hardware watchdog 1: Software watchdog
Bit 4	0x0: Not used. Do not change the default value of this bit which is “0” out of fab.
Bits 3:2	<b>BOR_LEV:</b> 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 For full details on BOR characteristics, refer to the Electrical characteristics section of the product datasheet.
Bits 1:0	0x1: Not used
<b>Option bytes (word, address 0x1FFF C008)</b>	
Bit 15	<b>SPRMOD:</b> Selection of Protection Mode of nWPRI bits 0: nWPRI bits used for sector i write protection (Default) 1: nWPRI bits used for sector i PCROP protection (Sector)
Bits 14:8	Reserved

**Table 9. Description of the option bytes (continued)**

<b>nWRP:</b> <i>Flash memory write protection option bytes</i> Sectors 0 to 7 can be write protected.	
Bits 7:0	<b>nWRPi</b> If SPRMOD is reset (default value) : 0: Write protection active on sector i. 1: Write protection not active on sector i. If SPRMOD is set (active): 0: PCROP protection not active on sector i. 1: PCROP protection active on sector i.

### 3.6.2 Programming user option bytes

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, you have 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 user configuration sector and then programming all the option bytes with the values contained in the FLASH\_OPTCR register.*

### 3.6.3 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 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 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 from user code are allowed.

When Level 1 is active, programming the protection option byte (RDP) to Level 0 causes the Flash memory 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. The OTP area is not affected by mass erase and remains unchanged. 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 protection 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 or system memory bootloader is no more allowed.
- JTAG, SWV (single-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 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.*

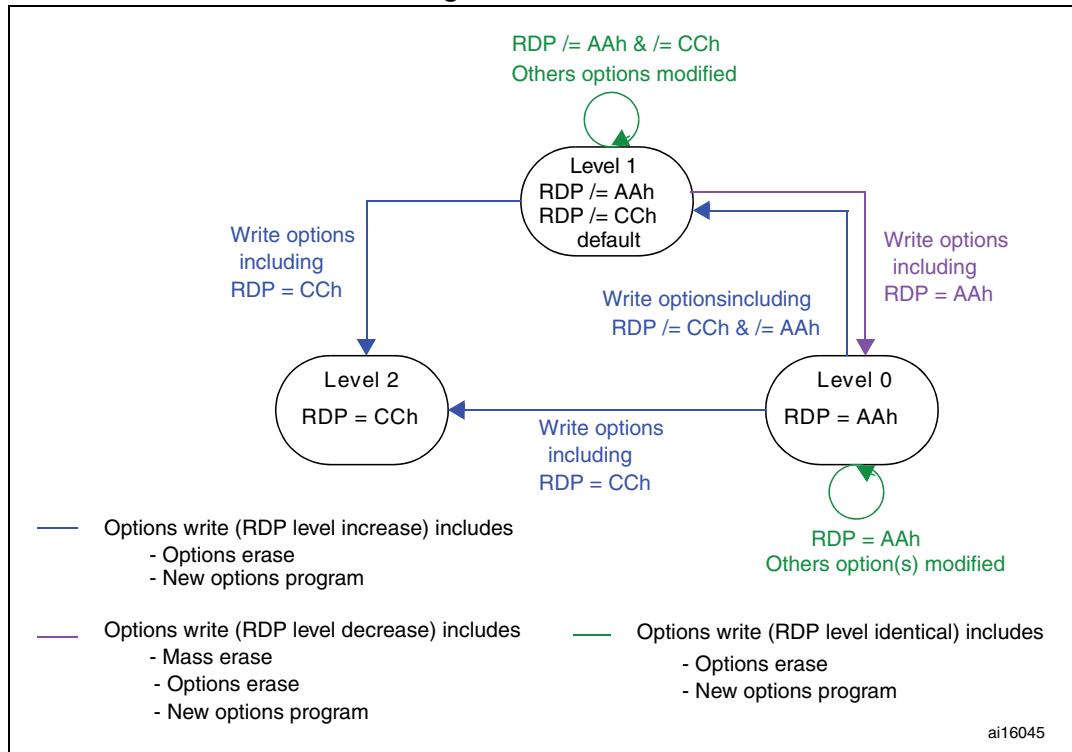
**Table 10. Access versus read protection level**

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	Level 1	NO		NO <sup>(1)</sup>	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 is 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**



### 3.6.4 Write protections

Up to 7 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:

- Mass, bank, sector erase are configured (MER and SER = 1)
- 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 = 1 and nWRPi = 0 with  $0 \leq i \leq 117$  bits in the FLASH\_OPTCRx register)
- A sector erase is requested on a write protected sector. (SER = 1, SNB = i and nWRPi = 0 with  $0 \leq i \leq 117$  bits in the FLASH\_OPTCRx 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 user configuration sector
- 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.

### 3.6.5 Proprietary code readout protection (PCROP)

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

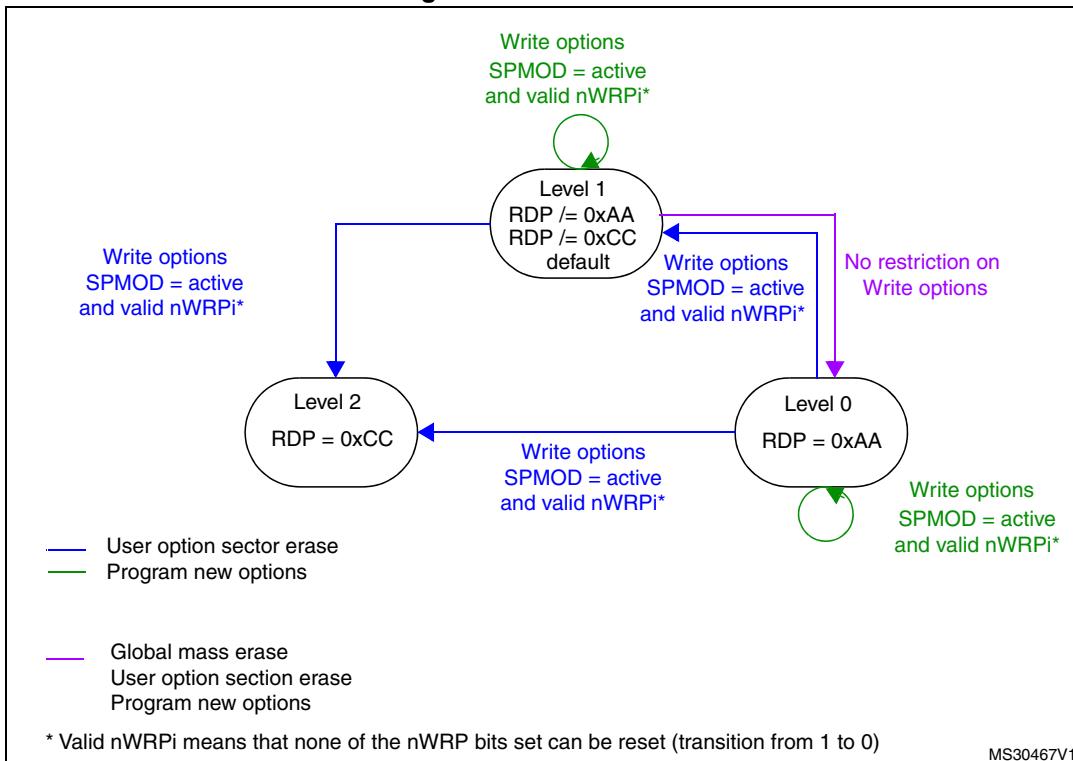
The PCROP protection is selected as follows, through the SPRMOD option bit in the FLASH\_CR register:

- SPRMOD = 0: nWRPi control the write protection of respective user sectors
- SPRMOD = 1: nWRPi control the read and write protection (PCROP) of respective user sectors.

When a sector is readout protected (PCROP mode activated), it can only be accessed for code fetch through ICODE Bus on Flash interface:

- Any read access performed through the D-bus triggers a RDERR flag error.
- Any program/erase operation on a PCROPed sector triggers a WRPERR flag error.

Figure 5. PCROP levels



The deactivation of the SPRMOD and/or the unprotection of PCROPed user sectors can only occur when, at the same time, the RDP level changes from 1 to 0. If this condition is not respected, the user option byte modification is canceled and the write error WRPERR flag is set. The modification of the users option bytes (BOR\_lev, RST\_STDBY, ..) is allowed since none of the active nWRPi bits is reset and SPRMOD is kept active.

Note:

The active value of nWRPi bits is inverted when PCROP mode is active (SPRMOD = 1).

If SPRMOD = 1 and nWRPi = 1, then user sector i of bank 1, respectively bank 2 is read/write protected (PCROP).

### 3.7 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

Block	[128:96]	[95:64]	[63:32]	[31:0]	Address byte 0
0	OTP0	OTP0	OTP0	OTP0	0x1FFF 7800
	OTP0	OTP0	OTP0	OTP0	0x1FFF 7810
1	OTP1	OTP1	OTP1	OTP1	0x1FFF 7820
	OTP1	OTP1	OTP1	OTP1	0x1FFF 7830
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.

**Table 11. OTP area organization (continued)**

Block	[128:96]	[95:64]	[63:32]	[31:0]	Address byte 0
15	OTP15	OTP15	OTP15	OTP15	0x1FFF 79E0
	OTP15	OTP15	OTP15	OTP15	0x1FFF 79F0
Lock block	LOCKB15 ... LOCKB12	LOCKB11 ... LOCKB8	LOCKB7 ... LOCKB4	LOCKB3 ... LOCKB0	0x1FFF 7A00

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.

## 3.8 Flash interface registers

### 3.8.1 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.	DCRST	ICRST	DCEN	ICEN	PRFTEN	Res.	Res.	Res.	Res.			LATENCY	
			rw	w	rw	rw	rw					rw	rw	rw	rw

Bits 31:13 Reserved, must be kept cleared.

Bit 12 **DCRST:** Data cache reset

0: Data cache is not reset

1: Data cache is reset

This bit can be written only when the D cache is disabled.

Bit 11 **ICRST:** Instruction cache reset

0: Instruction cache is not reset

1: Instruction cache is reset

This bit can be written only when the I cache is disabled.

Bit 10 **DCEN:** Data cache enable

0: Data cache is disabled

1: Data cache is enabled

Bit 9 **ICEN**: Instruction cache enable  
 0: Instruction cache is disabled  
 1: Instruction cache 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**: 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

### 3.8.2 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**: 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

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

The Flash option key register is used to allow program and erase operations in the user configuration sector.

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**: 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

### 3.8.4 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.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	BSY							
															r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	RDERR	PGSERR	PGPERR	PGAERR	WRPERR	Res.	Res.	OPERR	EOP						
							rw	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**: Read Protection Error (pcrop)

Set by hardware when an address to be read through the Dbus belongs to a read protected part of the flash.

Reset by writing 1.

Bit 7 **PGSERR**: Programming 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.

### 3.8.5 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.	Res.	ERRIE	EOPIE	Res.	Res.	Res.	Res.	Res.	Res.	Res.	STRT
rs						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.	SNB[3:0]				MER	SER	PG
						rw	rw		rw	rw	rw	rw	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 30:26 Reserved, must be kept cleared.

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:** Sector number

These bits select the sector to erase.

0000 sector 0

0001 sector 1

...

0101 sector 5

0110 sector 6

0111 sector 7

1000 not allowed

...

1011 not allowed

1100 user specific sector

1101 user configuration sector

1110 not allowed

1111 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.

### 3.8.6 Flash option control register (FLASH\_OPTCR)

The FLASH\_OPTCR register is used to modify the user option bytes.

Address offset: 0x14

Reset value: 0xFFFF AAED. The option bits 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
SPRMOD	Res.	nWRP[7:0]													
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	WDG_SW	Res.	BOR_LEV		OPTST_RT	OPTLO_CK
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rs	rs

Bit 31 **SPRMOD:** Selection of Protection Mode of nWPRI bits

0: PCROP disabled, nWPRI bits used for Write Protection on sector i

1: PCROP enabled, nWPRI bits used for PCROP Protection on sector i

Bits 30:24 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 of sectors after reset. They can be written to program a new write protect value into Flash memory.

0: Write protection active on selected sector

1: Write protection not active on selected sector

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

If SPRMOD is reset:

0: Write protection active on sector i

1: Write protection not active on sector i

If SPRMOD is set:

0: PCROP protection not active on sector i

1: PCROP protection active on sector i

Bits 15:8 **RDP:** 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 7:5 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: WDG\_SW

*Note: When changing the WDG mode from hardware to software or from software to hardware, a system reset is required to make the change effective.*

Bit 4 Reserved, must be kept cleared. Always read as “0”.

**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. 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 about BOR characteristics, refer to the “Electrical characteristics” section in the device 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.

### 3.8.7 Flash interface register map

Table 12. Flash register map and reset value

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	<b>FLASH_ACR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	LATENCY	0	
	Reset value																																
0x04	<b>FLASH_KEYR</b>	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	<b>FLASH_OPTKEYR</b>	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	<b>FLASH_SR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value																																
0x10	<b>FLASH_CR</b>	1	LOCK	Res.	Res.	Res.	Res.	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	SPRMOD	0	ERRIE	0	EOPIE	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
0x14	<b>FLASH_OPTCR</b>	nWRP[7:0]										RDP[7:0]																					
	Reset value	0	PRFEN	0	RDERR	0	PGSERR	0	PGPERR	0	PGAERR	0	WRPERR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

Refer to [Table 1 on page 56](#) for the register boundary addresses.

## 4 CRC calculation unit

### 4.1 CRC introduction

The CRC (cyclic redundancy check) calculation unit is used to get a CRC code from a 32-bit data word and a fixed generator polynomial.

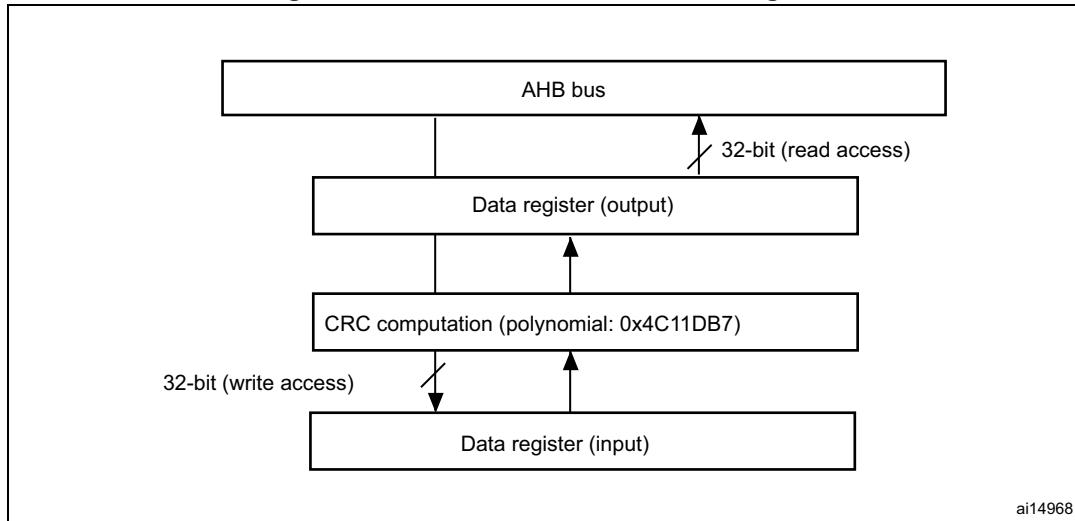
Among other applications, CRC-based techniques are used to verify data transmission or storage integrity. In the scope of the EN/IEC 60335-1 standard, they offer a way of verifying the Flash memory integrity. The CRC calculation unit helps compute a signature of the software during runtime, to be compared with a reference signature generated at link-time and stored at a given memory location.

### 4.2 CRC main features

- Uses CRC-32 (Ethernet) polynomial: 0x4C11DB7
  - $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$
- Single input/output 32-bit data register
- CRC computation done in four AHB clock cycles (HCLK)
- General-purpose 8-bit register (can be used for temporary storage)

The block diagram is shown in [Figure 6](#).

**Figure 6. CRC calculation unit block diagram**



### 4.3 CRC functional description

The CRC calculation unit mainly consists of a single 32-bit data register, which:

- is used as an input register to enter new data in the CRC calculator (when writing into the register)
- holds the result of the previous CRC calculation (when reading the register)

Each write operation into the data register creates a combination of the previous CRC value and the new one (CRC computation is done on the whole 32-bit data word, and not byte per byte).

The write operation is stalled until the end of the CRC computation, thus allowing back-to-back write accesses or consecutive write and read accesses.

The CRC calculator can be reset to 0xFFFF FFFF with the RESET control bit in the CRC\_CR register. This operation does not affect the contents of the CRC\_IDR register.

## 4.4 CRC registers

The CRC calculation unit contains two data registers and a control register. The peripheral The CRC registers have to be accessed by words (32 bits).

### 4.4.1 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 Data register bits

Used as an input register when writing new data into the CRC calculator.  
Holds the previous CRC calculation result when it is read.

#### 4.4.2 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.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	IDR[7:0]														

Bits 31:8 Reserved, must be kept at reset value.

##### Bits 7:0 General-purpose 8-bit data register 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.

#### 4.4.3 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.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	RESET														
															w

Bits 31:1 Reserved, must be kept at reset value.

##### Bit 0 RESET bit

Resets the CRC calculation unit and sets the data register to 0xFFFF FFFF.

This bit can only be set, it is automatically cleared by hardware.

#### 4.4.4 CRC register map

**Table 13. CRC calculation unit register map and reset values**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	<b>CRC_DR</b>																																
		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
0x04	<b>CRC_IDR</b>																																
		Reset value																															
0x08	<b>CRC_CR</b>																																
		Reset value																															

## 5 Power controller (PWR)

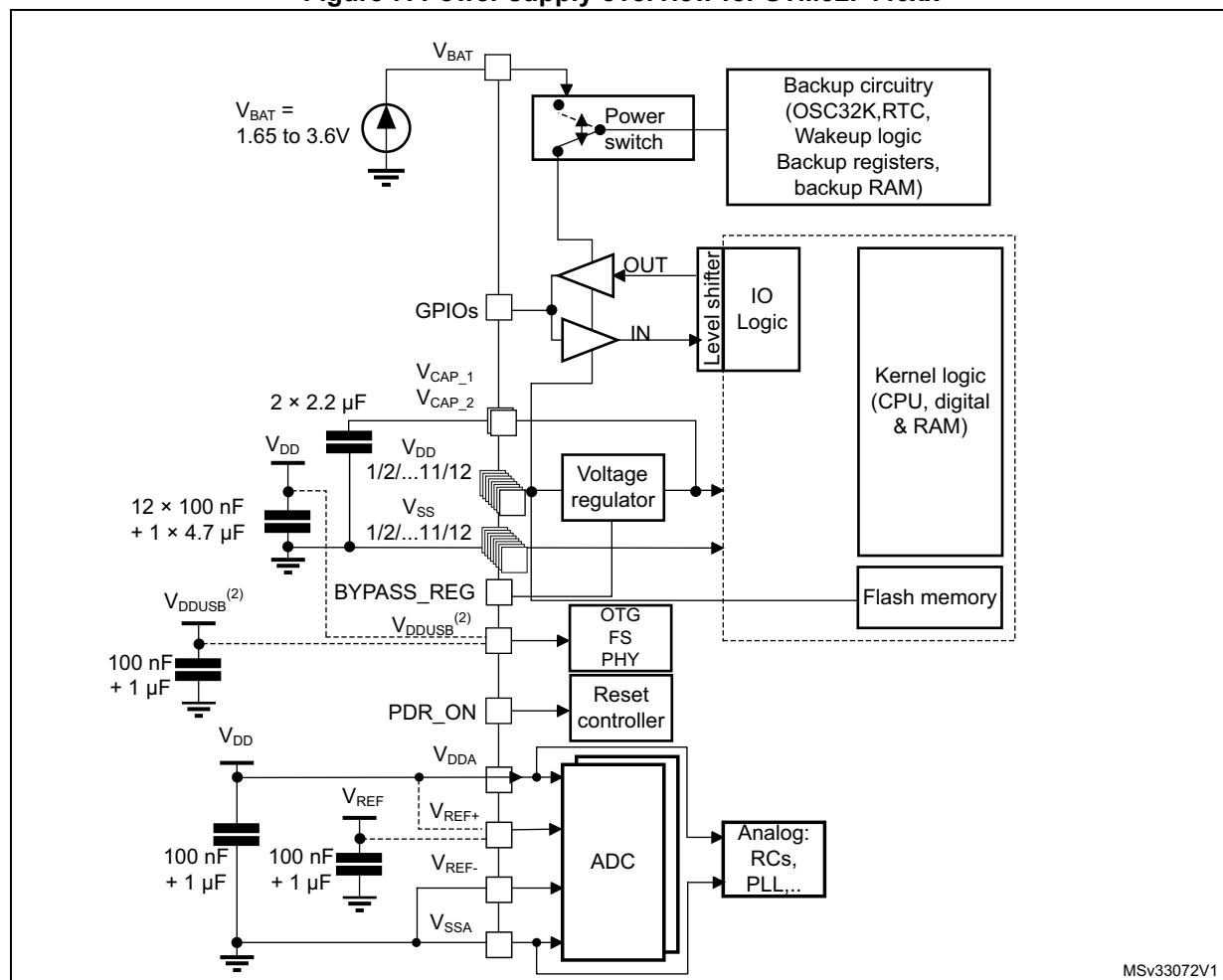
### 5.1 Power supplies

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 STM32F446xx datasheet.

Figure 7. Power supply overview for STM32F446xx



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

### 5.1.1 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  $V_{DDA}$ .

### 5.1.2 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

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 can be used as a GPIOas the RTC\_AF1 pin (refer to [Table 24: RTC\\_AF1 pin](#) for more details about this pin configuration)

**Note:** Due to the fact that the switch only sinks a limited amount of current (3 mA), the use of GPIOs PC13 to PC15 in output mode is restricted: the speed has to be limited to 2 MHz with a maximum load of 30 pF and these 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 the RTC\_AF1 pin (refer to [Table 24: RTC\\_AF1 pin](#) for more details about this pin configuration).

### 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 6.3.13: RCC APB1 peripheral clock enable register \(RCC\\_APB1ENR\)](#))
- 2. Set the DBP bit in the [Section 5.4.1: PWR power control register \(PWR\\_CR\)](#) and [PWR power control register \(PWR\\_CR\)](#) to enable access to the backup domain
- 3. Select the RTC clock source: see [Section 6.2.8: RTC/AWU clock](#)
- 4. Enable the RTC clock by programming the RTCEN [15] bit in the [Section 6.3.20: 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 6.3.13](#)).
- 2. Set the DBP bit in the [PWR power control register \(PWR\\_CR\)](#) to enable access to the backup domain
- 3. Enable the backup SRAM clock by setting BKPSRAMEN bit in the [RCC APB1 peripheral clock enable register \(RCC\\_APB1ENR\)](#).

### 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 20 backup data registers (80 bytes) which are reset when a tamper detection event occurs. For more details refer to .

### 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.

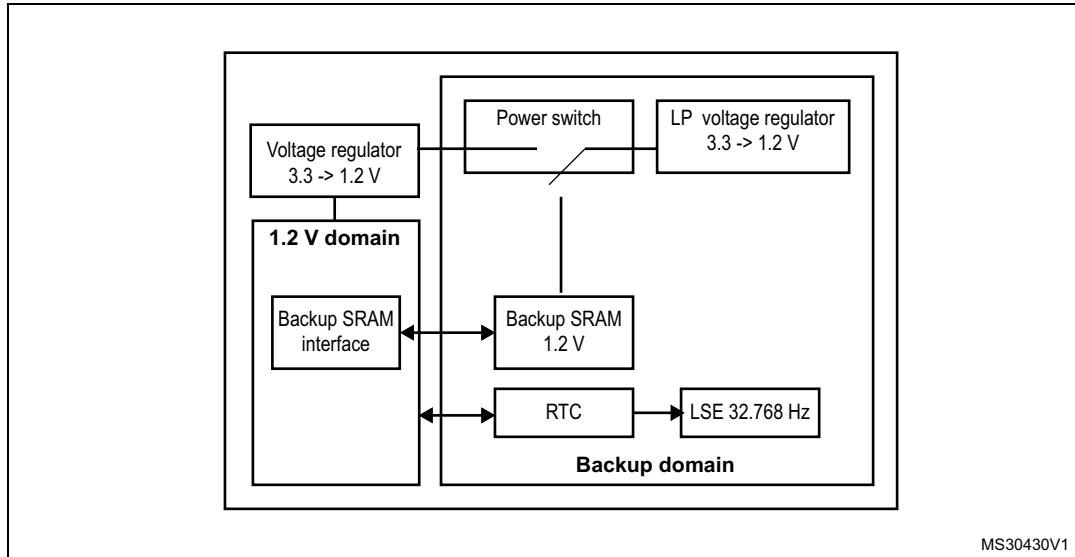
When the backup domain is supplied by  $V_{BAT}$  (analog switch connected to  $V_{BAT}$  because  $V_{DD}$  is not present), the backup SRAM is powered by a dedicated low-power regulator. This

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\_CSR register.

The backup SRAM is not mass erased by a tamper event.

When the Flash is read out protected, the backup SRAM is also read protected to prevent confidential data (such as cryptographic private key) from being accessed. When the protection level change from level 1 to level 0 is requested, the backup SRAM content is erased. Refer to the description of Read protection (RDP) option byte.

**Figure 8. Backup domain**



### 5.1.3 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\_CR 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 5.4.1: PWR power control register \(PWR\\_CR\)](#)).

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.
  - **Low voltage mode.**
  - **Under-drive mode:** the 1.2 V domain is preserved in reduced leakage mode. This mode is only available when the main regulator or the low-power regulator is in low voltage mode (see [Table 14](#)).
- 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 STM32F446xx datasheet.*

**Table 14. Voltage regulator configuration mode versus device operating mode<sup>(1)</sup>**

Voltage regulator configuration	Run mode	Sleep mode	Stop mode	Standby mode
Normal mode	MR	MR	MR or LPR	-
Low-voltage mode	-	-	MR or LPR	-
Over-drive mode <sup>(2)</sup>	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\_CR register to enable the Over-drive mode and wait for the ODRDY flag to be set in the PWR\_CSR register.
4. Set the ODSW bit in the PWR\_CR 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\_CSR 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 and SAI2 clocks, 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\_CR 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 and SAI2 clocks, USB\_48MHz clock,...)
3. Reset simultaneously the ODEN and the ODSW bits in the PWR\_CR register to switch back the voltage regulator to Normal mode and disable the Over-drive mode.
4. Wait for the ODWRDY flag of PWR\_CSR 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 and SAI2 clocks, USB\_48MHz clock,...).
3. Reset the ODSW bit in the PWR\_CR 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\_CSR to be reset.
5. Reset the ODEN bit in the PWR\_CR 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.*

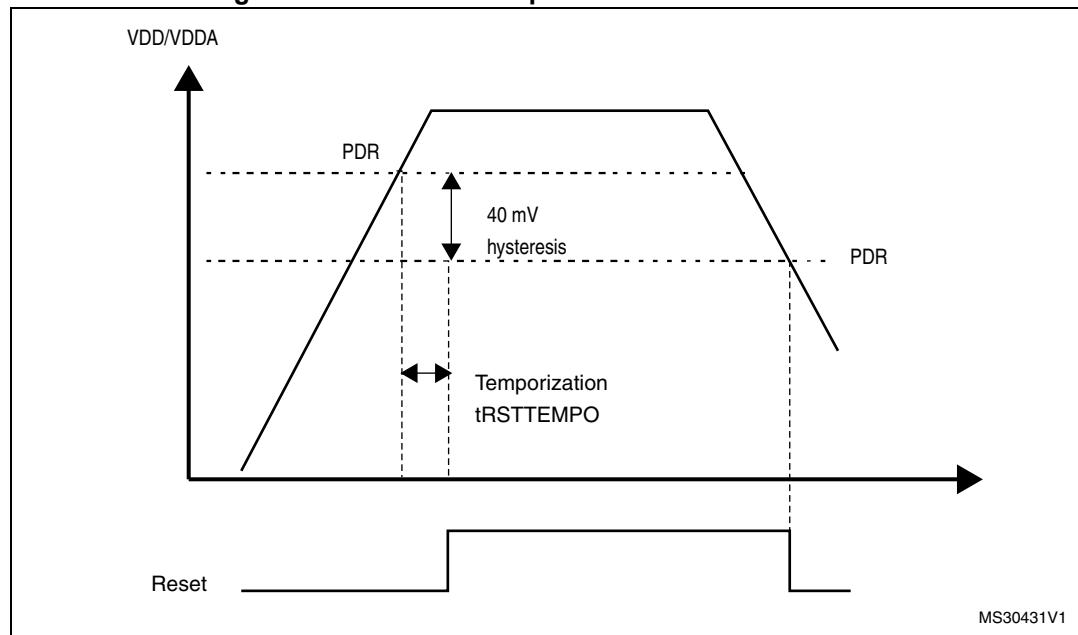
## 5.2 Power supply supervisor

### 5.2.1 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 9. Power-on reset/power-down reset waveform**



## 5.2.2 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.

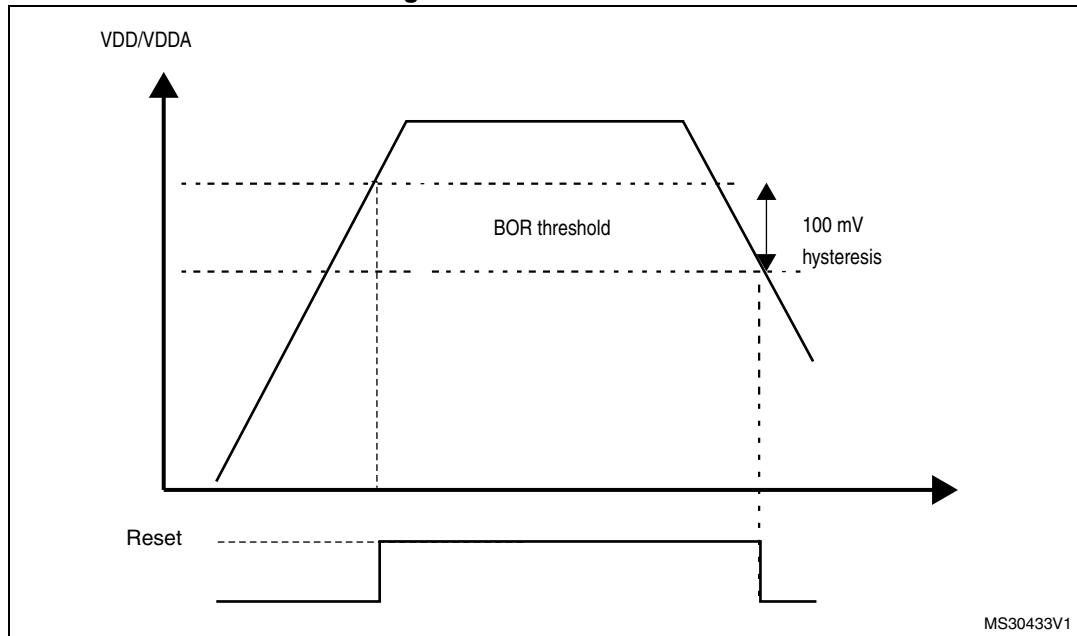
**Note:** *For full details about BOR characteristics, refer to the "Electrical characteristics" section in the device datasheet.*

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 5.2.1: 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 10. BOR thresholds**



## 5.2.3 Programmable voltage detector (PVD)

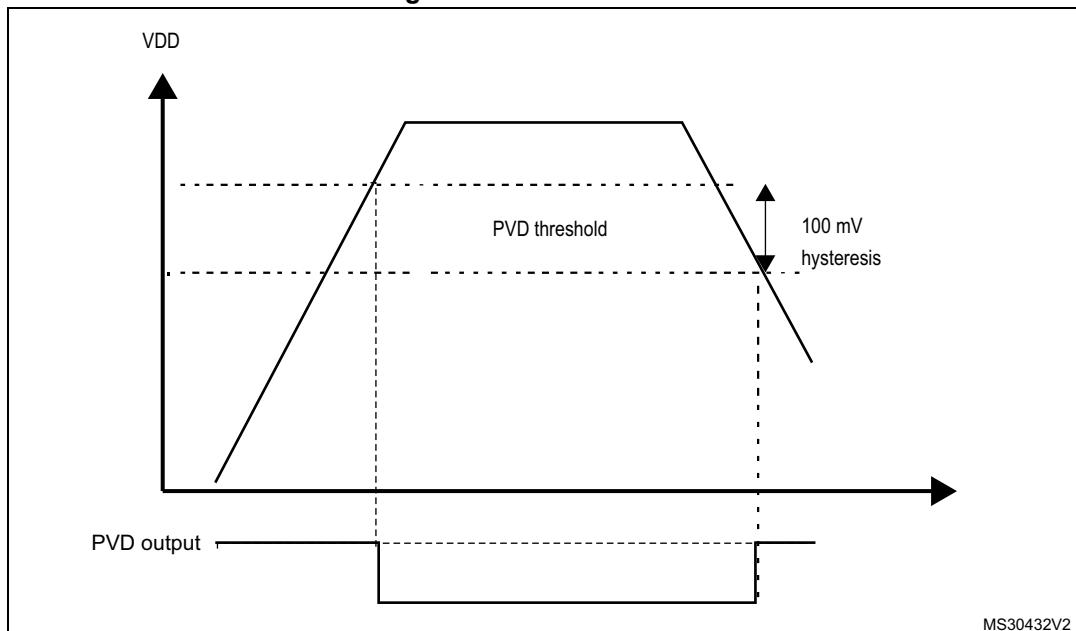
You can use the PVD to monitor the  $V_{DD}$  power supply by comparing it to a threshold selected by the PLS[2:0] bits in the [Section 5.4.1: PWR power control register \(PWR\\_CR\)](#).

The PVD is enabled by setting the PVDE bit.

A PVDO flag is available, in the [Section 5.4.2: PWR power control/status register \(PWR\\_CSR\)](#), to indicate if  $V_{DD}$  is higher or lower than the PVD threshold. This event is

internally connected to the EXTI line16 and can generate an interrupt if enabled through the EXTI registers. The PVD output interrupt can be generated when  $V_{DD}$  drops below the PVD threshold and/or when  $V_{DD}$  rises above the PVD threshold depending on EXTI line16 rising/falling edge configuration. As an example the service routine could perform emergency shutdown tasks.

**Figure 11. PVD thresholds**



### 5.3 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®-M4 with FPU 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.

**Table 15. Low-power mode summary**

Mode name	Entry	Wakeup	Effect on 1.2 V domain clocks	Effect on V <sub>DD</sub> domain clocks	Voltage regulator
<b>Sleep (Sleep now or Sleep-on-exit)</b>	WFI or Return from ISR	Any interrupt	CPU CLK OFF no effect on other clocks or analog clock sources	None	ON
	WFE	Wakeup event			
<b>Stop</b>	PDDS and LPDS bits + SLEEPDEEP bit + WFI or Return from ISR 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	ON or in low-power mode (depends on <i>PWR power control register (PWR_CR)</i> )
<b>Standby</b>	PDDS bit + SLEEPDEEP bit + WFI or Return from ISR or WFE	WKUP pin rising 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

### 5.3.1 Slowing down system clocks

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

For more details refer to [Section 6.3.3: RCC clock configuration register \(RCC\\_CFGR\)](#).

### 5.3.2 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 for STM32F446xx is controlled by the AHB1 peripheral clock enable register (RCC\_AHB1ENR), AHB2 peripheral by the clock enable register (RCC\_AHB2ENR), AHB3 by the peripheral clock enable register (RCC\_AHB3ENR) (see [Section 6.3.10: RCC AHB1 peripheral clock enable register \(RCC\\_AHB1ENR\)](#), [Section 6.3.11: RCC AHB2 peripheral clock enable register \(RCC\\_AHB2ENR\)](#) and [Section 6.3.12: RCC AHB3 peripheral clock enable register \(RCC\\_AHB3ENR\)](#), respectively).

Disabling the peripherals clocks in Sleep mode can be performed automatically by resetting the corresponding bit in RCC\_AHBxLPENR and RCC\_APBxLPENR registers.

### 5.3.3 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®-M4 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®-M4 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®-M4 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 and Shutdown modes 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 239: RTC block diagram](#)).

### 5.3.4 Sleep mode

#### Entering Sleep mode

The Sleep mode is entered according to [Entering low power mode](#), when the SLEEPDEEP bit in the Cortex®-M4 System Control register is cleared.

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

### Exiting Sleep mode

The Sleep mode is exited according to [Exiting low power mode](#).

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

**Table 16. Sleep-now entry and exit**

Sleep-now mode	Description
<b>Mode entry</b>	WFI (Wait for Interrupt) or WFE (Wait for Event) while: – SLEEPDEEP = 0 Refer to the Cortex®-M4 with FPU System Control register.
	On Return from ISR while: – SLEEPDEEP = 0 and – SLEEPONEXIT = 1 Refer to the Cortex®-M4 with FPU System Control register.
<b>Mode exit</b>	If WFI or Return from ISR was used for entry: – Interrupt: refer to <a href="#">Table 38: Vector table for STM32F446xx</a> If WFE was used for entry and SEVONPEND = 0 – Wakeup event: refer to <a href="#">Section 10.2.3: Wakeup event management</a> If WFE was used for entry and SEVONPEND = 1 – Interrupt event when disabled in NVIC: refer to <a href="#">Table 38: Vector table for STM32F446xx</a> – Wakeup event: refer to <a href="#">Section 10.2.3: Wakeup event management</a> .
<b>Wakeup latency</b>	None

### 5.3.5 Stop mode

The Stop mode is based on the Cortex®-M4 with FPU 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\_CR register. However this will induce an additional startup delay when waking up from Stop mode (see [Table 17](#)).

**Table 17. 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

## Entering Stop mode

The Stop mode is entered according to [Entering low power mode](#), when the SLEEPDEEP bit in Cortex®-M4 System Control register is set.

Refer to [Table 18](#) 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 \(PWR\\_CR\)](#).

Stop mode can be entered from Run mode and Low power run mode.

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 20.3: IWDG functional description](#).
- 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

The Stop mode is exited according to [Exiting low power mode](#).

Refer to [Table 18](#) 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 or low voltage 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 18. Stop mode entry and exit for STM32F446xx**

Stop mode	Description
<b>Mode entry</b>	<p>WFI (Wait for Interrupt) or WFE (Wait for Event) while:</p> <ul style="list-style-type: none"> <li>– Set SLEEPDEEP bit in Cortex®-M4 with FPU System Control register</li> <li>– Clear PDDS bit in Power Control register (PWR_CR)</li> <li>– Select the voltage regulator mode by configuring LPDS, MRUDS, LPUDS and UDEN bits in PWR_CR (see <a href="#">Table 17: Stop operating modes</a>).</li> </ul>
	<p>On Return from ISR while:</p> <ul style="list-style-type: none"> <li>– SLEEPDEEP bit is set in Cortex®-M4 with FPU System Control register</li> <li>– SLEEPONEXIT = 1</li> <li>– LPMS = “000” in PWR_C1: volatage regulator in main regulator mode</li> <li>– LPMS = “001” in PWR_C1: volatage regulator in low power regulator mode</li> </ul>
	<p><b>Note:</b> To enter the Stop mode, all EXTI Line pending bits in <a href="#">Pending register (EXTI_PR)</a>, all peripheral interrupts pending bits, the RTC Alarms (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.</p>
<b>Mode exit</b>	<p>If WFI or Return from ISTR was used for entry:</p> <p>All EXTI lines configured in Interrupt mode (the corresponding EXTI Interrupt vector must be enabled in the NVIC). Refer to <a href="#">Table 38: Vector table for STM32F446xx</a>.</p> <p>If WFE was used for entry and SEVONPEND = 0:</p> <p>All EXTI Lines configured in event mode. Refer to <a href="#">Section 10.2.3: Wakeup event management</a></p> <p>If WFE was used for entry and SEVONPEND = 1:</p> <p>Any EXTI line 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 <a href="#">Table 38: Vector table for STM32F446xx</a>.</p> <p>Wakeup event: refer to <a href="#">Section 10.2.3: Wakeup event management</a></p>
<b>Wakeup latency</b>	Refer to <a href="#">Table 17: Stop operating modes</a>

### 5.3.6 Standby mode

The Standby mode allows to achieve the lowest power consumption. It is based on the Cortex®-M4 with FPU 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 7](#)).

#### Entering Standby mode

The Standby mode is entered according to [Entering low power mode](#), when the SLEEPDEEP bit in the Cortex®-M4 with FPU System Control register is set.

Refer to [Table 19](#) 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 20.3: IWDG functional description](#).
- 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 according to [Exiting low power mode](#). The SBF status flag in the [PWR power control/status register \(PWR\\_CSR\)](#) indicates that the MCU was in Standby mode. All registers are reset after wakeup from standby except for [PWR power control/status register \(PWR\\_CSR\)](#).

Refer to [Table 19](#) for more details on how to exit Standby mode.

**Table 19. Standby mode entry and exit**

Standby mode	Description
<b>Mode entry</b>	WFI (Wait for Interrupt) or WFE (Wait for Event) while: – SLEEPDEEP is set in Cortex®-M4 with FPU with FPU System Control register – PDDS bit is set in Power Control register (PWR_CR) – no interrupt (for WFI or event (for WFE) is pending – WUF bit is cleared in Power Control/Status register (PWR_CR) – the RTC flag corresponding to the chosen wakeup source (RTC Alarm A, RTC Alarm B, RTC wakeup, Tamper or Timestamp flags) is cleared
	On Return from ISR while: – SLEEPDEEP bit is set in Cortex®-M4 with FPU with FPU System Control register and – SLEEPONEXIT = 1 and – PDDS bit is set in Power Control register (PWR_CR) and – no interrupt is pending and – WUF bit is cleared in Power Control/Status register (PWR_SR) and – the RTC flag corresponding to the chosen wakeup source (RTC Alarm A, RTC Alarm B, RTC wakeup, Tamper or Timestamp flags) is cleared
<b>Mode exit</b>	WKUP pin rising edge, RTC alarm (Alarm A and Alarm B), RTC wakeup, tamper event, time stamp event, external reset in NRST pin, IWDG reset.
<b>Wakeup latency</b>	Reset phase.

### I/O states in Standby mode

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

- Reset pad (still available)
- RTC\_AF1 pin (PC13) if configured for tamper, time stamp, RTC Alarm out, or RTC clock calibration out
- WKUP pin (PA0), if enabled

### Debug mode

By default, the debug connection is lost if the application puts the MCU in Stop or Standby mode while the debug features are used. This is due to the fact that the Cortex®-M4 with FPU 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 33.16.1: Debug support for low-power modes](#).

## 5.3.7 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 µA 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
- 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
- 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

### Safe RTC alternate function wakeup flag clearing sequence

If the selected RTC alternate function is set before the PWR wakeup flag (WUTF) is cleared, it will not be detected on the next event as detection is made once on the rising edge.

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) Clear the PWR Wakeup (WUF) flag
  - d) Enable the RTC alarm interrupt
  - e) 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) Clear the RTC Wakeup (WUTF) flag
  - c) Clear the PWR Wakeup (WUF) flag
  - d) Enable the RTC Wakeup interrupt
  - e) 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) Clear the PWR Wakeup (WUF) flag
  - d) Enable the RTC tamper interrupt
  - e) 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) Clear the PWR Wakeup (WUF) flag
  - d) Enable the RTC TimeStamp interrupt
  - e) Re-enter the low-power mode

## 5.4 Power control registers

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

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.	FISSR	FMSSR	UDEN[1:0]	ODSWEN	ODEN	
										rw	rw	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	CWUF	PDDS	LPDS	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rc_w1	rc_w1	rw	rw	

Bits 31:22 Reserved, must be kept at reset value.

Bit 21 **FISSR**: Flash Interface Stop while System Run

- 0: Flash interface clock run (Default value)
- 1: Flash Interface clock off.

*Note: This bit could not be set while executing with the Flash itself. It should be done with a specific routine executed from RAM.*

Bit 20 **FMSSR**: Flash Memory Stop while System Run

- 0: Flash standard mode (Default value)
- 1: Flash forced to be in STOP or Deep Power Down mode (depending of **FPDS** value bit) by hardware.

*Note: This bit could not be set while executing with the Flash itself. It should be done with a specific routine executed from RAM*

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 enable 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 STM32F446xx datasheet 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 and when the Prefetch is OFF.*

**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\_CSR 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 SBF Standby Flag (write).

**Bit 2 CWUF: Clear wakeup flag**

This bit is always read as 0.

- 0: No effect
- 1: Clear the WUF Wakeup Flag **after 2 System clock cycles**

**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

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

Address offset: 0x04

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

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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.	VOSRDY	Res.	Res.	Res.	Res.	BRE	EWUP1	EWUP2	Res.	Res.	Res.	BRR	PVDO	SBF	WUF
	r					rw	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 the Under-drive mode is entered in Stop mode and exited. 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<sub>BAT</sub> 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<sub>BAT</sub> 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<sub>BAT</sub> 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 EWUP1: Enable WKUP1 pin**

This bit is set and cleared by software.

0: WKUP1 pin is used for general purpose I/O. An event on the WKUP1 pin does not wakeup the device from Standby mode.

1: WKUP1 pin is used for wakeup from Standby mode and forced in input pull down configuration (rising edge on WKUP1 pin wakes-up the system from Standby mode).

*Note: This bit is reset by a system reset.*

**Bit 7 EWUP2: Enable WKUP2 pin**

This bit is set and cleared by software

0: WKUP2 pin is used for general purpose I/O. An event on the WKUP2 pin does not wakeup the device from Standby mode.

1: WKUP2 pin is used for wakeup from Standby mode and forced in input pull down configuration (rising edge on WKUP2 pin wakes-up the system from Standby mode).

*Note: This bit is reset by a system reset.*

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 **PVDO**: PVD output

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

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

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

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

Bit 1 **SBF**: Standby flag

This bit is set by hardware and cleared only by a POR/PDR (power-on reset/power-down reset) or by setting the CSBF bit in the [PWR power control register \(PWR\\_CR\)](#)

0: Device has not been in Standby mode

1: Device has been in Standby mode

Bit 0 **WUF**: Wakeup flag

This bit is set by hardware and cleared either by a system reset or by setting the CWUF bit in the PWR\_CR register.

0: No wakeup event occurred

1: A wakeup event was received from the WKUP pin or from the RTC alarm (Alarm A or Alarm B), RTC Tamper event, RTC TimeStamp event or RTC Wakeup).

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

## 5.5 PWR register map

The following table summarizes the PWR registers.

**Table 20. PWR register map and reset values**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5
0x000	PWR_CR	Res.	FISSR	FMSSR	UDEN[1:0]	UDEN[1:0]	ODSWEN	ODEN	VOS[1:0]	ADCDC1	Res.	MIRUDS	LPUDS	FPDS	DBP	EWUP1	EWUP2	Res.	Res.									
	Reset value									0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0		
0x004	PWR_CSR	Res.	UDRDY[1:0]	ODSRDY	ODRDY	ODRDY	ODSRDY	Res.	VOSRDY	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	

Refer to [Table 1 on page 56](#) for the register boundary addresses.

## 6 Reset and clock control (RCC)

### 6.1 Reset

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

#### 6.1.1 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 12](#)).

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<sup>®</sup>-M4 with FPU Application Interrupt and Reset Control Register must be set to force a software reset on the device. Refer to the Cortex<sup>®</sup>-M4 with FPU 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.

For further information on the user option bytes, refer to [Section 3: Embedded Flash memory \(FLASH\)](#).

#### 6.1.2 Power reset

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

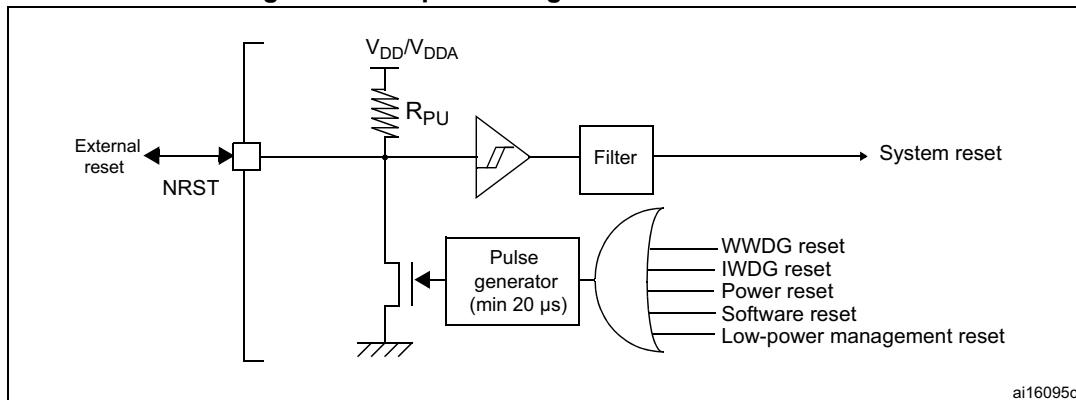
1. Power-on/power-down reset (POR/PDR reset) or brownout (BOR) reset
2. When exiting the Standby mode

A power reset sets all registers to their reset values except the Backup domain (see [Figure 12](#))

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  $\mu$ s for each internal reset source. In case of an external reset, the reset pulse is generated while the NRST pin is asserted low.

**Figure 12. Simplified diagram of the reset circuit**



The Backup domain has two specific resets that affect only the Backup domain (see [Figure 12](#)).

### 6.1.3 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.

## 6.2 Clocks

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

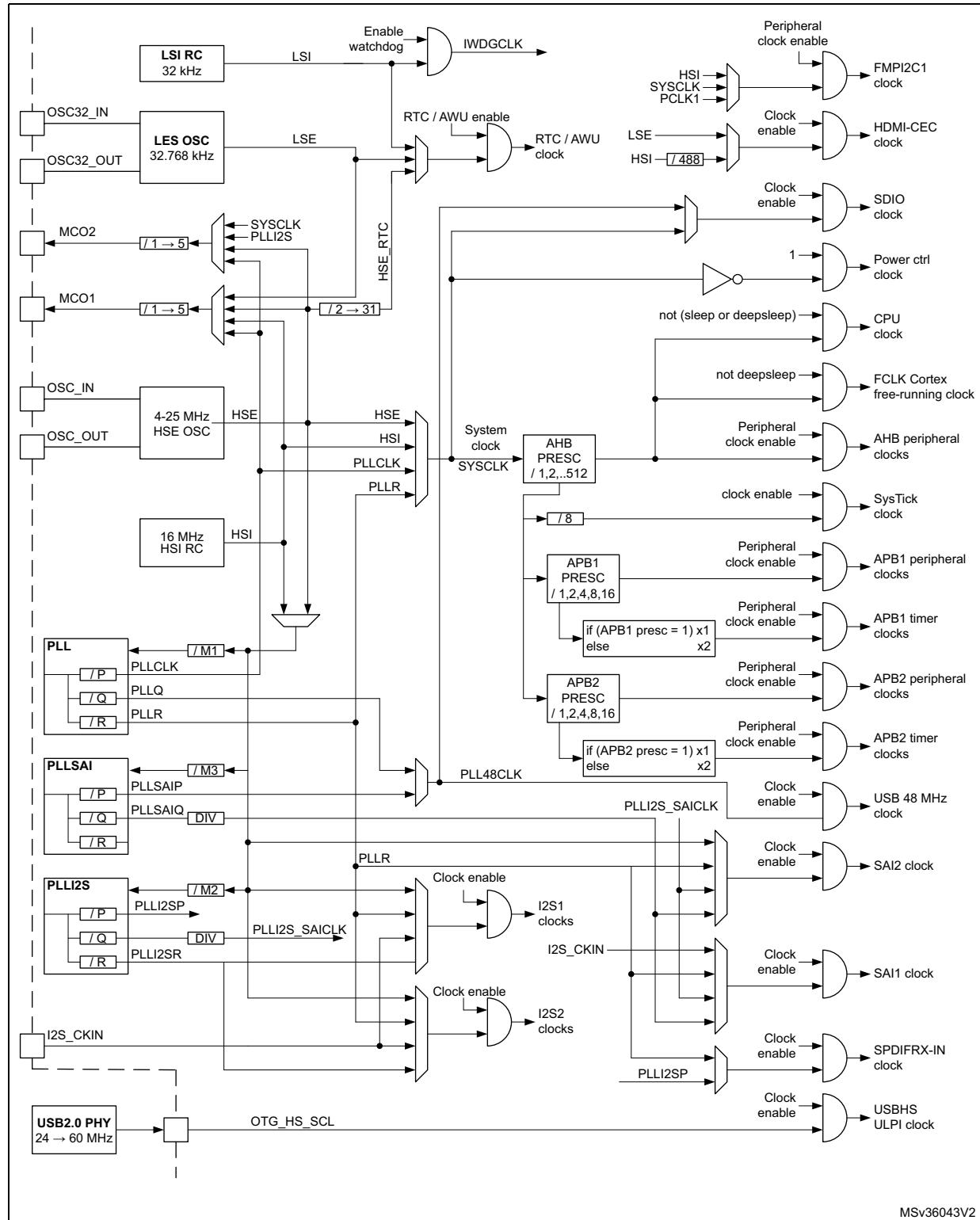
- HSI oscillator clock
- HSE oscillator clock
- Two main PLL (PLL) clocks

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 13. Clock tree



MSv36043V2

- For full details about the internal and external clock source characteristics, refer to the Electrical characteristics section in the device datasheet.
- When TIMPRE bit of the RCC\_DCKCFGR register is reset, if APBx prescaler is 1, then TIMxCLK = PCLKx,

otherwise  $\text{TIMxCLK} = 2 \times \text{PCLK}_x$ .

3. When **TIMPRE** bit in the **RCC\_DCKCFGR** register is set, if APB<sub>x</sub> prescaler is 1,2 or 4, then  $\text{TIMxCLK} = \text{HCLK}$ , otherwise  $\text{TIMxCLK} = 4 \times \text{PCLK}_x$ .

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 and, guarantee the appropriate frequency for peripherals that need a specific clock like USB OTG FS and HS, I<sup>2</sup>S, SAI, and SDIO.

Several prescalers are used to configure the AHB frequency, the high-speed APB (APB2) and the low-speed APB (APB1) domains. The maximum frequency of the AHB domain is 180 MHz. The maximum allowed frequency of the high-speed APB2 domain is 90 MHz. The maximum allowed frequency of the low-speed APB1 domain is 45 MHz

All peripheral clocks are derived from the system clock (SYSCLK) except for:

- The USB OTG FS clock (48 MHz), which is coming from a specific output of the PLL (PLLP) or PLLSAI (PLLSAIP)
- The SDIO clock (48 MHz) which is coming from a specific output of the PLL48CLK (PLLQ, PLLSAIP), or System Clock.
- I<sup>2</sup>S1/2 clocks

To achieve high-quality audio performance and for a better configuration flexibility, the I<sup>2</sup>S1 clock and I<sup>2</sup>S2 clock (which are respectively clocks for I<sup>2</sup>Ss mapped on APB1 and APB2) can be derived from four sources: specific main PLL output, a specific PLLI2S output, from an external clock mapped on the I<sup>2</sup>S\_CKIN pin or from HSI/HSE

- SAIs clock

The SAI1/SAI2 clocks are generated from a specific PLL (Main PLL, PLLSAI, or PLLI2S), from an external clock mapped on the I<sup>2</sup>S\_CKIN pin or from HSI/HSE clock.

The PLLSAI can be used as clock source for SAI1 peripheral in case the PLLI2S is programmed to achieve another audio sampling frequency (49.152 MHz or 11.2896 MHz), and the application requires both frequencies at the same time.

- The USB OTG HS (60 MHz) clock which is provided from the external PHY.
- SPDIF-Rx clock

The SPDIF-Rx clock is generated from a specific output of PLLI2S or from a specific output of main PLL.

- HDMI-CEC clock which is generated from LSE or HSI divided by 488.
- FMPI2C1 clock which can also be generated from HSI, SYSCLK or APB1 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\_DCKCFGR** register is reset:  
If the APB prescaler is configured to a division factor of 1, the timer clock frequencies ( $\text{TIMxCLK}$ ) are set to  $\text{PCLK}_x$ . Otherwise, the timer clock frequencies are twice the frequency of the APB domain to which the timers are connected:  $\text{TIMxCLK} = 2 \times \text{PCLK}_x$ .
- If **TIMPRE** bit in **RCC\_DCKCFGR** register is set:  
If the APB prescaler is configured to a division factor of 1, 2 or 4, the timer clock frequencies ( $\text{TIMxCLK}$ ) are set to  $\text{HCLK}$ . Otherwise, the timer clock frequencies is four times the frequency of the APB domain to which the timers are connected:  $\text{TIMxCLK} = 4 \times \text{PCLK}_x$ .

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®-M4 with FPU free-running clock. For more details, refer to the Cortex®-M4 with FPU technical reference manual.

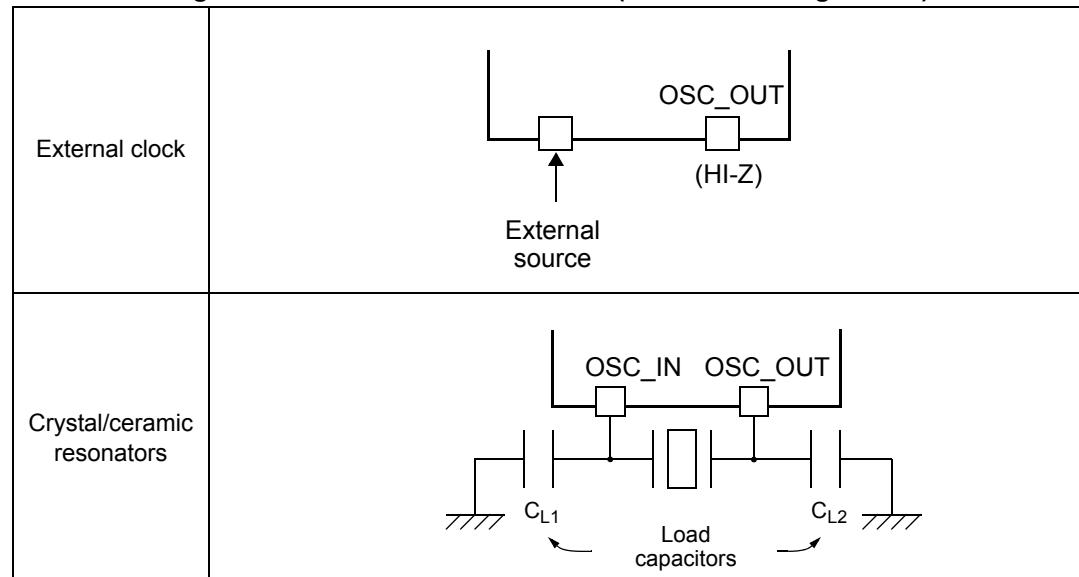
## 6.2.1 HSE clock

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

- HSE external crystal/ceramic resonator
- HSE 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 14. HSE/ LSE clock sources (hardware configuration)**



### 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 14](#).

### 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 14](#). 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)*.

### 6.2.2 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  $T_A = 25^\circ\text{C}$ .

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 6.2.7: Clock security system \(CSS\) on page 122](#).

### 6.2.3 PLL configuration

The STM32F446xx devices feature three PLLs:

- A main PLL (PLL) clocked by the HSE or HSI oscillator and featuring three different output clocks:
  - The first output is used to generate the high speed system clock (up to 180 MHz)
  - The second output can be used to generate the clock for the USB OTG FS (48 MHz) or the SDIO ( $\leq 48$  MHz).
  - The third output can be used to generate the clock for I2S1 and I2S2 clocks, SPDIF-Rx clock or the high speed system clock.
- Two dedicated PLLs (PLLI2S and PLLSAI) used to generate an accurate clock to achieve high-quality audio performance on the I2S and SAIs interfaces. PLLSAI and PLLI2S are also used to generate SPDIF-Rx clock or the 48 MHz clock for USB OTG FS and SDIO.

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, R and Q).

The PLLI2S and PLLSAI use the same input clock as PLL (PLLSRC bit is common to both PLLs). However, the PLLI2S and PLLSAI have dedicated enable/disable and division factors (M, N, P, R 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 Clock Configuration Register \(RCC\\_DCKCFGR\)](#) can be used to configure PLL, PLLI2S, and PLLSAI.

## 6.2.4 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 14](#).

## 6.2.5 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 \(RCC\\_CIR\)](#).

## 6.2.6 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.

## 6.2.7 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®-M4 with FPU 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 was enabled, it is also disabled when the HSE fails.

## 6.2.8 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.
- 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 6.2.5: LSI clock on page 121](#) 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).

**Note:**

*To read the RTC calendar register when the APB1 clock frequency is less than seven times the RTC clock frequency ( $f_{APB1} < 7 \times f_{RTCLK}$ ), 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.*

## 6.2.9 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.

## 6.2.10 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 \(RCC\\_CFGR\)](#).
- 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 \(RCC\\_CFGR\)](#).

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

The selected clock to output onto MCO must not exceed 100 MHz (the maximum I/O speed).

### 6.2.11 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 15](#) and [Figure 16](#).

#### 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.

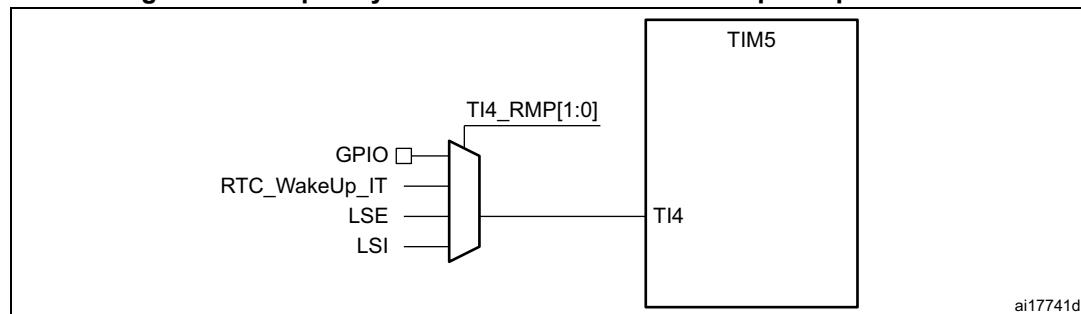
The basic concept consists in providing a relative measurement (e.g. HSI/LSE ratio): the precision is therefore tightly linked to the ratio between the two clock sources. The greater the ratio, the better the measurement.

It is also possible to measure the LSI frequency: this is useful for applications that do not have a crystal. The ultralow-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 15. Frequency measurement with TIM5 in Input capture mode**

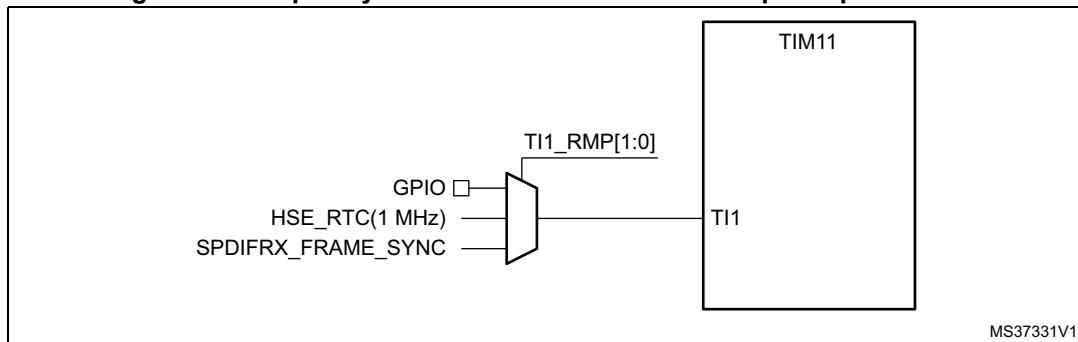


ai17741d

### 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, by SPDIF-Rx Frame Sync 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 16. Frequency measurement with TIM11 in Input capture mode**



## 6.3 RCC registers

Refer to [Section 1.1: List of abbreviations for registers](#) for a list of abbreviations used in register descriptions.

### 6.3.1 RCC clock control register (RCC\_CR)

Address offset: 0x00

Reset value: 0x0000 XX83 where X is undefined.

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	PLLSAI RDY	PLLSAI ON	PLL2S RDY	PLL2S ON	PLL RDY	PLL ON	Res.	Res.	Res.	Res.	CSS ON	HSE BYP	HSE RDY	HSE ON
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]							
r	r	r	r	r	r	r	r	rw	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

### 6.3.2 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:

- $f_{(VCO\ clock)} = f_{(PLL\ clock\ input)} \times (PLL_N / PLL_M)$
- $f_{(PLL\ general\ clock\ output)} = f_{(VCO\ clock)} / PLL_P$
- $f_{(USB\ OTG\ FS,\ SDIO)} = f_{(VCO\ clock)} / PLL_Q$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	PLL_R[2:0]			PLL_Q[3:0]				Res.	PLL_SRC	Res.	Res.	Res.	Res.	PLL_P[1:0]	
	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
Res.	PLL_N[8:0]								PLL_M[5:0]						
	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 30:28 **PLL\_R[2:0]**: Main PLL division factor for I2Ss, SAIs, SYSTEM and SPDIF-Rx clocks

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

Clock frequency = VCO frequency / PLLR with  $2 \leq PLLR \leq 7$

000: PLLR = 0, wrong configuration

001: PLLR = 1, wrong configuration

010: PLLR = 2

011: PLLR = 3

...

111: PLLR = 7

Bits 27:24 **PLL\_Q[3:0]**: Main PLL (PLL) division factor for USB OTG FS, SDIOclocks

Set and cleared by software to control the frequency of USB OTG FS clock and the SDIOclock. These bits should be written only if PLL is disabled.

**Caution:** The USB OTG FS requires a 48 MHz clock to work correctly. The SDIOneeds a frequency lower than or equal to 48 MHz to work correctly.

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

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 **PLL\_SRC**: 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[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 180 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 **PLLN[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 **PLLM[5:0]**: Division factor for the main PLL (PLL) input clock

Set and cleared by software to divide the PLL input clock before the VCO. These bits can be written only when the PLL is 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  $2 \leq \text{PLLM} \leq 63$

000000: PLLM = 0, wrong configuration

000001: PLLM = 1, wrong configuration

000010: PLLM = 2

000011: PLLM = 3

000100: PLLM = 4

...

111110: PLLM = 62

111111: PLLM = 63

### 6.3.3 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[1:0]		MCO2 PRE[2:0]		MCO1 PRE[2:0]		Res.		MCO1		RTCPRE[4:0]					
rw		rw	rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PPRE2[2:0]		PPRE1[2:0]		Res.	Res.			HPRE[3:0]		SWS[1:0]		SW[1:0]			
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 Reserved, must be kept at reset value.

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 90 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 45 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.

**Caution:** The AHB clock frequency must be at least 25 MHz when the Ethernet is used.

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[1:0]**: 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: PLL\_R used as the system clock

Bits 1:0 **SW[1:0]**: 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\_P selected as system clock

11: PLL\_R selected as system clock

**6.3.4 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	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 **LSEIRDYIE:** 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 PLLSAIRDYDIE is set.  
It is cleared by software by setting the PLLSAIRDYDIE 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 PLLI2SRDYDIE is set.  
It is cleared by software by setting the PLLI2SRDYDIE 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 PLLRDYDIE is set.  
It is cleared by software setting the PLLRDYDIE 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

### 6.3.5 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.	OTGHS 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.	CRC RST	Res.	Res.	Res.	Res.	GPIOH RST	GPIOG RST	GPIOE RST	GPIOF RST	GPIOE RST	GPIOF RST	GPIOC RST	GPIOB RST	GPIOA RST
			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 28:23 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 11:8 Reserved, must be kept at reset value.

**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

### 6.3.6 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	Res.	Res.	Res.	Res.	Res.	DCMI RST	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

Bits 6:1 Reserved, must be kept at reset value.

Bit 0 **DCMIRST**: Camera interface reset

Set and cleared by software.

0: does not reset the Camera interface

1: resets the Camera interface

### 6.3.7 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**: QUADSPI module reset

Set and reset by software

0: does not reset QUADSPI module

1: resets QUADSPI module

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

### 6.3.8 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
Res.	Res.	DAC RST	PWR RST	CECRS T	CAN2 RST	CAN1 RST	FMPI2C1 RST	I2C3 RST	I2C2 RST	I2C1 RST	UART5 RST	UART4 RST	UART3 RST	UART2 RST	SPDIFRX RST
		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
SPI3 RST	SPI2 RST	Res.	Res.	WWDG RST	Res.	Res.	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

Bits 31:30 Reserved, must be kept at reset value.

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

Bit 27 **CECRST:** CEC reset

Set and cleared by software.

0: does not reset CEC

1: resets CEC

Bit 26 **CAN2RST:** CAN2 reset

Set and cleared by software.

0: does not reset CAN2

1: resets CAN2

Bit 25 **CAN1RST:** CAN1 reset

Set and cleared by software.

0: does not reset CAN1

1: resets CAN1

Bit 24 **IFMPI2C1RST:** FMPI2C1 reset

Set and cleared by software.

0: does not reset FMPI2C1

1: resets FMPI2C1

- 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 **SPDIFRXRST:** SPDIF-Rx reset  
Set and cleared by software.  
0: does not reset SPDIF-Rx  
1: resets SPDIF-Rx
- 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
- Bits 10:9 Reserved, must be kept at reset value.

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

### 6.3.9 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
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SAI2 RST	SAI1 RST	Res.	Res.	Res.	TIM11 RST	TIM10 RST	TIM9 RST
								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	SDIO RST	Res.	Res.	ADC RST	Res	Res	USART6 RST	USART1 RST	Res.	Res.	TIM8 RST	TIM1 RST
	rw	rw	rw	rw			rw			rw	rw			rw	rw

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

Bit 23 **SAI2RST**: SAI2 reset

This bit is set and reset 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

Bits 21:19 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 SDIORST:** SDIO reset

This bit is set and cleared by software.

0: does not reset the SDIO module

1: resets the SDIO 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

Bits 7:6 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

### 6.3.10 RCC AHB1 peripheral clock enable 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	OTGHS EN	Res.	Res.	Res.	Res.	Res.	Res.	DMA2 EN	DMA1 EN	Res.	Res.	BKP SRAMEN	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.	CRC EN	Res.	Res.	Res.	Res.	GPIOH EN	GPIOG EN	GPIOF EN	GPIOE EN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN	
			rw					rw	rw	rw	rw	rw	rw	rw	rw	

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

Bits 20:19 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 11:8 Reserved, must be kept at reset value.

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 **GIPOBEN**: 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

### 6.3.11 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.	OTGFSEN	Res.	Res.	Res.	Res.	Res.	Res.	DCMIEN							
								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

Bits 6:1 Reserved, must be kept at reset value.

Bit 0 **DCMIEN**: Camera interface enable

This bit is set and cleared by software.

0: Camera interface clock disabled

1: Camera interface clock enabled

### 6.3.12 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.	Res.														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	QSPIEN	FMCEN													
														rw	rw

Bits 31:2 Reserved, must be kept at reset value.

Bit 1 **QSPIEN**: QUADSPI memory controller module clock enable

This bit is set and cleared by software.

0: QUADSPI module clock disabled

1: QUADSPI module clock enabled

Bit 0 **FMCEN**: Flexible memory controller module clock enable

This bit is set and cleared by software.

0: FMC module clock disabled

1: FMC module clock enabled

### 6.3.13 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
Res.	Res.	DAC EN	PWR EN	CEC EN	CAN2 EN	CAN1 EN	FMP12C1 EN	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART3 EN	USART2 EN	SPDIFRX EN
		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
SPI3 EN	SPI2 EN	Res.	Res.	WWDG EN	Res.	Res.	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

Bits 31:30 Reserved, must be kept at reset value.

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

Bit 27 **CECEN:** CEC interface clock enable

This bit is set and cleared by software.

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

Bit 26 **CAN2EN:** CAN 2 clock enable

This bit is set and cleared by software.

- 0: CAN 2 clock disabled
- 1: CAN 2 clock enabled

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 **FMP12C1EN:** FMP12C1 clock enable

This bit is set and cleared by software.

- 0: FMP12C1 clock disabled
- 1: FMP12C1 clock enabled

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 **SPDIFRXEN:** SPDIF-Rx clock enable

This bit is set and cleared by software.

0: SPDIF-Rx clock disabled

1: SPDIF-Rx clock enabled

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

Bits 10:9 Reserved, must be kept at reset value.

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

**6.3.14 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
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SAI2 EN	SAI1 EN	Res.	Res.	Res.	TIM11 EN	TIM10 EN	TIM9 EN
								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	SDIO EN	ADC3 EN	ADC2 EN	ADC1 EN	Res.	Res.	USART6 EN	USART1 EN	Res.	Res.	TIM8 EN	TIM1 EN
	rw	rw	rw	rw	rw	rw	rw			rw	rw			rw	rw

Bits 31:24 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

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

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 **SDIOEN:** SDIO clock enable

This bit is set and cleared by software.

0: SDIO module clock disabled

1: SDIO module clock enabled

Bit 10 **ADC3EN:** ADC3 clock enable

This bit is set and cleared by software.

0: ADC3 clock disabled

1: ADC3 clock disabled

- Bit 9 **ADC2EN:** ADC2 clock enable  
This bit is set and cleared by software.  
0: ADC2 clock disabled  
1: ADC2 clock disabled
- Bit 8 **ADC1EN:** ADC1 clock enable  
This bit is set and cleared by software.  
0: ADC1 clock disabled  
1: ADC1 clock disabled
- Bits 7:6 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

### 6.3.15 RCC AHB1 peripheral clock enable in low power mode register (RCC\_AHB1LPENR)

Address offset: 0x50

Reset value: 0x6067 90FF

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	OTGHS LPEN	Res.	Res.	Res.	Res.	Res.	Res.	DMA2 LPEN	DMA1 LPEN	Res.	Res.	BKP SRAM LPEN	SRAM2 LPEN	SRAM1 LPEN
	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.	Res.	CRC LPEN	Res.	Res.	Res.	Res.	GPIOH LPEN	GPIOG LPEN	GPIOF LPEN	GPIOE LPEN	GPIOD LPEN	GPIOC LPEN	GPIOB LPEN	GPIOA LPEN
	rw			rw				rw	rw	rw	rw	rw	rw	rw	rw

- 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
- Bits 20:19 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
- Bits 14:13 Reserved, must be kept at reset value.
- 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 11:8 Reserved, must be kept at reset value.
- 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

Bit 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 **GPIOBLPEN:** 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

### 6.3.16 RCC AHB2 peripheral clock enable in low power mode register (RCC\_AHB2LPENR)

Address offset: 0x54

Reset value: 0x0000 0081

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 LPEN	Res.	Res.	Res.	Res.	Res.	Res.	DCMI LPEN							
								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

Bits 6:1 Reserved, must be kept at reset value.

Bit 0 **DCMILPEN**: Camera interface enable during Sleep mode

This bit is set and cleared by software.

0: Camera interface clock disabled during Sleep mode

1: Camera interface clock enabled during Sleep mode

### 6.3.17 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	rw

Bits 31:2Reserved, must be kept at reset value.

Bit 1 **QSPILPEN**: QUADSPI memory controller module clock enable during Sleep mode

This bit is set and cleared by software.

0: QUADSPI module clock disabled during Sleep mode

1: QUADSPI module 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

### 6.3.18 RCC APB1 peripheral clock enable in low power mode register (RCC\_APB1LPENR)

Address offset: 0x60

Reset value: 0x3FFF C9FF

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.	DAC LPEN	PWR LPEN	CECLP EN	CAN2 LPEN	CAN1 LPEN	FMPI2C1 LPEN	I2C3 LPEN	I2C2 LPEN	I2C1 LPEN	UART5 LPEN	UART4 LPEN	USART3 LPEN	USART2 LPEN	SPDIFRX LPEN
		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
SPI3 LPEN	SPI2 LPEN	Res.	Res.	WWDG LPEN	Res.	Res.	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

Bits 31:30 Reserved, must be kept at reset value.

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

Bit 27 **CECLPEN:** CEC clock enable during Sleep mode

This bit is set and cleared by software.

0: CEC clock disabled during Sleep mode

1: CEC clock enabled during Sleep mode

Bit 26 **CAN2LPEN:** CAN 2 clock enable during Sleep mode

This bit is set and cleared by software.

0: CAN 2 clock disabled during sleep mode

1: CAN 2 clock enabled during sleep mode

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 **FMPI2C1LPEN:** FMPI2C1 clock enable during Sleep mode

This bit is set and cleared by software.

0: FMPI2C1 clock disabled during Sleep mode

1: FMPI2C1 clock enabled during Sleep mode

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 **SPDIFLPEN:** SPDIF-Rx clock enable during Sleep mode  
 This bit is set and cleared by software.  
 0: SPDIF-Rx clock disabled during Sleep mode  
 1: SPDIF-Rx clock enabled during Sleep mode
- 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
- Bits 10:9 Reserved, must be kept at reset value.
- 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

### 6.3.19 RCC APB2 peripheral clock enabled in low power mode register (RCC\_APB2LPENR)

Address offset: 0x64

Reset value: 0x00C7 FF33

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.	Res.	Res.	TIM11 LPEN	TIM10 LPEN	TIM9 LPEN
								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	SDIO LPEN	ADC3 LPEN	ADC2 LPEN	ADC1 LPEN	Res.	Res.	USART6 LPEN	USART1 LPEN	Res.	Res.	TIM8 LPEN	TIM1 LPEN
	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

Bits 21:19 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 **SDIOLPEN:** SDIO clock enable during Sleep mode  
 This bit is set and cleared by software.  
 0: SDIO module clock disabled during Sleep mode  
 1: SDIO 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 disabled 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 disabled 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 disabled during Sleep mode
- Bits 7:6 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

### 6.3.20 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, 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\_CR)* has to be set before these can be modified. Refer to *Section 6.1.1: System reset on page 115* for further information. These bits are only reset after a Backup domain Reset (see *Section 6.1.3: 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.	Res.	Res.	LSEMOD	LSEBYP	LSERDY	LSEON
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 7:4 Reserved, must be kept at reset value.

Bit 3 **LSEMOD:** External low-speed oscillator mode

This bit is set and cleared by software to select the low speed oscillator crystal mode. Two power modes are available. This bit can be written only when the LSE clock is disabled.

0: LSE oscillator "low power" mode selection

1: LSE oscillator "high drive" mode selection

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

**6.3.21 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:  $0 \leq$  wait state  $\leq 3$ , 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.						
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.	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 **IWDGRSTF:** Independent watchdog reset flag

This bit is set by hardware when an independent watchdog reset from V<sub>DD</sub> domain occurs.

Cleared by writing to the RMVF bit.

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

**Bit 28 SFTRSTF:** 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

### 6.3.22 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		
SSCGEN	SPREADSEL	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	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.

### 6.3.23 RCC PLLI2S configuration register (RCC\_PLLI2SCFGR)

Address offset: 0x84

Reset value: 0x2400 3010

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

This register is used to configure the PLLI2S clock outputs according to the formulas:

$$f_{(VCO\ clock)} = f_{(PLLI2S\ clock\ input)} \times (PLLI2SN / PLLI2SM)$$

$$f_{(PLL\ I2S\ clock\ output)} = f_{(VCO\ clock)} / PLLI2SR$$

$$f_{(PLL\ SPDIFRX\ clock\ output)} = f_{(VCO\ clock)} / PLLI2SP$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	PLLI2SR[2:0]			PLLI2SQ[3:0]				Res.	Res.	Res.	Res.	Res.	Res.	PLLI2SP[1:0]	
	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.	PLLI2SN[8:0]								PLLI2SM[5:0]						
	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 30:28 **PLL2SR[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 26.6.4: Clock generator](#) 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 **PLL2SQ[3:0]**: PLLI2S division factor for SAI1 clock

These bits are set and cleared by software to control the SAI1 clock frequency.

They should be written when the PLLI2S is disabled.

SAI1 clock frequency = VCO frequency / PLLI2SQ with  $2 \leq \text{PLLI2SIQ} \leq 15$

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 23:18 Reserved, must be kept at reset value.

Bits 17:16 **PLL2SP[1:0]**: PLLI2S division factor for SPDIF-Rx clock

These bits are set and cleared by software to control the SPDIF-Rx clock frequency.

They should be written when the PLLI2S is disabled.

**Caution:** The software has to set these bits correctly to ensure that the output frequency doesn't exceed 120 MHz on this output.

PLL output clock frequency = VCO frequency / PLLI2SP with  $\text{PLLI2SIP} = 2, 4, 6 \text{ or } 8$

00: PLLI2SP = 2

01: PLLI2SP = 4

10: PLLI2SP = 6

11: PLLI2SP = 8

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 × 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 **PLL2SM[5:0]**: Division factor for audio PLL (PLLI2S) input clock

Set and cleared by software to divide PLLI2S input clock before the VCO.

These bits can be written only when PLLI2S is 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 / PLLI2S with  $2 \leq \text{PLLI2SM} \leq 63$

000000: PLLI2SM = 0, wrong configuration

000001: PLLI2SM = 1, wrong configuration

000010: PLLI2SM = 2

000011: PLLI2SM = 3

000100: PLLI2SM = 4

...

111110: PLLI2SM = 62

111111: PLLI2SM = 63

### 6.3.24 RCC PLL 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:

- $f_{(VCO\ clock)} = f_{(PLL\ SAI\ clock\ input)} \times (PLL\ SAIN / PLLM)$
- $f_{(PLL\ SAI\ 48MHz\ clock\ output)} = f_{(VCO\ clock)} / PLL\ SAIP$
- $f_{(PLL\ SAI1\ clock\ output)} = f_{(VCO\ clock)} / PLL\ SAIQ$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Res.	Res.	Res.	Res.	PLL\ SAIQ[3:0]				Res.	Res.	Res.	Res.	Res.	Res.	PLL\ SAIP[1:0]		
				rw	rw	rw	rw							rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	PLL\ SAIN[8:0]										PLL\ SAIM[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 **PLL\ SAIQ**: PLLSAI division factor for SAIs clock

Set and reset by software to control the frequency of SAIs clock.

These bits should be written when the PLLSAI is disabled.

SAI1 clock frequency = VCO frequency / PLLSAIQ with  $2 \leq PLL\ SAIQ \leq 15$

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 **PLL\ SAIP**: PLLSAI division factor for 48 MHz clock

These bits are set and cleared by software to control the output clock frequency.

They should be written when the PLLSAI is disabled.

**Caution:** The software has to set these bits correctly to ensure that the output frequency not exceed 120 MHz on this output

PLL output clock frequency = VCO frequency / PLLSAIP with PLLSAIP = 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:** 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  $\times$  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 PLLSAIM: Division factor for audio PLLSAI input clock

Set and cleared by software to divide PLLSAI input clock before the VCO.

These bits can be written only when PLLSAI is 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 / PLLSAI with  $2 \leq \text{PLLSAIM} \leq 63$

000000: PLLSAIM = 0, wrong configuration

000001: PLLSAIM = 1, wrong configuration

000010: PLLSAIM = 2

000011: PLLSAIM = 3

000100: PLLSAIM = 4

...

111110: PLLSAIM = 62

111111: PLLSAIM = 63

### 6.3.25 RCC Dedicated Clock Configuration Register (RCC\_DCKCFGR)

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 peripherals 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.	I2S2SRC		I2S1SRC		TIMPRE	SAI2SRC		SAI1SRC		Res.	Res.	Res.	Res.
			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.	PLLSAIDIVQ					Res.	Res.	Res.	PLLIS2DIVQ				
			rw	rw	rw	rw	rw				rw	rw	rw	rw	rw

Bits 31:29 Reserved, must be kept at reset value.

Bits 28:27 **I2S2SRC:** I2S APB2 clock source selection

Set and reset by software to control the frequency of the APB2 I2S clock.

These bits should be written when the PLL, PLLSAI and PLLI2S are disabled.

00: I2S2 clock frequency =  $f_{(PLL2S\_R)}$

01: I2S2 clock frequency = I2S\_CKIN Alternate function input frequency

10: I2S2 clock frequency =  $f_{(PLL\_R)}$

11: I2S2 clock frequency = HSI/HSE depends on PLLSRC bit (PLLCFGR[22])

Bits 26:25 **I2S1SRC:** I2S APB1 clock source selection

Set and reset by software to control the frequency of the APB1 I2S clock.

These bits should be written when the PLL, PLLSAI and PLLI2S are disabled.

00: I2S1 clock frequency =  $f_{(PLL2S\_R)}$

01: I2S1 clock frequency = I2S\_CKIN Alternate function input frequency

10: I2S1 clock frequency =  $f_{(PLL\_R)}$

11: I2S1 clock frequency = HSI/HSE depends on PLLSRC bit (PLLCFGR[22])

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:  
TIMxCLK = 2xPCLKx.

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:  
TIMxCLK = 4xPCLKx.

Bits 23:22 **SAI2SRC:** SAI2 clock source selection

These bits are set and cleared by software to control the SAI2 clock frequency.

They should be written when the PLL, PLLSAI and PLLI2S are disabled.

00: SAI2 clock frequency =  $f_{(PLLSAI\_Q)} / PLLSAIDIVQ$

01: SAI2 clock frequency =  $f_{(PLL2S\_Q)} / PLLI2SDIVQ$

10: SAI2 clock frequency =  $f_{(PLL\_R)}$

11: SAI2 clock frequency = HSI/HSE depends on PLLSRC (PLLCFGR[22])

Bits 21:20 **SAI1SRC:** SAI1 clock source selection

These bits are set and cleared by software to control the SAI1-A clock frequency.

They should be written when the PLLSAI and PLLI2S are disabled.

00: SAI1 clock frequency =  $f_{(PLLSAI\_Q)} / PLLSAIDIVQ$

01: SAI1 clock frequency =  $f_{(PLL2S\_Q)} / PLLI2SDIVQ$

10: SAI1 clock frequency =  $f_{(PLL\_R)}$

11: I2S\_CKIN Alternate function input frequency

Bits 19:13 Reserved, must be kept at reset value.

Bits 12:8 **PLLSAIDIVQ[4:0]**: PLLSAI division factor for SAIs clock

These bits are set and reset by software to control the SAIs 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 **PLL2SDIVQ[4:0]**: PLLI2S division factor for SAIs clock

These bits are set and reset by software to control the SAIs 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

### 6.3.26 RCC clocks gated enable register (CKGATENR)

Address offset: 0x90

Reset value: 0x0000 0000

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

This register allows to enable or disable the clock gating for the specified IPs.

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.	RCC_CKEN	FLITF_CKEN	SRAM_CKEN	SPARE_CKEN	CM4DBG_CKEN	AHB2APB2_CKEN	AHB2APB1_CKEN								
									rw	rw	rw	rw	rw	rw	rw

Bits 31:7 Reserved, must be kept at reset value.

Bit 6 **RCC\_CKEN**: RCC clock enable

0: the clock gating is enabled

1: the clock gating is disabled, the clock is always enabled.

Bit 5 **FLITF\_CKEN**: Flash Interface clock enable

0: the clock gating is enabled

1: the clock gating is disabled, the clock is always enabled.

- Bit 4 **SRAM\_CKEN:** SRQAM controller clock enable  
0: the clock gating is enabled  
1: the clock gating is disabled, the clock is always enabled.
- Bit 3 **SPARE\_CKEN:** Spare clock enable  
0: the clock gating is enabled  
1: the clock gating is disabled, the clock is always enabled.
- Bit 2 **CM4DBG\_CKEN:** Cortex M4 ETM clock enable  
0: the clock gating is enabled  
1: the clock gating is disabled, the clock is always enabled.
- Bit 1 **AHB2APB2\_CKEN:** AHB to APB2 Bridge clock enable  
0: the clock gating is enabled  
1: the clock gating is disabled, the clock is always enabled.
- Bit 0 **AHB2APB1\_CKEN:** AHB to APB1 Bridge clock enable  
0: the clock gating is enabled  
1: the clock gating is disabled, the clock is always enabled.

### 6.3.27 RCC dedicated clocks configuration register 2 (DCKCFGR2)

Address offset: 0x94

Reset value: 0x0000 0000

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

This register allows to enable or disable the clock gating for the specified IPs.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	SPDIFRXSEL	SDIOSEL	CK48MSEL	CECSEL	Res.	Res.	FMP12C1SEL [1:0]	Res.						
		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.	Res.

Bits 31:30 Reserved, must be kept at reset value.

Bit 29 **SPDIFRXSEL:** SPDIF-Rx clock selection

- 1:  $f_{(PLL2S\_P)}$
- 0:  $f_{(PLL\_R)}$

Bit 28 **SDIOSEL:** SDIO clock selection

- 1: Clock System
- 0: Clock 48 MHz

Bit 27 **CK48MSEL:** SDIO/USBFS/HS clock selection

- 1:  $f_{(PLLSAI\_P)}$
- 0:  $f_{(PLL\_Q)}$

Bit 26 **CECSEL:** HDMI CEC clock source selection

- 1: LSE
- 0: HSI/488

Bits 25:24 Reserved, must be kept at reset value.

Bits 23:22 **FMPI2C1SEL[1:0]**: I2C4 kernel clock source selection

00: APB clock selected as FMPI2C1 clock

01: System clock selected as FMPI2C1 clock

10: HSI clock selected as FMPI2C1 clock

11: APB clock selected as FMPI2C1 clock (same as “00”)

Bits 21:0 Reserved, must be kept at reset value.

### 6.3.28 RCC register map

*Table 21* gives the register map and reset values.

**Table 21. RCC register map and reset values**

**Table 21. RCC register map and reset values (continued)**

**Table 21. RCC register map and reset values (continued)**

**Table 21. RCC register map and reset values (continued)**

Refer to [Section 2.2.2: Memory map and register boundary addresses](#) for the register boundary addresses.

## 7 General-purpose I/Os (GPIO)

### 7.1 GPIO introduction

Each general-purpose I/O port has four 32-bit configuration registers (GPIOx\_MODER, GPIOx\_OTYPER, GPIOx\_OSPEEDR and GPIOx\_PUPDR), two 32-bit data registers (GPIOx\_IDR and GPIOx\_ODR), a 32-bit set/reset register (GPIOx\_BSRR), a 32-bit locking register (GPIOx\_LCKR) and two 32-bit alternate function selection register (GPIOx\_AFRH and GPIOx\_AFRL).

### 7.2 GPIO main features

- Up to 16 I/Os under control
- Output states: push-pull or open drain + pull-up/down
- Output data from output data register (GPIOx\_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 (GPIOx\_IDR) or peripheral (alternate function input)
- Bit set and reset register (GPIOx\_BSRR) for bitwise write access to GPIOx\_ODR
- Locking mechanism (GPIOx\_LCKR) provided to freeze the I/O configuration
- Analog function
- Alternate function input/output selection registers (at most 16 AFs per I/O)
- Fast toggle capable of changing every two clock cycles
- Highly flexible pin multiplexing allows the use of I/O pins as GPIOs or as one of several peripheral functions

### 7.3 GPIO functional description

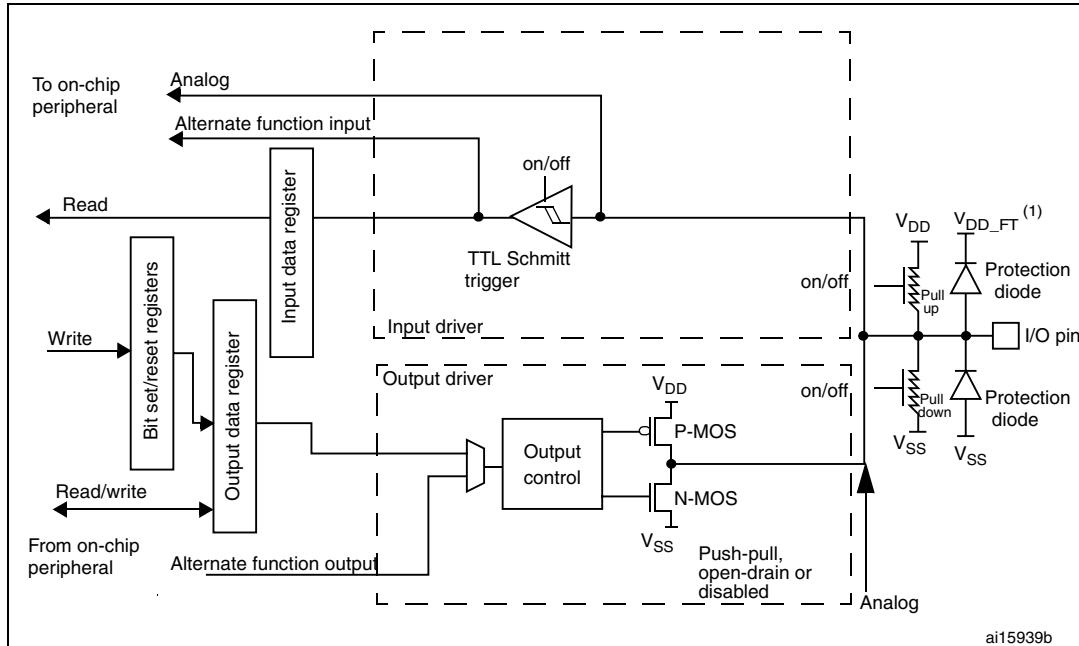
Subject to the specific hardware characteristics of each I/O port listed in the datasheet, each port bit of the general-purpose I/O (GPIO) ports can be individually configured by software in several modes:

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

Each I/O port bit is freely programmable, however the I/O port registers have to be accessed as 32-bit words, half-words or bytes. The purpose of the GPIOx\_BSRR register is to allow atomic read/modify accesses to any of the GPIO registers. In this way, there is no risk of an IRQ occurring between the read and the modify access.

*Figure 17* shows the basic structure of a 5 V tolerant I/O port bit, *Table 22* gives the possible port bit configurations.

**Figure 17. Basic structure of a 5 V tolerant I/O port bit**



ai15939b

1.  $V_{DD\_FT}$  is a potential specific to 5 V tolerant I/Os and different from  $V_{DD}$ .

**Table 22. Port bit configuration table<sup>(1)</sup>**

MODER(i)[1:0]	OTYPER(i)	OSPEEDR(i)[B:A]	PUPDR(i)[1:0]	I/O configuration
01	0	SPEED[B:A]	0	GP output
	0		0	PP
	0		1	GP output
	0		1	PP + PU
	1		0	GP output
	1		1	PP + PD
	1		1	Reserved
	1		0	GP output
10	0	SPEED[B:A]	0	AF
	0		0	PP
	0		1	AF
	0		1	PP + PU
	1		0	AF
	1		0	PP + PD
	1		1	Reserved
	1		1	OD

**Table 22. Port bit configuration table<sup>(1)</sup> (continued)**

MODER(i)[1:0]	OTYPER(i)	OSPEEDR(i)[B:A]	PUPDR(i)[1:0]	I/O configuration	
00	x	x	x	0	0
	x	x	x	0	1
	x	x	x	1	0
	x	x	x	1	1
11	x	x	x	0	0
	x	x	x	0	1
	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.

### 7.3.1 General-purpose I/O (GPIO)

During and just after reset, the alternate functions are not active and the I/O ports are configured in input floating mode.

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 N-MOS is activated when 0 is output).

The input data register (GPIOx\_IDR) captures the data present on the I/O pin at every AHB1 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.

### 7.3.2 I/O pin multiplexer and mapping

The microcontroller I/O pins are connected to onboard peripherals/modules through a multiplexer that allows only one peripheral's alternate function (AF) connected to an I/O pin at a time. In this way, there can be no conflict between peripherals sharing the same I/O pin.

Each I/O pin has a multiplexer with 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 all I/Os are connected to the system's alternate function 0 (AF0)
- The peripherals' alternate functions are mapped from AF1 to AF13
- Cortex®-M4 with FPU EVENTOUT is mapped on AF15

This structure is shown in [Figure 18](#).

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, proceed as follows:

- **System function**

Connect the I/O to AF0 and configure it depending on the function used:

- JTAG/SWD, after each device reset these pins are assigned as dedicated pins immediately usable by the debugger host (not controlled by the GPIO controller)
- RTC\_REFIN: this pin should be configured in Input floating mode
- MCO1 and MCO2: these pins have to be configured in alternate function mode.

**Note:** You can disable some or all of the JTAG/SWD pins and so release the associated pins for GPIO usage.

For more details refer to [Section 6.2.10: Clock-out capability](#).

**Table 23. Flexible SWJ-DP pin assignment**

Available debug ports	SWJ I/O 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

- **GPIO**

Configure the desired I/O as output or input in the GPIOx\_MODER register.

- **Peripheral alternate function**

For the ADC and DAC, configure the desired I/O as analog in the GPIOx\_MODER register.

For other peripherals:

- Configure the desired I/O as an alternate function in the GPIOx\_MODER register
- Select the type, pull-up/pull-down and output speed via the GPIOx\_OTYPER, GPIOx\_PUPDR and GPIOx\_OSPEEDER registers, respectively
- Connect the I/O to the desired AFx in the GPIOx\_AFRL or GPIOx\_AFRH register

- **EVENTOUT**

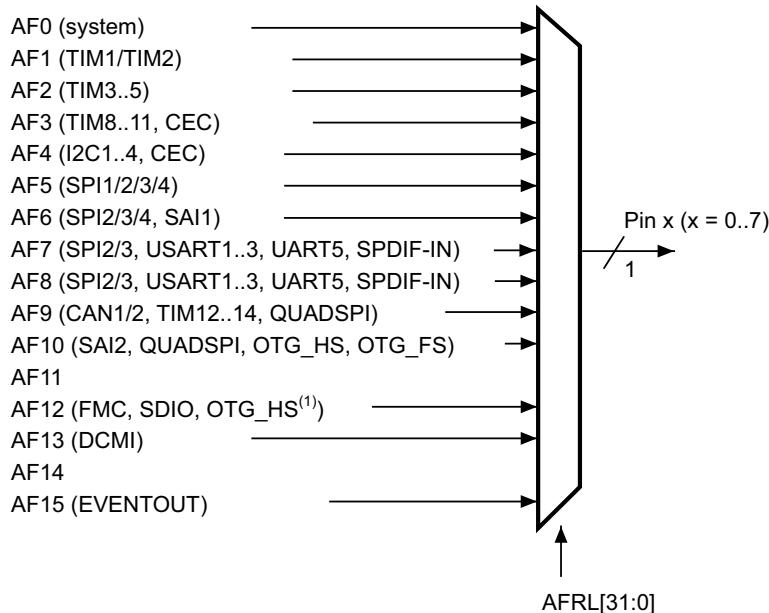
Configure the I/O pin used to output the Cortex®-M4 with FPU EVENTOUT signal by connecting it to AF15

**Note:** EVENTOUT is not mapped onto the following I/O pins: PC13, PC14, PC15, PH0 and PH1.

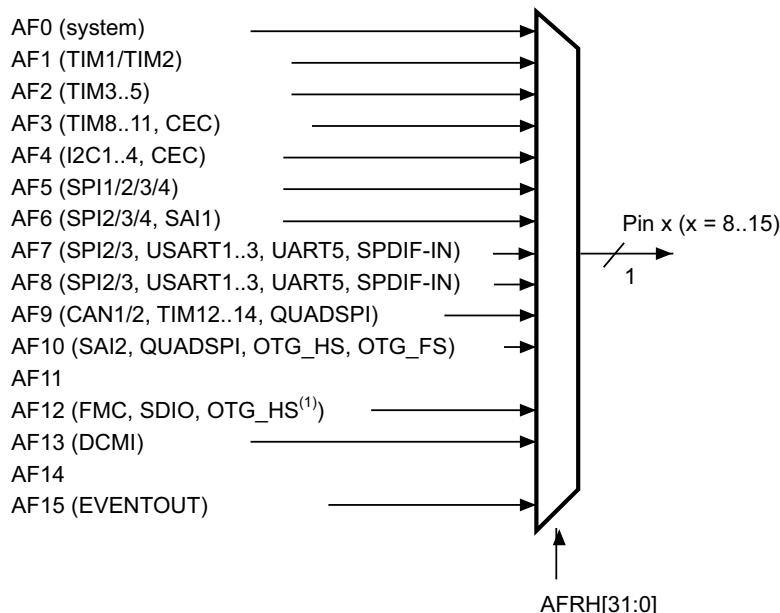
Refer to the “Alternate function mapping” table in the datasheets for the detailed mapping of the system and peripherals’ alternate function I/O pins.

**Figure 18. Selecting an alternate function on STM32F446xx**

For pins 0 to 7, the GPIOx\_AFRL[31:0] register selects the dedicated alternate function



For pins 8 to 15, the GPIOx\_AFRH[31:0] register selects the dedicated alternate function



MS35850V1

1. Configured in FS.

### 7.3.3 I/O port control registers

Each of the GPIOs 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 direction (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 I/O speed pins are directly connected to the corresponding GPIOx\_OSPEEDR register bits whatever the I/O direction). The GPIOx\_PUPDR register is used to select the pull-up/pull-down whatever the I/O direction.

### 7.3.4 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 7.4.5: GPIO port input data register \(GPIOx\\_IDR\) \(x = A..H\)](#) and [Section 7.4.6: GPIO port output data register \(GPIOx\\_ODR\) \(x = A..H\)](#) for the register descriptions.

### 7.3.5 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: BSRR(i) and BSRR(i+SIZE). When written to 1, bit BSRR(i) sets the corresponding ODR(i) bit. When written to 1, bit BSRR(i+SIZE) 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 AHB1 write access.

### 7.3.6 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 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 7.4.8: GPIO port configuration lock register \(GPIOx\\_LCKR\) \(x = A..H\)](#)) 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 7.4.8: GPIO port configuration lock register \(GPIOx\\_LCKR\) \(x = A..H\)](#).

### 7.3.7 I/O alternate function input/output

Two registers are provided to select one out of the sixteen alternate function inputs/outputs available for each I/O. With these registers, you can connect an alternate function to some other pin as required by your 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 one I/O.

To know which functions are multiplexed on each GPIO pin, refer to the datasheets.

**Note:** *The application is allowed to select one of the possible peripheral functions for each I/O at a time.*

### 7.3.8 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 10.2: External interrupt/event controller \(EXTI\)](#) and [Section 10.2.3: Wakeup event management](#).

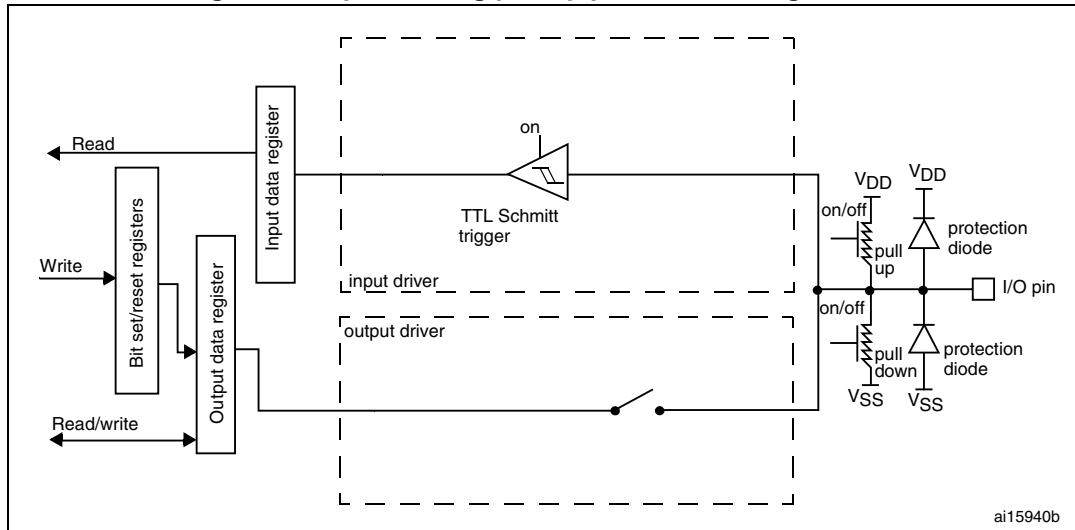
### 7.3.9 Input configuration

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 AHB1 clock cycle
- A read access to the input data register provides the I/O State

[Figure 19](#) shows the input configuration of the I/O port bit.

Figure 19. Input floating/pull up/pull down configurations



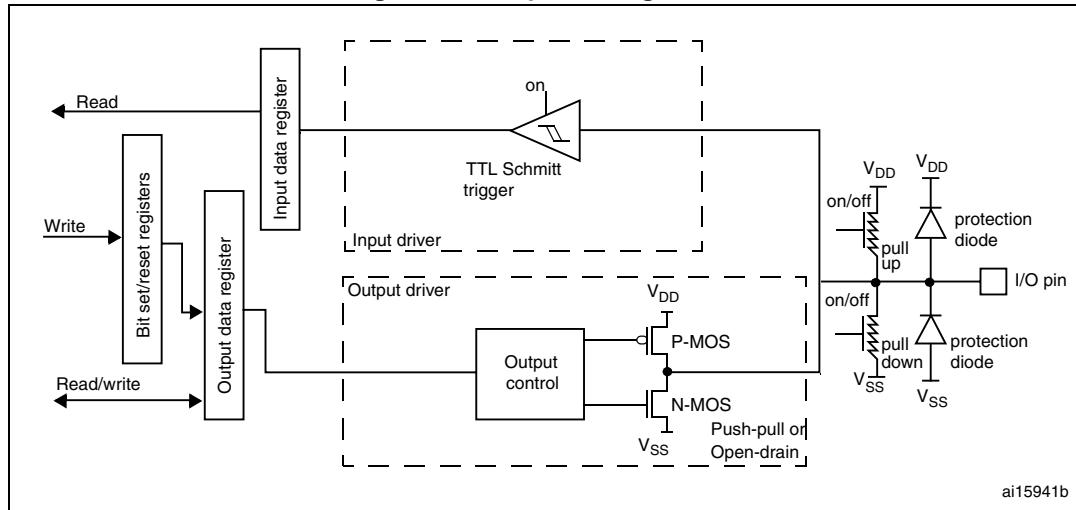
### 7.3.10 Output configuration

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 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 AHB1 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 20* shows the output configuration of the I/O port bit.

Figure 20. Output configuration



ai15941b

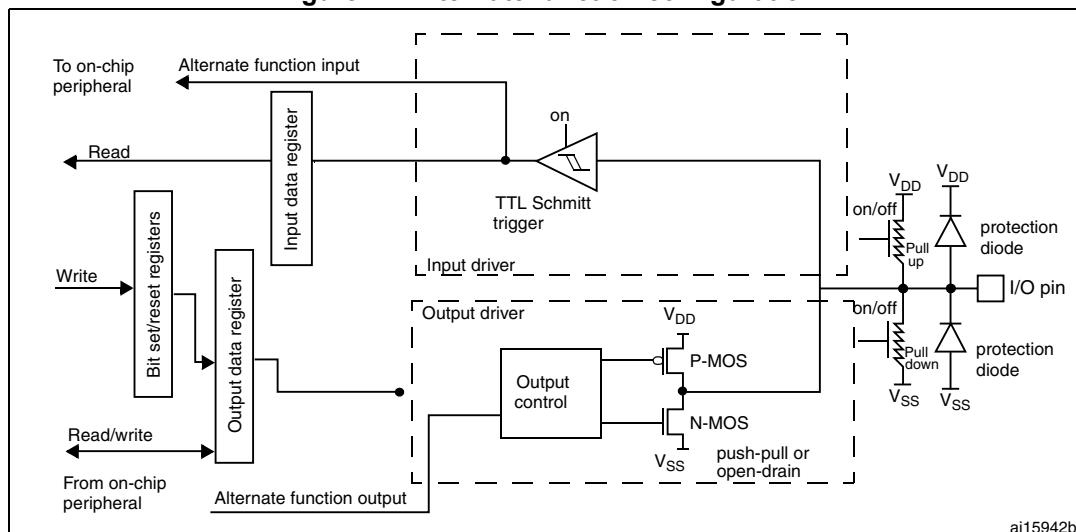
### 7.3.11 Alternate function configuration

When the I/O port is programmed as alternate function:

- The output buffer can be configured as open-drain or push-pull
- The output buffer is driven by the signal 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 AHB1 clock cycle
- A read access to the input data register gets the I/O state

*Figure 21* shows the Alternate function configuration of the I/O port bit.

Figure 21. Alternate function configuration



ai15942b

### 7.3.12 Analog configuration

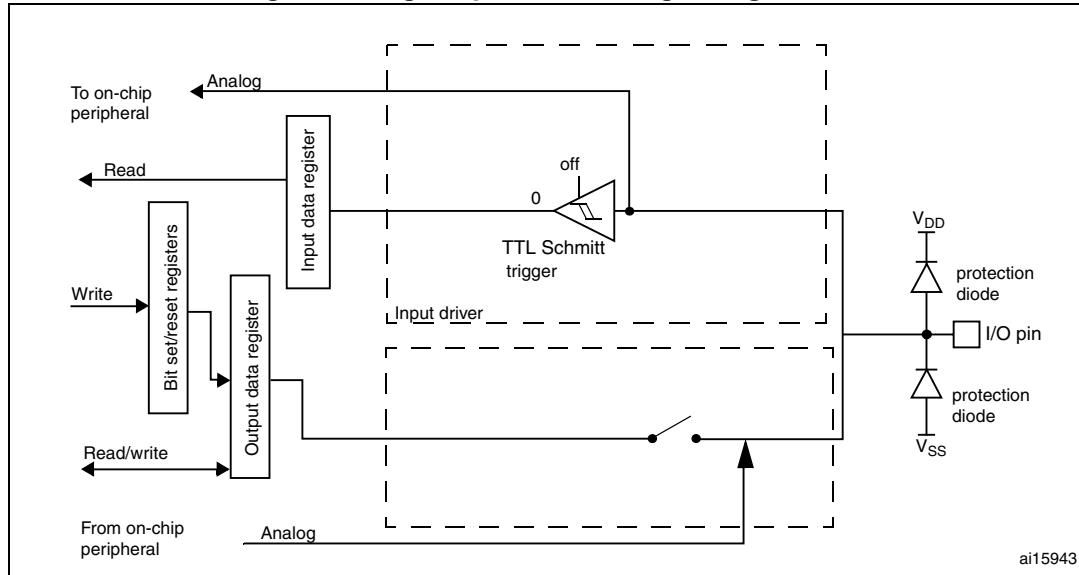
When the I/O port is programmed as analog configuration:

- 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
- Read access to the input data register gets the value “0”

*Note:* In the analog configuration, the I/O pins cannot be 5 Volt tolerant.

*Figure 22* shows the high-impedance, analog-input configuration of the I/O port bit.

**Figure 22. High impedance-analog configuration**



### 7.3.13 Using the OSC32\_IN/OSC32\_OUT pins as GPIO PC14/PC15 port pins

The LSE oscillator pins OSC32\_IN and OSC32\_OUT can be used as general-purpose PC14 and PC15 I/Os, respectively, when the LSE oscillator is off. The PC14 and PC15 I/Os are only configured as LSE oscillator pins OSC32\_IN and OSC32\_OUT when the LSE oscillator is ON. This is done by setting the LSEON bit in the RCC\_BDCR register. The LSE has priority over the GPIO function.

*Note:* The PC14/PC15 GPIO functionality is lost when the 1.2 V domain is powered off (by the device entering the standby mode) or when the backup domain is supplied by  $V_{BAT}$  ( $V_{DD}$  no more supplied). In this case the I/Os are set in analog input mode.

### 7.3.14 Using the OSC\_IN/OSC\_OUT pins as GPIO PH0/PH1 port pins

The HSE oscillator pins OSC\_IN/OSC\_OUT can be used as general-purpose PH0/PH1 I/Os, respectively, when the HSE oscillator is OFF. (after reset, the HSE oscillator is off). The PH0/PH1 I/Os are only configured as OSC\_IN/OSC\_OUT HSE oscillator pins when the HSE oscillator is ON. This is done by setting the HSEON bit in the RCC\_CR register. The HSE has priority over the GPIO function.

### 7.3.15 Selection of RTC additional\_AF1 and RTC\_AF2 alternate functions

The STM32F446xx feature two GPIO pins RTC\_AF1 and RTC\_AF2 that can be used for the detection of a tamper or time stamp event, or RTC\_ALARM, or RTC\_CALIB RTC outputs.

- The RTC\_AF1 (PC13) can be used for the following purposes:

RTC\_ALARM output: this output can be RTC Alarm A, RTC Alarm B or RTC Wakeup depending on the OSEL[1:0] bits in the RTC\_CR register

- RTC\_CALIB output: this feature is enabled by setting the COE[23] in the RTC\_CR register
- RTC\_TAMP1: tamper event detection
- RTC\_TS: time stamp event detection

The RTC\_AF2 (PA0) can be used for the following purposes:

- RTC\_TAMP1: tamper event detection
- RTC\_TAMP2: tamper event detection
- RTC\_TS: time stamp event detection

The selection of the corresponding pin is performed through the RTC\_TAFCR register as follows:

- TAMP1INSEL is used to select which pin is used as the RTC\_TAMP1 tamper input
- TSINSEL is used to select which pin is used as the RTC\_TS time stamp input
- ALARMOUTTYPE is used to select whether the RTC\_ALARM is output in push-pull or open-drain mode

The output mechanism follows the priority order listed in [Table 24](#) and [Table 25](#).

**Table 24. RTC\_AF1 pin<sup>(1)</sup>**

Pin configuration and function	RTC_ALARM enabled	RTC_CALIB enabled	Tamper enabled	Time stamp enabled	TAMP1INSEL TAMPER1 pin selection	TSINSEL TIMESTAMP pin selection	ALARMOUTTYPE RTC_ALARM configuration
Alarm out output OD	1	Don't care	Don't care	Don't care	Don't care	Don't care	0
Alarm out output PP	1	Don't care	Don't care	Don't care	Don't care	Don't care	1
Calibration out output PP	0	1	Don't care	Don't care	Don't care	Don't care	Don't care
TAMPER1 input floating	0	0	1	0	0	Don't care	Don't care
TIMESTAMP and TAMPER1 input floating	0	0	1	1	0	0	Don't care
TIMESTAMP input floating	0	0	0	1	Don't care	0	Don't care
Standard GPIO	0	0	0	0	Don't care	Don't care	Don't care

1. OD: open drain; PP: push-pull.

**Table 25. RTC\_AF2 pin**

Pin configuration and function	Tamper enabled	Time stamp enabled	TAMP1INSEL TAMPER1 pin selection	TSINSEL TIMESTAMP pin selection	ALARMOUTTYPE RTC_ALARM configuration
TAMPER1 input floating	1	0	1	Don't care	Don't care
TIMESTAMP and TAMPER1 input floating	1	1	1	1	Don't care
TIMESTAMP input floating	0	1	Don't care	1	Don't care
Standard GPIO	0	0	Don't care	Don't care	Don't care

## 7.4 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 26](#).

The GPIO registers can be accessed by byte (8 bits), half-words (16 bits) or words (32 bits).

### 7.4.1 GPIO port mode register (GPIOx\_MODER) (x = A..H)

Address offset: 0x00

Reset values:

- 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 2y:2y+1 **MODERy[1:0]**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

00: Input (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

### 7.4.2 GPIO port output type register (GPIOx\_OTYPER) (x = A..H)

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 OTy: Port x configuration bits (y = 0..15)

These bits are written by software to configure the output type of the I/O port.

0: Output push-pull (reset state)

1: Output open-drain

### 7.4.3 GPIO port output speed register (GPIOx\_OSPEEDR) (x = A..H)

Address offset: 0x08

Reset values:

- 0x0000 00C0 for port B
- 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 2y:2y+1 OSPEEDRy[1:0]: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O output speed.

00: Low speed

01: Medium speed

10: Fast speed

11: High speed

*Note:* Refer to the product datasheets for the values of OSPEEDRy bits versus  $V_{DD}$  range and external load.

#### 7.4.4 GPIO port pull-up/pull-down register (GPIOx\_PUPDR) (x = A..H)

Address offset: 0x0C

Reset values:

- 0x6400 0000 for port A
- 0x0000 0100 for port B
- 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	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
PUPDR7[1:0]	PUPDR6[1:0]	PUPDR5[1:0]	PUPDR4[1:0]	PUPDR3[1:0]	PUPDR2[1:0]	PUPDR1[1:0]	PUPDRO[1:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **PUPDR<sub>y</sub>[1:0]**: Port x configuration bits (y = 0..15)

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

#### 7.4.5 GPIO port input data register (GPIOx\_IDR) (x = A..H)

Address offset: 0x10

Reset value: 0x0000 XXXX (where X means undefined)

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<sub>y</sub>**: Port input data (y = 0..15)

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

### 7.4.6 GPIO port output data register (GPIOx\_ODR) (x = A..H)

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 **ODRy**: Port output data (y = 0..15)

These bits can be read and written by software.

*Note:* For atomic bit set/reset, the ODR bits can be individually set and reset by writing to the GPIOx\_BSRR register (x = A..H).

### 7.4.7 GPIO port bit set/reset register (GPIOx\_BSRR) (x = A..H)

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 **BRy**: Port x reset bit y (y = 0..15)

These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit  
1: Resets the corresponding ODRx bit

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

Bits 15:0 **BSy**: Port x set bit y (y = 0..15)

These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit  
1: Sets the corresponding ODRx bit

### 7.4.8 GPIO port configuration lock register (GPIOx\_LCKR) (x = A..H)

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 reset.

**Note:** A specific write sequence is used to write to the GPIOx\_LCKR register. Only word access (32-bit long) is allowed during this write sequence.

Each lock bit freezes a specific configuration register (control and alternate function registers).

Address offset: 0x1C

Reset value: 0x0000 0000

Access: 32-bit word only, read/write 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.	LCK K16
															rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LCK K15	LCK K14	LCK K13	LCK K12	LCK K11	LCK K10	LCK K9	LCK K8	LCK K7	LCK K6	LCK K5	LCK K4	LCK K3	LCK K2	LCK K1	LCK K0
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[16]: 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 an MCU reset occurs.

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 CPU reset.*

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

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

0: Port configuration not locked

1: Port configuration locked

### 7.4.9 GPIO alternate function low register (GPIOx\_AFRL) (x = A..H)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRL7[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]			
rw	rw	rw	rw												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]			
rw	rw	rw	rw												

Bits 31:0 **AFRLy**: Alternate function selection for port x bit y (y = 0..7)

These bits are written by software to configure alternate function I/Os

AFRLy selection:

0000: AF0	1000: AF8
0001: AF1	1001: AF9
0010: AF2	1010: AF10
0011: AF3	1011: AF11
0100: AF4	1100: AF12
0101: AF5	1101: AF13
0110: AF6	1110: AF14
0111: AF7	1111: AF15

### 7.4.10 GPIO alternate function high register (GPIOx\_AFRH) (x = A..H)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRH15[3:0]				AFRH14[3:0]				AFRH13[3:0]				AFRH12[3:0]			
rw	rw	rw	rw												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRH11[3:0]				AFRH10[3:0]				AFRH9[3:0]				AFRH8[3:0]			
rw	rw	rw	rw												

Bits 31:0 **AFRHy**: Alternate function selection for port x bit y (y = 8..15)

These bits are written by software to configure alternate function I/Os

AFRHy selection:

0000: AF0	1000: AF8
0001: AF1	1001: AF9
0010: AF2	1010: AF10
0011: AF3	1011: AF11
0100: AF4	1100: AF12
0101: AF5	1101: AF13
0110: AF6	1110: AF14
0111: AF7	1111: AF15

### 7.4.11 GPIO register map

The following table gives the GPIO register map and the reset values.

**Table 26. GPIO register map and reset values**

Offset	Register	Reset value	31
0x00	GPIOA_MODER	0 MODER5[1:0] 0 MODER15[1:0]	30
0x00	GPIOB_MODER	0 MODER4[1:0] 0 MODER14[1:0]	29
0x00	GPIOx_MODER (where x = C..H)	0 MODER3[1:0] 0 MODER13[1:0]	28
0x04	GPIOx_OTYPER (where x = A..H)	0 MODER2[1:0] 0 MODER12[1:0]	27
0x08	GPIOx_OSPEEDER (where x = A..H except B)	0 MODER1[1:0] 0 MODER11[1:0]	26
0x08	GPIOB_OSPEEDER	0 MODER0[1:0] 0 MODER10[1:0]	25
0x0C	GPIOA_PUPDR	0 PUPDR15[1:0] 0 OSPEEDR15[1:0]	24
0x0C	GPIOB_PUPDR	0 PUPDR15[1:0] 0 OSPEEDR15[1:0]	23
0x0C	Reset value	0 PUPDR14[1:0] 0 OSPEEDR14[1:0]	22
0x0C	Reset value	0 PUPDR13[1:0] 0 OSPEEDR13[1:0]	21
0x0C	Reset value	0 PUPDR12[1:0] 0 OSPEEDR12[1:0]	20
0x0C	Reset value	0 PUPDR11[1:0] 0 OSPEEDR11[1:0]	19
0x0C	Reset value	0 PUPDR10[1:0] 0 OSPEEDR10[1:0]	18
0x0C	Reset value	0 PUPDR9[1:0] 0 OSPEEDR9[1:0]	17
0x0C	Reset value	0 PUPDR8[1:0] 0 OSPEEDR8[1:0]	16
0x0C	Reset value	0 PUPDR7[1:0] 0 OSPEEDR7[1:0]	15
0x0C	Reset value	0 PUPDR6[1:0] 0 OSPEEDR6[1:0]	14
0x0C	Reset value	0 PUPDR5[1:0] 0 OSPEEDR5[1:0]	13
0x0C	Reset value	0 PUPDR4[1:0] 0 OSPEEDR4[1:0]	12
0x0C	Reset value	0 PUPDR3[1:0] 1 OSPEEDR3[1:0]	11
0x0C	Reset value	0 PUPDR2[1:0] 0 OSPEEDR2[1:0]	10
0x0C	Reset value	0 PUPDR1[1:0] 0 OSPEEDR1[1:0]	9
0x0C	Reset value	0 PUPDR0[1:0] 0 OSPEEDR0[1:0]	8

**Table 26. GPIO 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						
	GPIOx_PUPDR (where x = C..H)	PUPDR15[1:0]	PUPDR15[1:0]	PUPDR14[1:0]	PUPDR14[1:0]	PUPDR13[1:0]	PUPDR13[1:0]	PUPDR12[1:0]	PUPDR12[1:0]	PUPDR11[1:0]	PUPDR11[1:0]	PUPDR10[1:0]	PUPDR10[1:0]	PUPDR9[1:0]	PUPDR9[1:0]	PUPDR8[1:0]	PUPDR8[1:0]	PUPDR7[1:0]	PUPDR7[1:0]	PUPDR6[1:0]	PUPDR6[1:0]	PUPDR5[1:0]	PUPDR5[1:0]	PUPDR4[1:0]	PUPDR4[1:0]	PUPDR3[1:0]	PUPDR3[1:0]	PUPDR2[1:0]	PUPDR2[1:0]	PUPDR1[1:0]	PUPDR1[1:0]	PUPDR0[1:0]	PUPDR0[1:0]						
0x0C	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								
	GPIOx_IDR (where x = A..H)	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res																				
0x10	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								
	GPIOx_ODR (where x = A..H)	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res																				
0x14	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								
	GPIOx_BSRR (where x = A..H)	BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0	LCKK	BS15	O	ODR15	x	IDR15	0	PUPDR7[1:0]	PUPDR7[1:0]	PUPDR6[1:0]	PUPDR6[1:0]	PUPDR5[1:0]	PUPDR5[1:0]	PUPDR4[1:0]	PUPDR4[1:0]	PUPDR3[1:0]	PUPDR3[1:0]	PUPDR2[1:0]	PUPDR2[1:0]	PUPDR1[1:0]	PUPDR1[1:0]	PUPDR0[1:0]
0x18	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								
	GPIOx_LCKR (where x = A..H)	Res	Res	Res	Res	Res	0	0	0	0	0	0	0	0	0	0	0	0	0	0																			
0x1C	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								
	GPIOx_AFRL (where x = A..H)	AFRL7[3:0]	AFRL6[3:0]	AFRL5[3:0]	AFRL4[3:0]	AFRL3[3:0]	AFRL2[3:0]	AFRL1[3:0]	AFRL0[3:0]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0									
0x20	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								
	GPIOx_AFRH (where x = A..H)	AFRH15[3:0]	AFRH14[3:0]	AFRH13[3:0]	AFRH12[3:0]	AFRH11[3:0]	AFRH10[3:0]	AFRH9[3:0]	AFRH8[3:0]	AFRH7[3:0]	AFRH6[3:0]	AFRH5[3:0]	AFRH4[3:0]	AFRH3[3:0]	AFRH2[3:0]	AFRH1[3:0]	AFRH0[3:0]	AFRL7[3:0]	AFRL6[3:0]	AFRL5[3:0]	AFRL4[3:0]	AFRL3[3:0]	AFRL2[3:0]	AFRL1[3:0]	AFRL0[3:0]	AFRH7[3:0]	AFRH6[3:0]	AFRH5[3:0]	AFRH4[3:0]	AFRH3[3:0]	AFRH2[3:0]	AFRH1[3:0]	AFRH0[3:0]						
0x24	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								
	GPIOx_AFRL (where x = A..H)	AFRL7[3:0]	AFRL6[3:0]	AFRL5[3:0]	AFRL4[3:0]	AFRL3[3:0]	AFRL2[3:0]	AFRL1[3:0]	AFRL0[3:0]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0									
0x28	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								
	GPIOx_AFRH (where x = A..H)	BS3	BS2	BS1	BS0	LCK3	LCK2	LCK1	LCK0	BS3	BS2	BS1	BS0	LCK3	LCK2	LCK1	LCK0	BS3	BS2	BS1	BS0	LCK3	LCK2	LCK1	LCK0	BS3	BS2	BS1	BS0	LCK3	LCK2								

Refer to [Table 1](#) on page 56 for the register boundary addresses.

## 8 System configuration controller (SYSCFG)

The system configuration controller is mainly used to remap the memory accessible in the code area and to manage the external interrupt line connection to the GPIOs.

### 8.1 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.

### 8.2 SYSCFG registers

#### 8.2.1 SYSCFG memory remap register (SYSCFG\_MEMRMP)

This register is used for specific configurations on memory remap:

- Three bits are used to configure the type of memory accessible at address 0x0000 0000. These bits are used to select the physical remap by software and so, bypass the BOOT pins.
- After reset these bits take the value selected by the BOOT pins. When booting from main Flash memory with BOOT pins set to 10 [(BOOT1,BOOT0) = (1,0)] this register takes the value 0x00.
- Other bits are used to swap FMC SDRAM Bank 1/2 with FMC Bank 3/4.

There are two possible FMC remap at address 0x0000 0000:

- FMC Bank 1 (NOR/PSRAM 1 and 2) remap:  
Only the first two regions of Bank 1 memory controller (Bank1 NOR/PSRAM 1 and NOR/PSRAM 2) can be remapped.
- FMC SDRAM Bank 1 remap.

In remap mode at address 0x0000 0000, the CPU can access the external memory via ICode bus instead of System bus which boosts up the performance.

Address offset: 0x00

Reset value: 0x0000 000X (X is the memory mode selected by the BOOT pins)

*Note:* *Booting from NOR Flash memory or SDRAM is not allowed. The regions can only be mapped at 0x0000 0000 through software remap.*

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.	<b>SWP_FMC</b>		Res.	<b>MEM_MODE[2:0]</b>								
				rw	rw								rw	rw	rw

Bits 31:12 Reserved, must be kept at reset value.

Bits 11:10 **SWP\_FMC**: FMC memory mapping swap

Set and cleared by software. These bits are used to swap the FMC SDRAM Banks 1/2 from address 0xC000 0000 and 0xD000 0000 to address 0x6000 0000 and 0x7000 0000 in order to enable the code execution from SDRAM Banks without a physical remapping at 0x0000 0000 address. NOR/PSRAM Bank which is by default mapped at 0x6000 0000 is remapped at 0xC000 0000 when SDRAM bank1 is mapped at 0x6000 0000.

00: No FMC memory mapping swap

01: SDRAM banks mapping are swapped. SDRAM Bank 1 and 2 are mapped at 0x6000 0000 and 0x7000 0000 address, respectively. NOR/PSRAM Bank is mapped at 0xC000 0000.

10: Reserved

11: Reserved

Bits 9:3 Reserved, must be kept at reset value.

Bits 2:0 **MEM\_MODE**: Memory mapping selection

Set and cleared by software. This bit controls the memory internal mapping at address 0x0000 0000. After reset these bits take the value selected by the Boot pins (except for FMC).

000: Main Flash memory mapped at 0x0000 0000

001: System Flash memory mapped at 0x0000 0000

010: FMC Bank1 (NOR/PSRAM 1 and 2) mapped at 0x0000 0000

011: Embedded SRAM (SRAM1) mapped at 0x0000 0000

100: FMC/SDRAM Bank 1 mapped at 0x0000 0000

Other configurations are reserved

*Note:* Refer to [Section 2.2.2: Memory map and register boundary addresses](#) for details about the memory mapping at address 0x0000 0000.

## 8.2.2 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.	ADCxDC2														
															rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.															

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

Bits 18:16 ADCxDC2:

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 15:0 Reserved, must be kept at reset value.

## 8.2.3 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.

*Note:*

- 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

#### 8.2.4 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)

*Note:* 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

### 8.2.5 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.

*Note:* 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

### 8.2.6 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.

**Note:**

- 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: PG[x] pin

## 8.2.7 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.							
								r							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	READY	Res.	CMP_PD												
							r								rw

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 7:1 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

## 8.2.8 SYSCFG configuration register (SYSCFG\_CFGR)

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.														
								r							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	FMPI2C1_SDA	FMPI2C1_SCL													
														rw	rw

Bits 31:2 Reserved, must be kept at reset value.

Bit 1 **FMPI2C1\_SDA**

Set and cleared by software. When set it forces FM+ drive capability on FMPI2C1\_SDA pin selected through GPIO port mode register and GPIO alternate function selection bits

Bit 0 **FMPI2C1\_SCL**

Set and cleared by software. When set it forces FM+ drive capability on FMPI2C1\_SCL pin selected through GPIO port mode register and GPIO alternate function selection bits

## 8.2.9 SYSCFG register maps

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

**Table 27. 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	Reserved	MEM_MODE																														
	Reset value																											x	x	x			
0x04	SYSCFG_PMC	Reserved																															
	Reset value																																
0x08	SYSCFG_EXTICR1	Reserved	EXTI3[3:0]																														
	Reset value																																
0x0C	SYSCFG_EXTICR2	Reserved	EXTI10[3:0]																														
	Reset value																																
0x10	SYSCFG_EXTICR3	Reserved	EXTI7[3:0]																														
	Reset value																																
0x14	SYSCFG_EXTICR4	Reserved	EXTI11[3:0]																														
	Reset value																																
0x20	SYSCFG_CMPCR	Reserved	EXTI15[3:0]																														
	Reset value																																
0x2C	SYSCFG_CFGR	Reserved	READY	Reserved	EXTI12[3:0]																												
	Reset value																	0	Reserved	Reserved	Reserved	EXTI8[3:0]											

Refer to [Table 1 on page 56](#) for the register boundary addresses.

## 9 Direct memory access controller (DMA)

### 9.1 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 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. And each has an arbiter for handling the priority between DMA requests.

### 9.2 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 8 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 by hardware 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
- Each of the 8 streams are connected to dedicated hardware DMA channels (requests)
- Priorities between DMA stream requests are software-programmable (4 levels consisting of very high, high, medium, low) or hardware in case of equality (request 0 has priority over request 1, etc.)

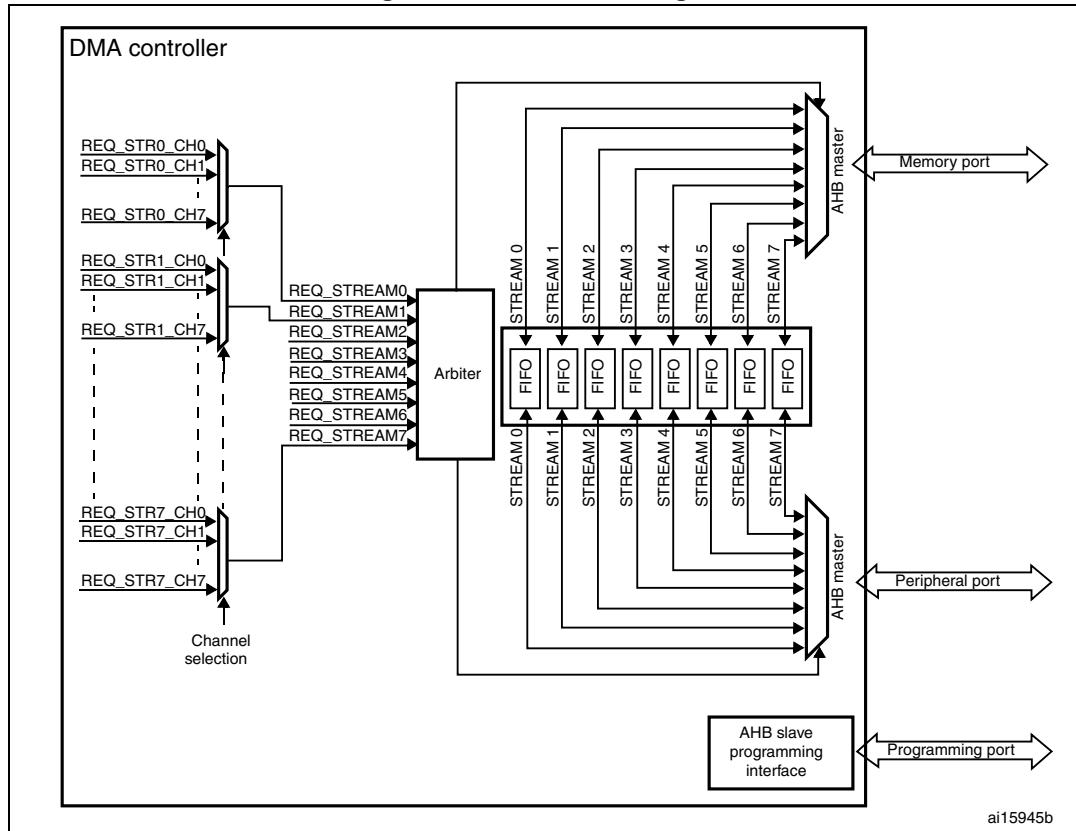
- Each stream also supports software trigger for memory-to-memory transfers (only available for the DMA2 controller)
- Each stream request can be selected via among up to 8 possible channel requests. This selection is software-configurable and allows several peripherals to initiate DMA requests
- The 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

## 9.3 DMA functional description

### 9.3.1 DMA block diagram

*Figure 23* shows the block diagram of a DMA.

**Figure 23. DMA block diagram**



### 9.3.2 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 can carry 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).

### 9.3.3 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:

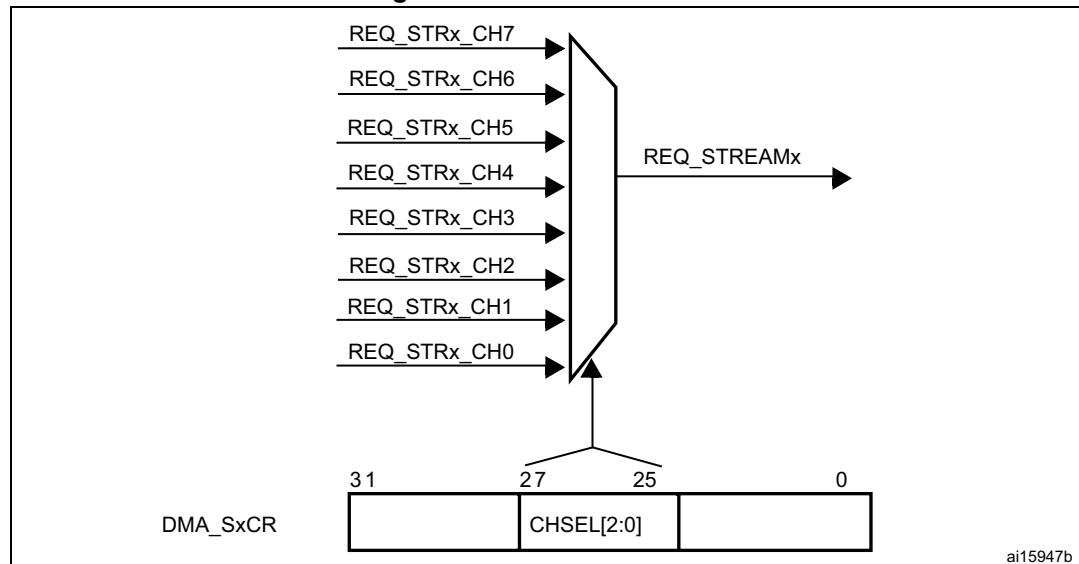
- A loading from the peripheral data register or a location in memory, addressed through the DMA\_SxPAR or DMA\_SxM0AR register
- A storage of the data loaded to the peripheral data register or a location in memory addressed through the DMA\_SxPAR or DMA\_SxM0AR register
- A post-decrement of the DMA\_SxNDTR register, which contains 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.

### 9.3.4 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 24. Channel selection**



ai15947b

The 8 requests from the peripherals (TIM, ADC, SPI, I2C, etc.) are independently connected to each channel and their connection depends on the product implementation.

[Table 28](#) and [Table 29](#) give examples of DMA request mappings.

**Table 28. DMA1 request mapping**

<b>Peripheral requests</b>	<b>Stream 0</b>	<b>Stream 1</b>	<b>Stream 2</b>	<b>Stream 3</b>	<b>Stream 4</b>	<b>Stream 5</b>	<b>Stream 6</b>	<b>Stream 7</b>
Channel 0	SPI3_RX	SPDIFRX_DT	SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX	SPDIFRX_CS	SPI3_TX
Channel 1	I2C1_RX	I2C3_RX	TIM7_UP	-	TIM7_UP	I2C1_RX	I2C1_TX	I2C1_TX
Channel 2	TIM4_CH1	-	FMPIC2C1_RX	TIM4_CH2	-	FMPIC2C1_RX	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	-	-	TIM3_CH4 TIM3_UP	-	TIM3_CH1 TIM3_TRIG	TIM3_CH2	-	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 29. DMA2 request mapping**

<b>Peripheral requests</b>	<b>Stream 0</b>	<b>Stream 1</b>	<b>Stream 2</b>	<b>Stream 3</b>	<b>Stream 4</b>	<b>Stream 5</b>	<b>Stream 6</b>	<b>Stream 7</b>
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	-	DCMI	ADC2	ADC2	SAI1_B	-	-	DCMI
Channel 2	ADC3	ADC3	-	-	-	-	-	-
Channel 3	SPI1_RX	-	SPI1_RX	SPI1_TX	SAI2_A	SPI1_TX	SAI2_B	QUADSPI
Channel 4	SPI4_RX	SPI4_TX	USART1_RX	SDIO		USART1_RX	SDIO	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	-	-	TIM8_CH4 TIM8_TRIG TIM8_COM

### 9.3.5 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.

### 9.3.6 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.

### 9.3.7 Source, destination and transfer modes

Both source and destination transfers can address peripherals and memories in the entire 4 GB 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 30* describes the corresponding source and destination addresses.

**Table 30. 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 25* 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.

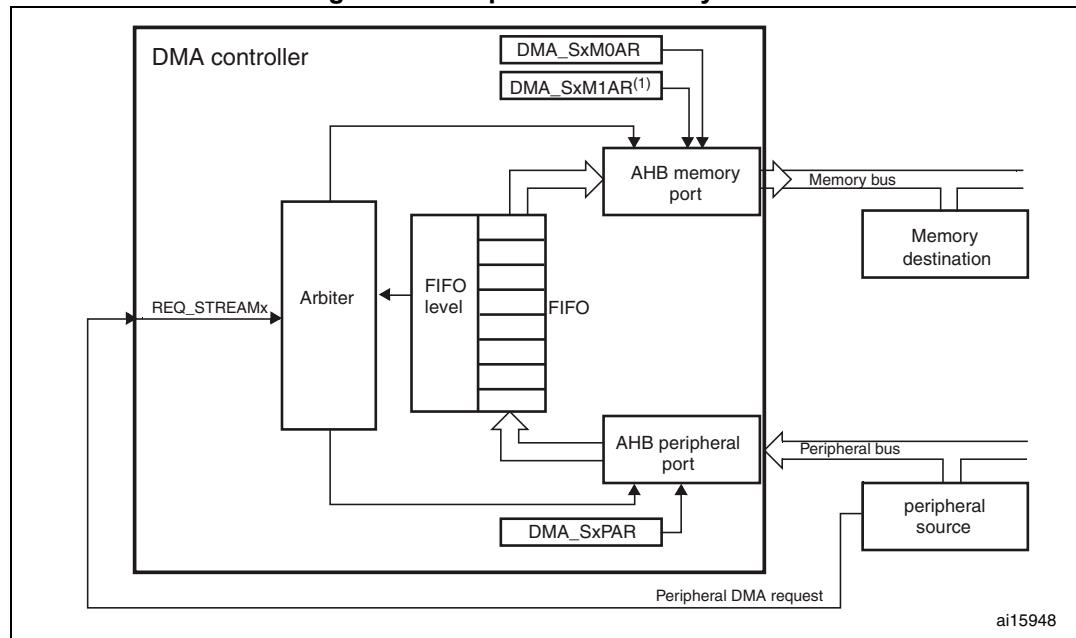
When the threshold level of the FIFO is reached, the contents of the FIFO are drained and stored into the destination.

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 25. Peripheral-to-memory mode**



1. For double-buffer mode.

### Memory-to-peripheral mode

*Figure 26* 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.

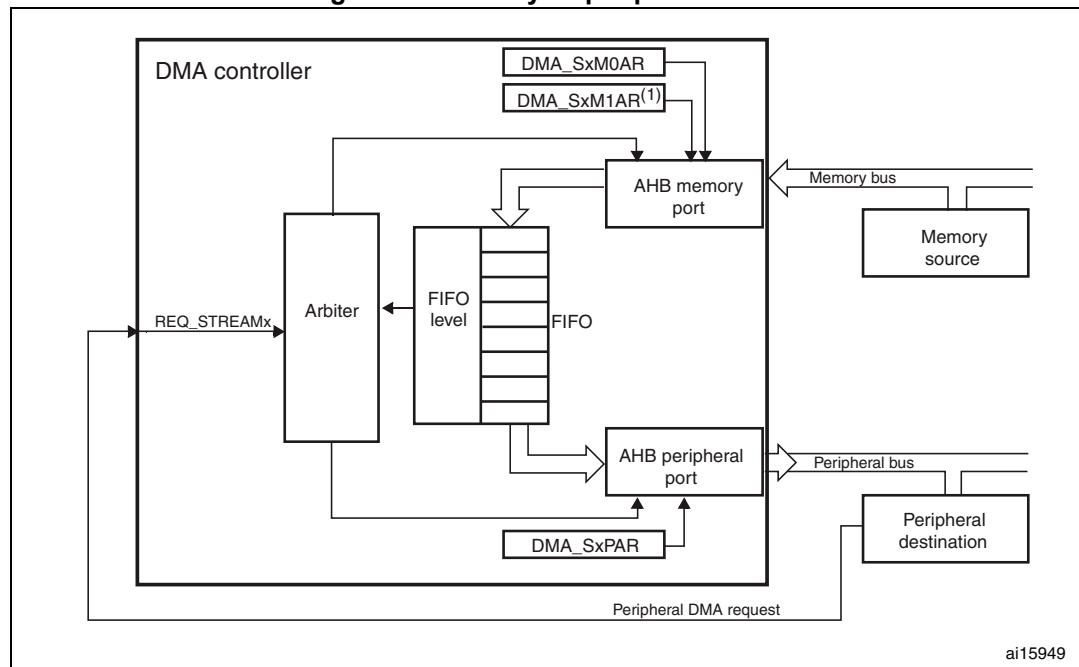
Each time a peripheral request occurs, the contents of the FIFO are drained and stored into the destination. When the level of the FIFO is lower than or equal to the predefined threshold level, the FIFO is fully reloaded with data from the memory.

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 transfer. 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 26. 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 27](#).

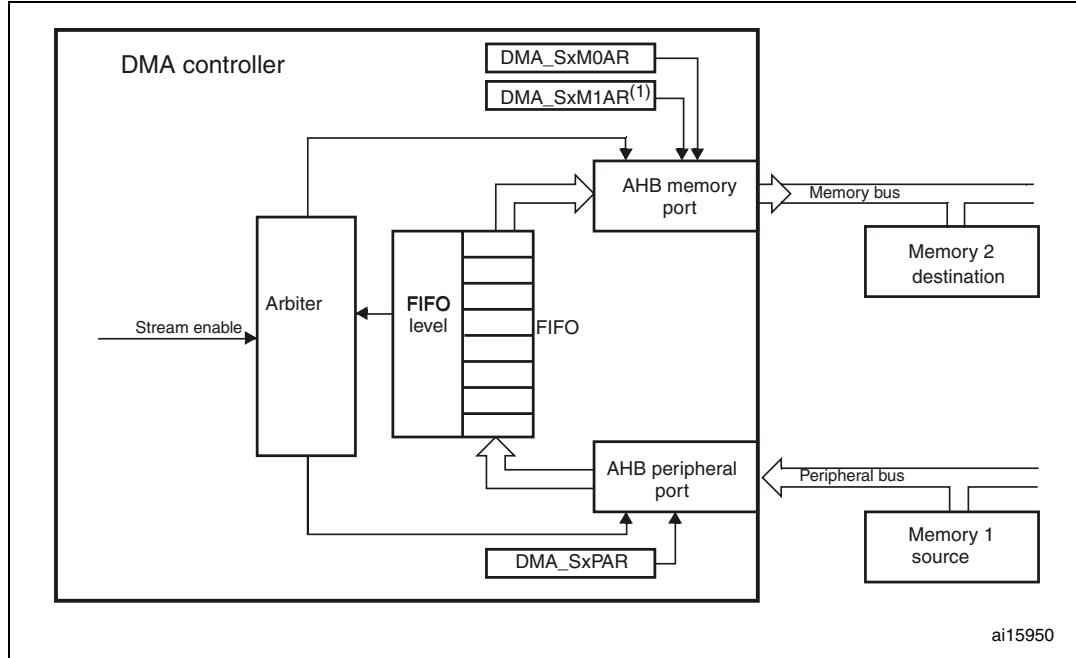
When the stream is enabled by setting the Enable bit (EN) in the DMA\_SxCR register, the stream immediately starts to fill the FIFO up to the threshold level. When the threshold level is reached, the FIFO contents are drained and stored into the destination.

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 27. Memory-to-memory mode



1. For double-buffer mode.

ai15950

### 9.3.8 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 are accessed through a single register.

If the Increment mode is enabled, the address of the next transfer will be 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.

### 9.3.9 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.

### 9.3.10 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 don't care) 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 31: 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 31. 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 <sup>(1)</sup>		
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.

### 9.3.11 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 32: Packing/unpacking & 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 9.3.12: Single and burst transfers](#)).

In direct mode (DMDIS = 0 in the DMA\_SxFCR 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 MSIZE bits are don't care).

Table 32. Packing/unpacking &amp; 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 2 3 4	0x0 / B0[7:0] 0x1 / B1[7:0] 0x2 / B2[7:0] 0x3 / B3[7:0]	1 2 3 4	0x0 / B0[7:0] 0x4 / B1[7:0] 0x8 / B2[7:0] 0xC / B3[7:0]	0x0 / B0[7:0] 0x1 / B1[7:0] 0x2 / B2[7:0] 0x3 / B3[7:0]
8	16	2	1 2 3 4	0x0 / B0[7:0] 0x1 / B1[7:0] 0x2 / B2[7:0] 0x3 / B3[7:0]	1 2	0x0 / B1 B0[15:0] 0x4 / B3 B2[15:0]	0x0 / B1 B0[15:0] 0x2 / B3 B2[15:0]
8	32	1	1 2 3 4	0x0 / B0[7:0] 0x1 / B1[7:0] 0x2 / B2[7:0] 0x3 / B3[7:0]	1	0x0 / B3 B2 B1 B0[31:0]	0x0 / B3 B2 B1 B0[31:0]
16	8	4	1 2	0x0 / B1 B0[15:0] 0x2 / B3 B2[15:0]	1 2 3 4	0x0 / B0[7:0] 0x4 / B1[7:0] 0x8 / B2[7:0] 0xC / B3[7:0]	0x0 / B0[7:0] 0x1 / B1[7:0] 0x2 / B2[7:0] 0x3 / B3[7:0]
16	16	2	1 2	0x0 / B1 B0[15:0] 0x2 / B1 B0[15:0]	1 2	0x0 / B1 B0[15:0] 0x4 / B3 B2[15:0]	0x0 / B1 B0[15:0] 0x2 / B3 B2[15:0]
16	32	1	1 2	0x0 / B1 B0[15:0] 0x2 / B3 B2[15:0]	1	0x0 / B3 B2 B1 B0[31:0]	0x0 / B3 B2 B1 B0[31:0]
32	8	4	1	0x0 / B3 B2 B1 B0[31:0]	1 2 3 4	0x0 / B0[7:0] 0x4 / B1[7:0] 0x8 / B2[7:0] 0xC / B3[7:0]	0x0 / B0[7:0] 0x1 / B1[7:0] 0x2 / B2[7:0] 0x3 / B3[7:0]
32	16	2	1	0x0 / B3 B2 B1 B0[31:0]	1 2	0x0 / B1 B0[15:0] 0x4 / B3 B2[15:0]	0x0 / B1 B0[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 33](#).

Table 33. 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

### 9.3.12 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 KB address boundary because the minimum address space that can be allocated to a single slave is 1 KB. This means that the 1 KB address boundary should not be crossed by a burst block transfer, otherwise an AHB error would be generated, that is not reported by the DMA registers.

### 9.3.13 FIFO

#### FIFO structure

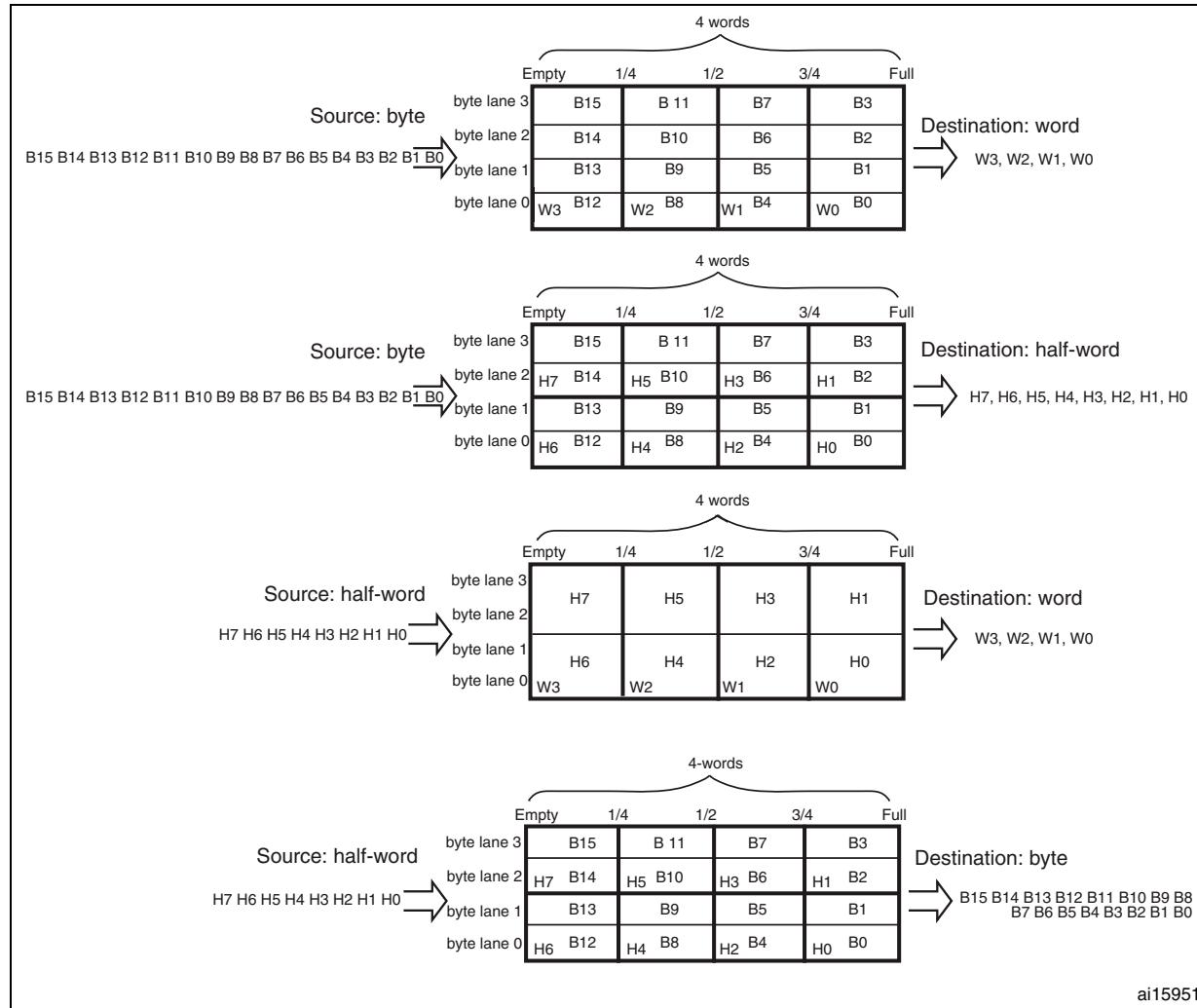
The FIFO is used to temporarily store data coming from the source before transmitting them to the destination.

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 28: FIFO structure](#).

Figure 28. 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 $\times$  of the DMA\_HISR or DMA\_LISR register) will be generated when the stream is enabled, then the stream will be automatically disabled. The allowed and forbidden configurations are described in [Table 34](#). The forbidden configurations are highlighted in gray in the table.

Table 34. 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 34. 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 will be managed in single mode by the DMA, even if a burst transaction was 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 will be 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 will be sent with the data width set in the MSIZE bit in the DMA\_SxCR register. This means that memory will be 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 will be generated to complete the FIFO flush.

### Direct mode

By default, the FIFO operates in direct mode (DMDIS bit in the DMA\_SxFIFO register 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 don't care)
- 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.

### 9.3.14 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).

### 9.3.15 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:** Note that 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.

### 9.3.16 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 which are able to signal the end of the transfer, that is:
  - SDIO

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 will not be 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.

### 9.3.17 Summary of the possible DMA configurations

[Table 35](#) summarizes the different possible DMA configurations. The forbidden configurations are highlighted in gray in the table.

**Table 35. 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		

### 9.3.18 Stream configuration procedure

The following sequence should 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 have 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 should be cleared before the stream can be re-enabled.
2. Set the peripheral port register address in the DMA\_SxPAR register. The data will be 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 will be 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.

---

### 9.3.19 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 9.3.10: 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 34: 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 are 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.*

## 9.4 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 36](#).

**Table 36. 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 to ‘1’, the corresponding event flag should be cleared, otherwise an interrupt is immediately generated.*

## 9.5 DMA registers

The DMA registers have to be accessed by words (32 bits).

### 9.5.1 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	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	r	r	r		r

Bits 31:28, 15:12 Reserved, must be kept at reset value.

Bits 27, 21, 11, 5 **TCIFx**: 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 **HTIFx**: 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 **TEIFx**: 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 **DMEIFx**: 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 **FEIFx**: 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

### 9.5.2 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 **TCIFx**: 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 **HTIFx**: 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 **TEIFx**: 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 **DMEIFx**: 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 **FEIFx**: 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

### 9.5.3 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 **CTCIFx**: 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 **CHTIFx**: 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 **CTEIFx**: 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 **CDMEIFx**: 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 **CFEIFx**: 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

### 9.5.4 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 **CTCIFx**: 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 **CHTIFx**: 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 **CTEIFx**: 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 **CDMEIFx**: Stream x clear direct mode error interrupt flag (x = 7..4)  
 Writing 1 to this bit clears the corresponding DMEIFx flag in the DMA\_HISR register

Bits 23, 17, 7, 1 Reserved, must be kept at reset value.

Bits 22, 16, 6, 0 **CFEIFx**: Stream x clear FIFO error interrupt flag (x = 7..4)  
 Writing 1 to this bit clears the corresponding CFEIFx flag in the DMA\_HISR register

### 9.5.5 DMA stream x configuration register (DMA\_SxCR) (x = 0..7)

This register is used to configure the concerned stream.

Address offset:  $0x10 + 0x18 \times \text{stream number}$

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	CHSEL[2:0]			MBURST [1:0]	PBURST[1:0]		Res.	CT	DBM	PL[1:0]		
				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
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 31:28 Reserved, must be kept at reset value.

Bits 27:25 **CHSEL[2:0]**: Channel selection

These bits are set and cleared by software.  
 000: channel 0 selected  
 001: channel 1 selected  
 010: channel 2 selected  
 011: channel 3 selected  
 100: channel 4 selected  
 101: channel 5 selected  
 110: channel 6 selected  
 111: channel 7 selected

These bits are protected and can be written only if EN is '0'

Bits 24:23 **MBURST**: 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 bits is set and cleared by hardware. It can also be written by software.

0: The current target memory is Memory 0 (addressed by the DMA\_SxM0AR pointer)

1: The 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 bits 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: The 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.*

### 9.5.6 DMA stream x number of data register (DMA\_SxNDTR) (x = 0..7)

Address offset:  $0x14 + 0x18 \times \text{stream number}$

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

Number of data items to be transferred (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 has 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.

### 9.5.7 DMA stream x peripheral address register (DMA\_SxPAR) (x = 0..7)

Address offset:  $0x18 + 0x18 \times \text{stream number}$

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 will be read/written.

These bits are write-protected and can be written only when bit EN = '0' in the DMA\_SxCR register.

### 9.5.8 DMA stream x memory 0 address register (DMA\_SxM0AR) (x = 0..7)

Address offset:  $0x1C + 0x18 \times \text{stream number}$

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 will be 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).

### 9.5.9 DMA stream x memory 1 address register (DMA\_SxM1AR) (x = 0..7)

Address offset:  $0x20 + 0x18 \times \text{stream number}$

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 will be 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.

### 9.5.10 DMA stream x FIFO control register (DMA\_SxFCR) (x = 0..7)

Address offset:  $0x24 + 0x24 \times \text{stream number}$

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.											
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 < fifo\_level < 1/4

001: 1/4 ≤ fifo\_level < 1/2

010: 1/2 ≤ fifo\_level < 3/4

011: 3/4 ≤ fifo\_level < 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'.

### 9.5.11 DMA register map

*Table 37* summarizes the DMA registers.

**Table 37. DMA register map and reset values**

**Table 37. DMA register map and reset values (continued)**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
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																													FTH[1:0]				
	Reset value																																		
0x0040	DMA_S2CR	Res																																	
	Reset value																															FS[2:0]	DMDIS	0	0
0x0044	DMA_S2NDTR	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								
	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	
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	
0x0054	DMA_S2FCR	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	FTH[1:0]							
	Reset value																																		
0x0058	DMA_S3CR	Res		Res	Res	Res	Res	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
	Reset value																																		
0x005C	DMA_S3NDTR	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	FTH[1:0]						
	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	
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	
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	

**Table 37. DMA register map and reset values (continued)**

**Table 37. DMA register map and reset values (continued)**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
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	
0x00B4	DMA_S6FCR	Res	FTH[1:0]																														
	Reset value																																
0x00B8	DMA_S7CR	Res	FS[2:0]																														
	Reset value																																
0x00BC	DMA_S7NDTR	Res	0	1																													
	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	
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	
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	
0x00CC	DMA_S7FCR	Res	FS[2:0]																														
	Reset value																																

Refer to [Section 2.2.2 on page 56](#) for the register boundary addresses.

## 10 Interrupts and events

### 10.1 Nested vectored interrupt controller (NVIC)

#### 10.1.1 NVIC features

The nested vector interrupt controller NVIC includes the following features:

- 96 maskable interrupt channels (not including the 16 interrupt lines of Cortex®-M4 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 PM0214.

#### 10.1.2 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).

#### 10.1.3 Interrupt and exception vectors

See [Table 38](#) for the vector table.

## 10.2 External interrupt/event controller (EXTI)

The external interrupt/event controller consists of up to 23 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.

**Table 38. Vector table for STM32F446xx**

Position	Priority	Type of priority	Acronym	Description	Address
-	-	-	-	Reserved	0x0000 0000
-	-3	fixed	Reset	Reset	0x0000 0004
-	-2	fixed	NMI	Non maskable interrupt, Clock Security System	0x0000 0008
-	-1	fixed	HardFault	All class of fault	0x0000 000C
-	0	settable	MemManage	Memory management	0x0000 0010

Table 38. Vector table for STM32F446xx (continued)

Position	Priority	Type of priority	Acronym	Description	Address
-	1	settable	BusFault	Pre-fetch fault, memory access fault	0x0000 0014
-	2	settable	UsageFault	Undefined instruction or illegal state	0x0000 0018
-	-	-	-	Reserved	0x0000 001C - 0x0000 002B
-	3	settable	SVCall	System Service call via SWI instruction	0x0000 002C
-	4	settable	Debug Monitor	Debug Monitor	0x0000 0030
-	-	-	-	Reserved	0x0000 0034
-	5	settable	PendSV	Pendable request for system service	0x0000 0038
-	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
22	29	settable	CAN1_SCE	CAN1 SCE interrupt	0x0000 0098
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000 009C

Table 38. Vector table for STM32F446xx (continued)

Position	Priority	Type of priority	Acronym	Description	Address
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 <sup>2</sup> C1 event interrupt	0x0000 00BC
32	39	settable	I2C1_ER	I <sup>2</sup> C1 error interrupt	0x0000 00C0
33	40	settable	I2C2_EV	I <sup>2</sup> C2 event interrupt	0x0000 00C4
34	41	settable	I2C2_ER	I <sup>2</sup> 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
47	54	settable	DMA1_Stream7	DMA1 Stream7 global interrupt	0x0000 00FC
48	55	settable	FMC	FMC global interrupt	0x0000 0100
49	56	settable	SDIO	SDIO 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

Table 38. Vector table for STM32F446xx (continued)

Position	Priority	Type of priority	Acronym	Description	Address
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	settable	CAN2_TX	CAN2 TX interrupts	0x0000 013C
64	71	settable	CAN2_RX0	CAN2 RX0 interrupts	0x0000 0140
65	72	settable	CAN2_RX1	CAN2 RX1 interrupt	0x0000 0144
66	73	settable	CAN2_SCE	CAN2 SCE interrupt	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 <sup>2</sup> C3 event interrupt	0x0000 0160
73	80	settable	I2C3_ER	I <sup>2</sup> 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
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	settable	DCMI	DCMI global interrupt	0x0000 0178
79	86	-	-	Reserved	0x0000 017C
80	87	-	-	Reserved	0x0000 0180
81	88	settable	FPU	FPU global interrupt	0x0000 0184
82	89	-	-	Reserved	0x0000 0188
83	90	-	-	Reserved	0x0000 018C

**Table 38. Vector table for STM32F446xx (continued)**

<b>Position</b>	<b>Priority</b>	<b>Type of priority</b>	<b>Acronym</b>	<b>Description</b>	<b>Address</b>
84	91	settable	SPI4	SPI 4 global interrupt	0x0000 0190
85	92	-	-	Reserved	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	HDMI-CEC	HDMI-CEC global interrupt	0x0000 01B4
94	101	settable	SPDIF-Rx	SPDIF-Rx global interrupt	0x0000 01B8
95	102	settable	FMP12C1	FMP12C1 event interrupt	0x0000 01BC
96	103	settable	FMP12C1 error	FMP12C1 error interrupt	0x0000 01C0

### 10.2.1 EXTI main features

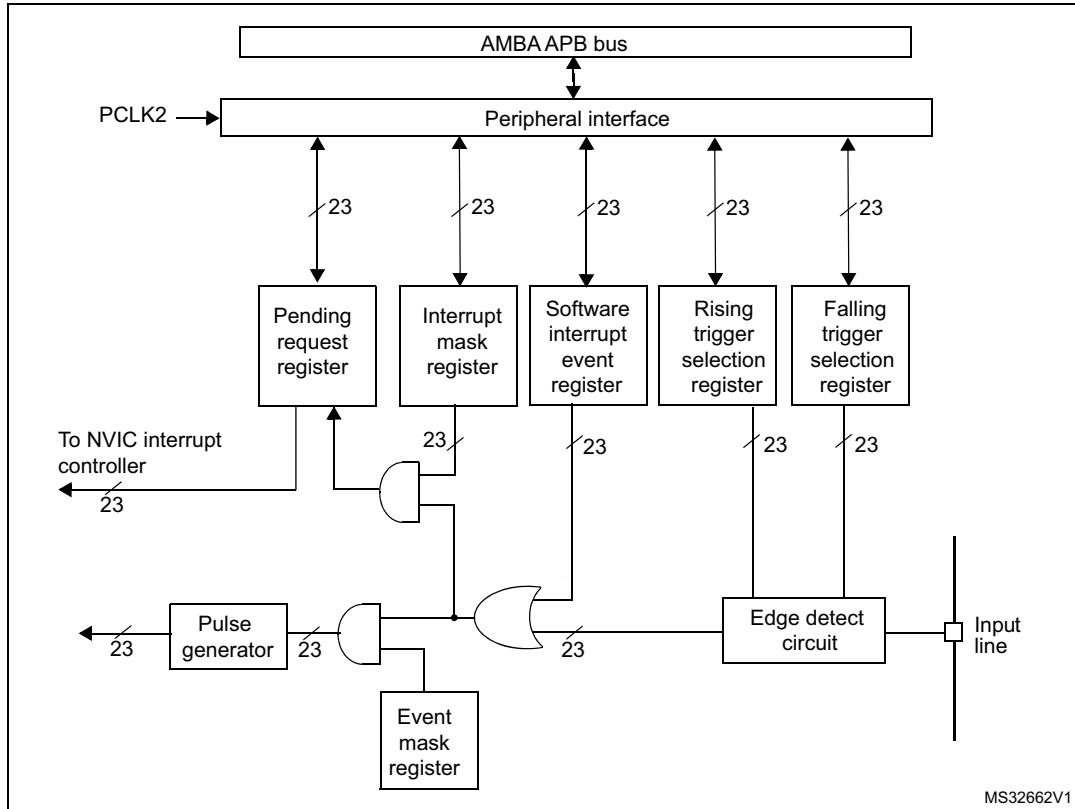
The main features of the EXTI controller are the following:

- independent trigger and mask on each interrupt/event line
- dedicated status bit for each interrupt line
- generation of up to 23 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 STM32F446xx datasheets for details on this parameter.

### 10.2.2 EXTI block diagram

*Figure 29* shows the block diagram.

**Figure 29. External interrupt/event controller block diagram**



### 10.2.3 Wakeup event management

The STM32F446xx microcontrollers 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®-M4 with FPU 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 10.2.4](#).

### 10.2.4 Functional description

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

generated. The pending bit corresponding to the interrupt line is also set. This request is reset by writing a '1' in the pending register.

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

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

### Hardware interrupt selection

To configure the 23 lines as interrupt sources, use the following procedure:

- Configure the mask bits of the 23 interrupt lines (EXTI\_IMR)
- Configure the Trigger selection bits of the interrupt lines (EXTI\_RTSR and EXTI\_FTSR)
- Configure the enable and mask bits that control the NVIC IRQ channel mapped to the external interrupt controller (EXTI) so that an interrupt coming from one of the 23 lines can be correctly acknowledged.

### Hardware event selection

To configure the 23 lines as event sources, use the following procedure:

- Configure the mask bits of the 23 event lines (EXTI\_EMR)
- Configure the Trigger selection bits of the event lines (EXTI\_RTSR and EXTI\_FTSR)

### Software interrupt/event selection

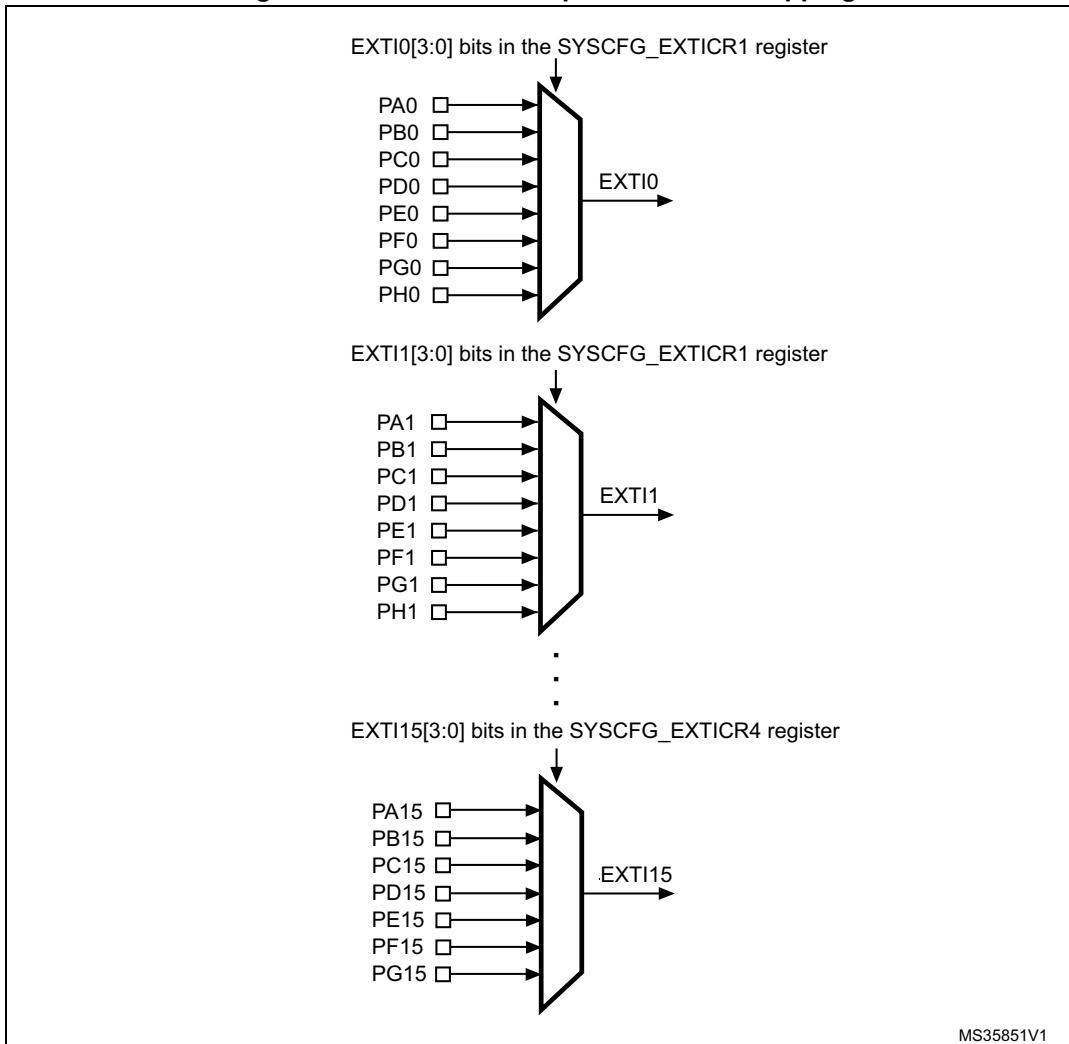
The 23 lines can be configured as software interrupt/event lines. The following is the procedure to generate a software interrupt.

- Configure the mask bits of the 23 interrupt/event lines (EXTI\_IMR, EXTI\_EMR)
- Set the required bit in the software interrupt register (EXTI\_SWIER)

### 10.2.5 External interrupt/event line mapping

Up to 114 GPIOs are connected to the 16 external interrupt/event lines in the following manner:

**Figure 30. External interrupt/event GPIO mapping**



The seven 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 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

## 10.3 EXTI registers

Refer to [Section 1.1 on page 51](#) for a list of abbreviations used in register descriptions.

### 10.3.1 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.	MR22	MR21	MR20	MR19	MR18	MR17	MR16								
									rw						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw															

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

Bits 22:0 **MRx**: Interrupt mask on line x

0: Interrupt request from line x is masked

1: Interrupt request from line x is not masked

### 10.3.2 Event mask register (EXTI\_EMR)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	MR22	MR21	MR20	MR19	MR18	MR17	MR16								
									rw						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw															

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

Bits 22:0 **MRx**: Event mask on line x

0: Event request from line x is masked

1: Event request from line x is not masked

### 10.3.3 Rising trigger selection register (EXTI\_RTSR)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	TR22	TR21	TR20	Res.	TR18	TR17	TR16								
									rw	rw	rw		rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw															

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

Bits 22:20 **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

Bit 19 Reserved, must be kept at reset value.

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

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

**Note:** *The external wakeup lines are edge triggered, no 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.*

### 10.3.4 Falling trigger selection register (EXTI\_FTSR)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	TR22	TR21	TR20	Res.	TR18	TR17	TR16								
									rw	rw	rw		rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw															

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

Bits 22:20 **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.

Bit 19 Reserved, must be kept at reset value.

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

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

**Note:** The external wakeup lines are edge triggered, no 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.

### 10.3.5 Software interrupt event register (EXTI\_SWIER)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	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 31:23 Reserved, must be kept at reset value.

Bits 22:0 **SWIER<sub>x</sub>:** Software Interrupt on line x

If interrupt are enabled on line x in the EXTI\_IMR register, writing '1' to SWIER<sub>x</sub> 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).

### 10.3.6 Pending register (EXTI\_PR)

Address offset: 0x14

Reset value: undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	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 31:23 Reserved, must be kept at reset value.

Bits 22:0 **PR<sub>x</sub>:** Pending bit on line x

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'.

### 10.3.7 EXTI register map

*Table 39* gives the EXTI register map and the reset values.

**Table 39. External interrupt/event controller register map and reset values**

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

Refer to *Table 1 on page 56* for the register boundary addresses.

## 11 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

### 11.1 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-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.

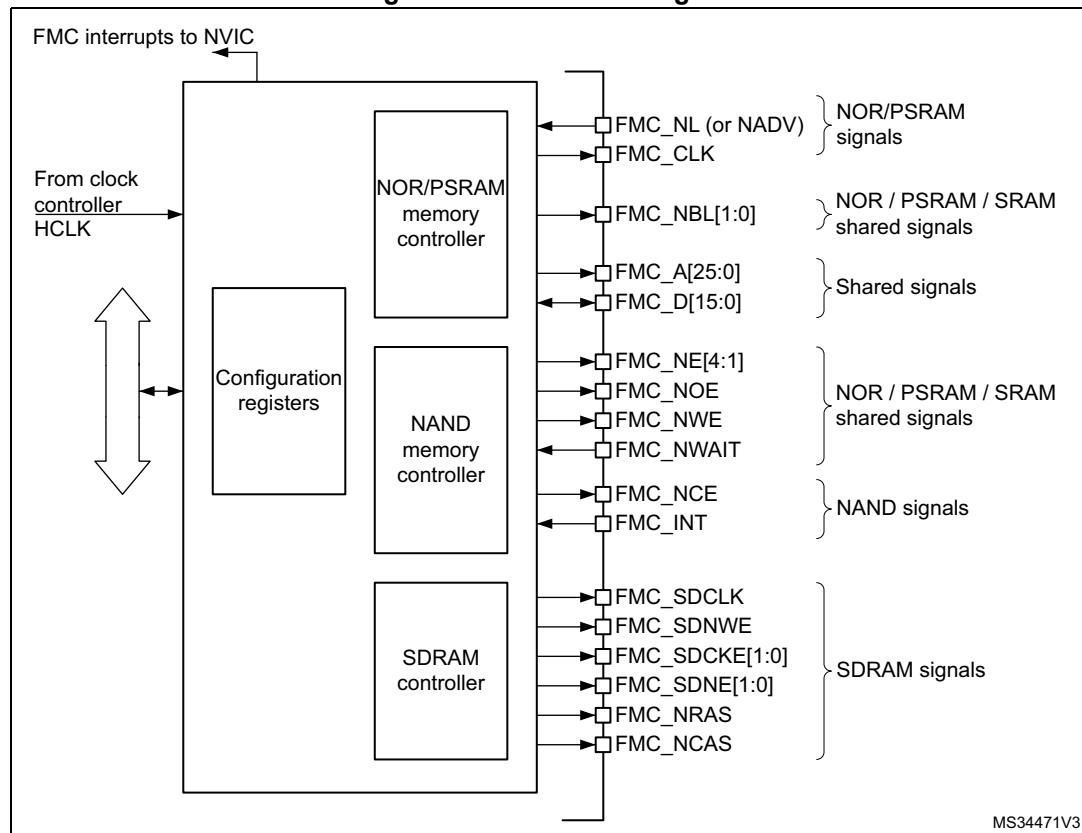
## 11.2 Block diagram

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 external device interface

The block diagram is shown in the figure below.

**Figure 31. FMC block diagram**



## 11.3 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®-M4 with FPU 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.

### 11.3.1 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[1:0].  
Bytes to be written are addressed by NBL[1:0].  
All memory bytes are read (NBL[1:0] 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

Wrap burst mode for synchronous memories is not supported. The memories must be configured in linear burst mode of undefined length.

### Configuration registers

The FMC can be configured through a set of registers. Refer to [Section 11.5.6](#), for a detailed description of the NOR Flash/PSRAM controller registers. Refer to [Section 11.6.7](#), for a detailed description of the NAND Flash registers and to [Section 11.7.5](#) for a detailed description of the SDRAM controller registers.

## 11.4 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 32](#)):

- 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 32. 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 Reserved	
0x7FFF FFFF		
0x8000 0000	Bank 3 4 x 64 MB	NAND Flash memory
0x8FFF FFFF		
0x9000 0000	Bank 4 Reserved	
0x9FFF FFFF		
0xC000 0000	SDRAM Bank 1 4 x 64 MB	SDRAM
0xCFFF FFFF		
0xD000 0000	SDRAM Bank 2 4 x 64 MB	
0xDFFF FFFF		

MS30444V3

### 11.4.1 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] <sup>(1)</sup>	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 <sup>(1)</sup>	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

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].  
Whatever the external memory width, FMC\_A[0] should be connected to external memory address A[0].

### 11.4.2 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.

### 11.4.3 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 <sup>(1)</sup>	Internal bank	Row address	Column address <sup>(2)</sup>	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

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 or 16 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<sup>(1)(2)</sup>**

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<sup>(1)(2)</sup>**

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 <sup>(3)</sup>							
	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.

## 11.5 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
- PSRAM (Cellular RAM)
  - 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 11.5.6: 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 11.5.5: 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 11.5.6: NOR/PSRAM controller registers](#)).

The programmable memory parameters include access times (see [Table 48](#)) and support for wait management (for PSRAM and NOR Flash accessed in burst mode).

**Table 48. 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

### 11.5.1 External memory interface signals

*Table 49*, *Table 50* and *Table 51* 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 49. 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[15:0]	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 50. 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 51. 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[15:0]	I/O	Data bidirectional bus
NE[x]	O	Chip Select, x = 1..4 (called NCE by PSRAM (Cellular RAM 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[1:0]	O	Byte lane output. Byte 0 and Byte 1 control (upper and lower byte enable)

The maximum capacity is 512 Mbits.

### PSRAM, 16-bit multiplexed I/Os

**Table 52. 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 (Cellular RAM 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).

#### 11.5.2 Supported memories and transactions

*Table 53* 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 53. 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]

### 11.5.3 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.

### 11.5.4 NOR Flash/PSRAM controller asynchronous transactions

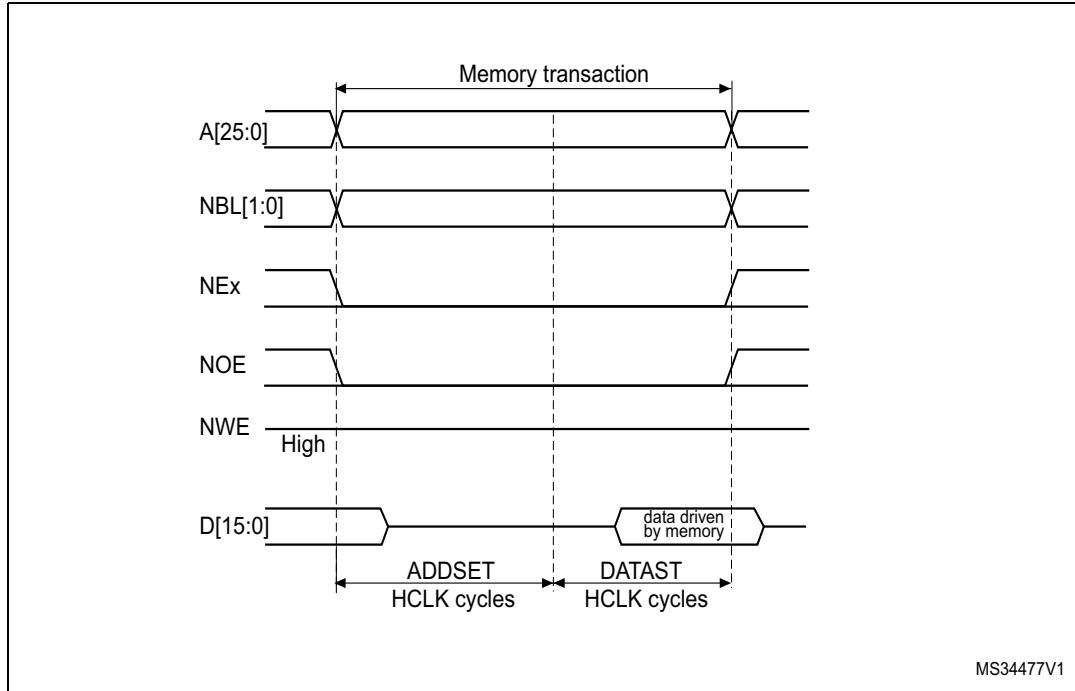
#### Asynchronous static memories (NOR Flash, PSRAM, SRAM)

- 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).

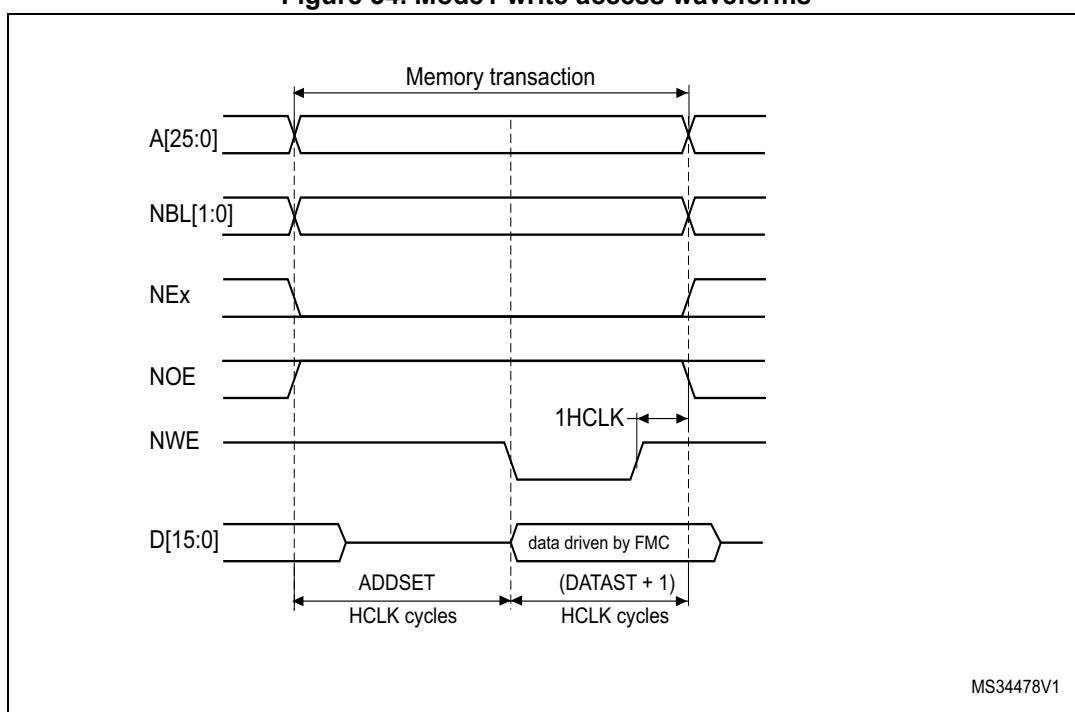
### Mode 1 - SRAM/PSRAM (CRAM)

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 33. Mode1 read access waveforms**



**Figure 34. 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 54. FMC\_BCRx bit fields**

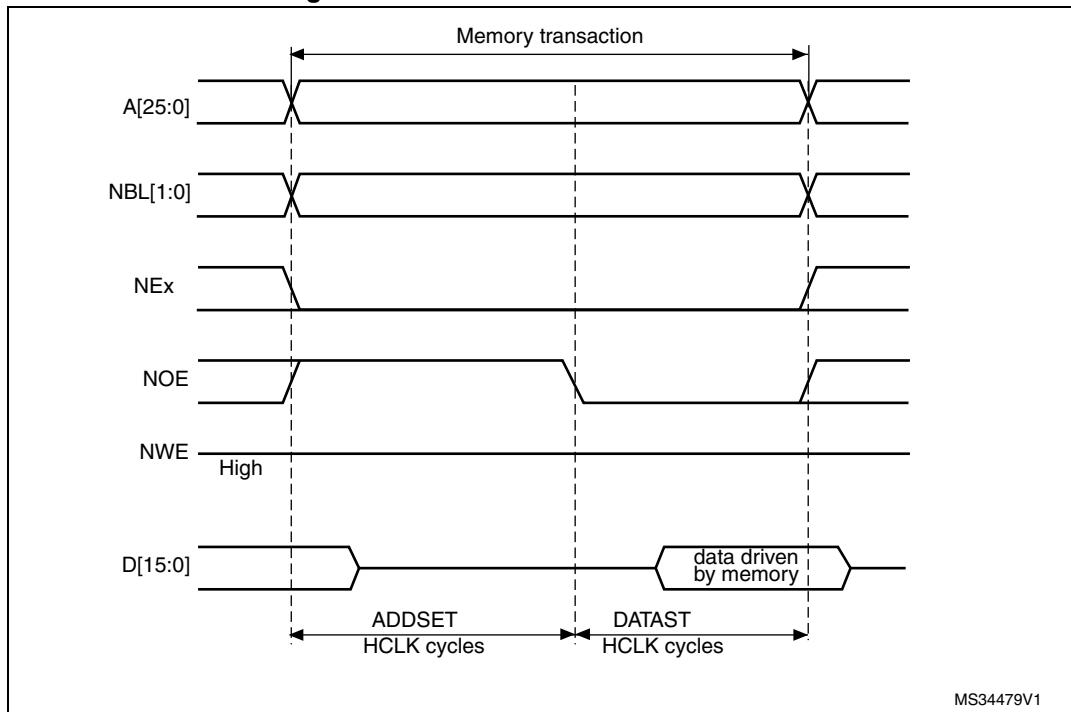
Bit number	Bit name	Value to set
31:22	Reserved	0x000
21	WFDIS	As needed
20	CCLKEN	As needed
19	CBURSTRW	0x0 (no effect in asynchronous mode)
18:16	CPSIZE	0x0 (no effect in asynchronous mode)
15	ASYNCWAIT	Set to 1 if the memory supports this feature. Otherwise keep at 0.
14	EXTMOD	0x0
13	WAITEN	0x0 (no effect in asynchronous mode)
12	WREN	As needed
11	Reserved	0x0
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 55. 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.

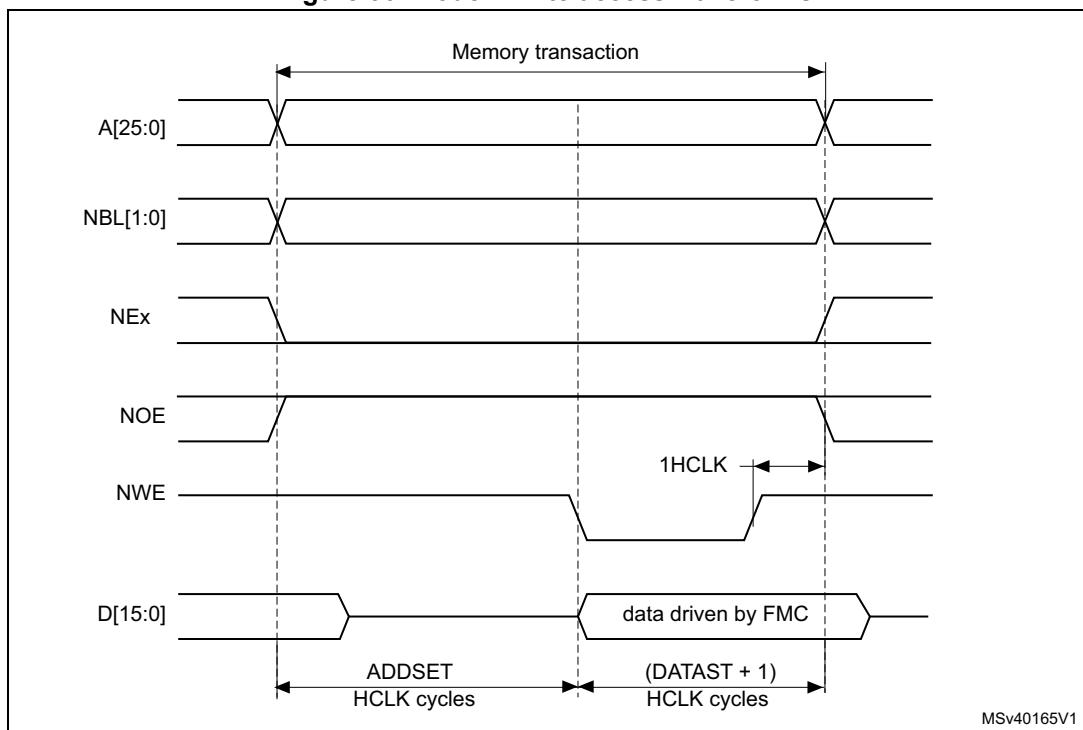
### Mode A - SRAM/PSRAM (CRAM) OE toggling

**Figure 35. ModeA read access waveforms**



1. NBL[1:0] are driven low during the read access

**Figure 36. ModeA write access waveforms**



The differences compared with Mode1 are the toggling of NOE and the independent read and write timings.

**Table 56. FMC\_BCRx bit fields**

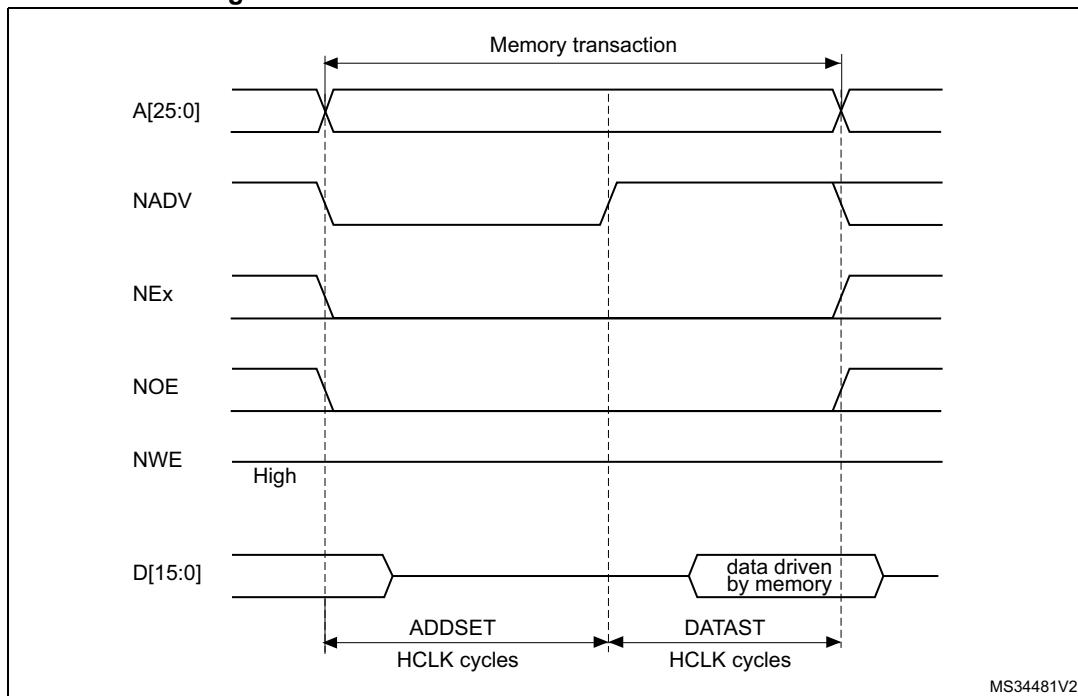
Bit number	Bit name	Value to set
31:22	Reserved	0x000
21	WFDIS	As needed
20	CCLKEN	As needed
19	CBURSTRW	0x0 (no effect in asynchronous mode)
18:16	CPSIZE	0x0 (no effect in asynchronous mode)
15	ASYNCWAIT	Set to 1 if the memory supports this feature. Otherwise keep at 0.
14	EXTMOD	0x1
13	WAITEN	0x0 (no 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

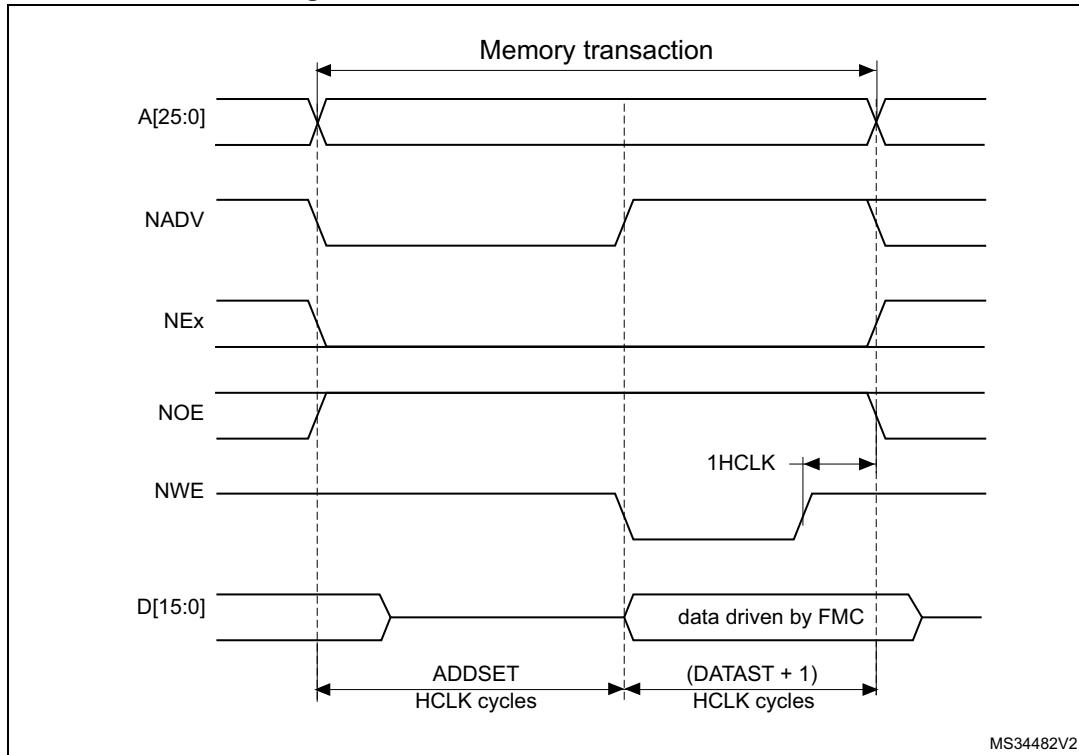
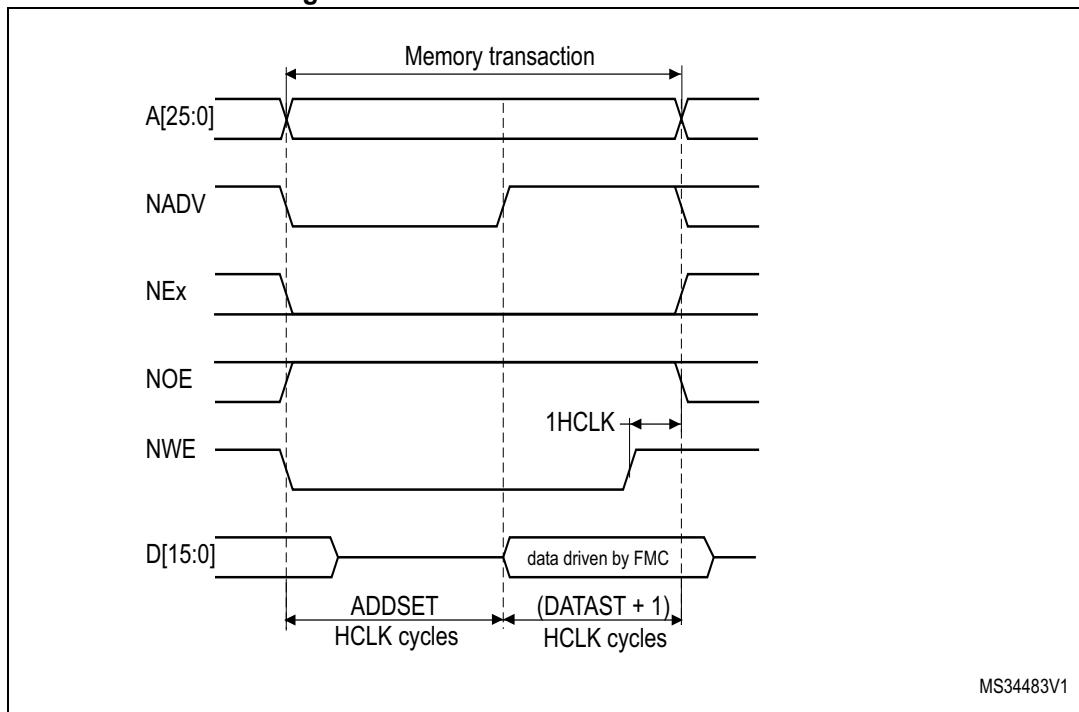
**Table 57. FMC\_BTRx bit fields**

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 58. 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 37. Mode2 and mode B read access waveforms**

**Figure 38. Mode2 write access waveforms****Figure 39. 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 59. 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 60. FMC\_BTRx bit fields**

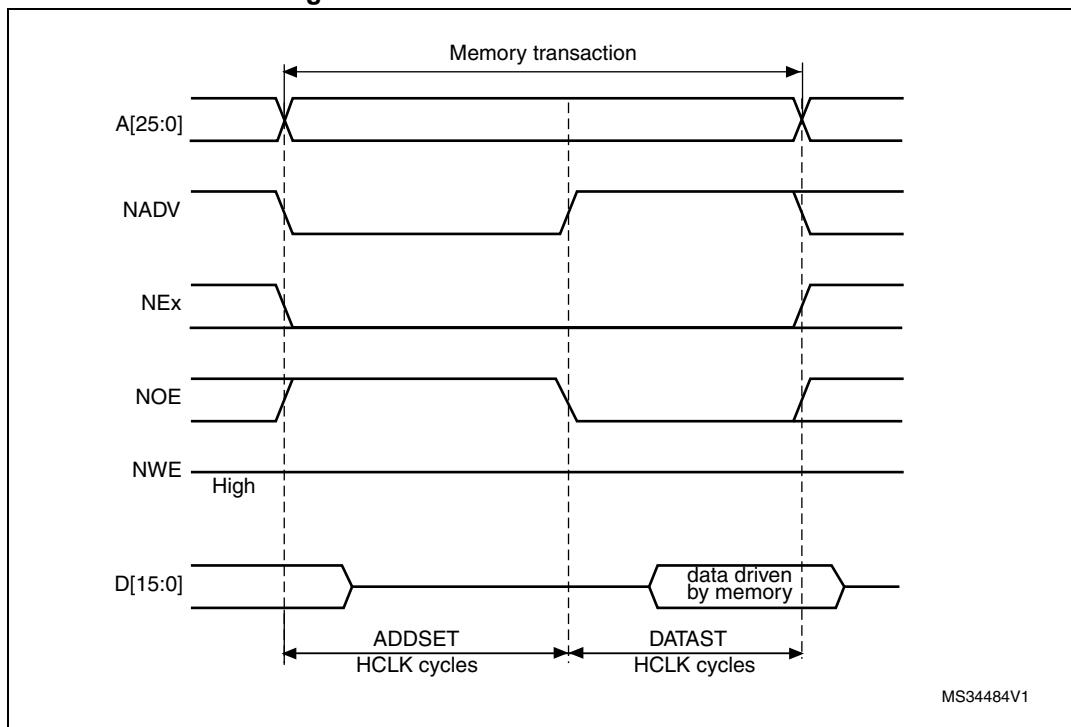
Bit number	Bit name	Value to set
31:30	Reserved	0x0
29:28	ACCMOD	0x1 if extended mode is set
27:24	DATLAT	Don't care
23:20	CLKDIV	Don't care
19:16	BUSTURN	Time between NEx high to NEx low (BUSTURN HCLK).
15:8	DATAST	Duration of the access second phase (DATAST HCLK cycles) for read accesses.
7:4	ADDHLD	Don't care
3:0	ADDSET	Duration of the access first phase (ADDSET HCLK cycles) for read accesses. Minimum value for ADDSET is 0.

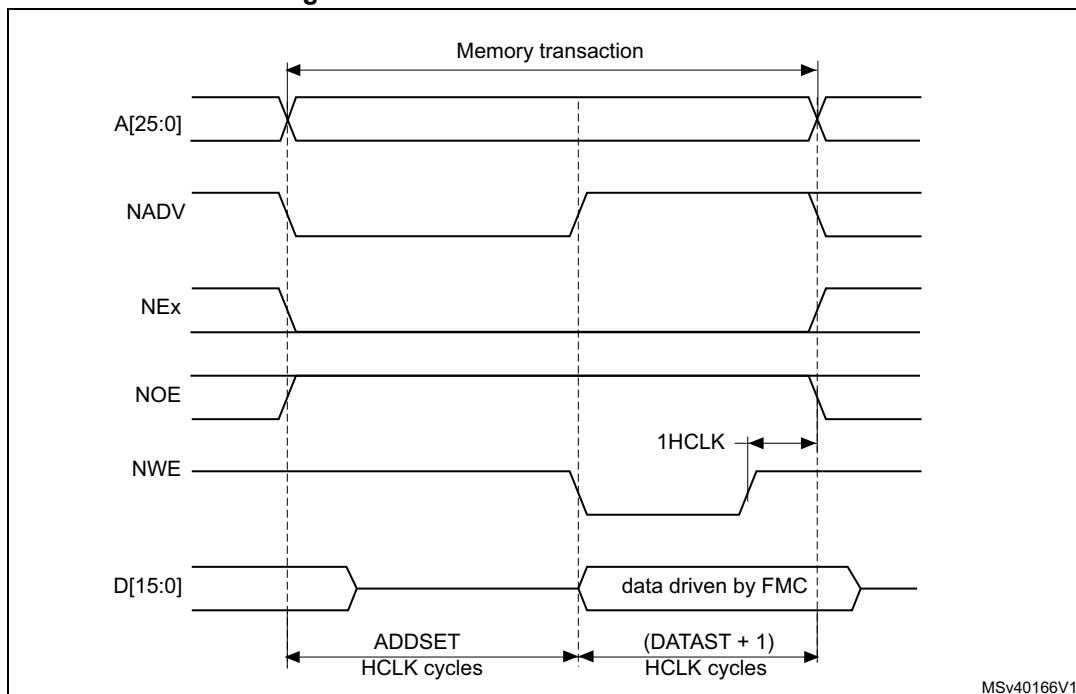
**Table 61. FMC\_BWTRx bit fields**

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 40. ModeC read access waveforms**

**Figure 41. ModeC write access waveforms**

The differences compared with Mode1 are the toggling of NOE and the independent read and write timings.

**Table 62. 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	0x1
5:4	MWID	As needed

**Table 62. FMC\_BCRx bit fields (continued)**

<b>Bit number</b>	<b>Bit name</b>	<b>Value to set</b>
3:2	MTYP	0x02 (NOR Flash memory)
1	MUXEN	0x0
0	MBKEN	0x1

**Table 63. FMC\_BTRx bit fields**

<b>Bit number</b>	<b>Bit name</b>	<b>Value to set</b>
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 64. FMC\_BWTRx bit fields**

<b>Bit number</b>	<b>Bit name</b>	<b>Value to set</b>
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 42. ModeD read access waveforms

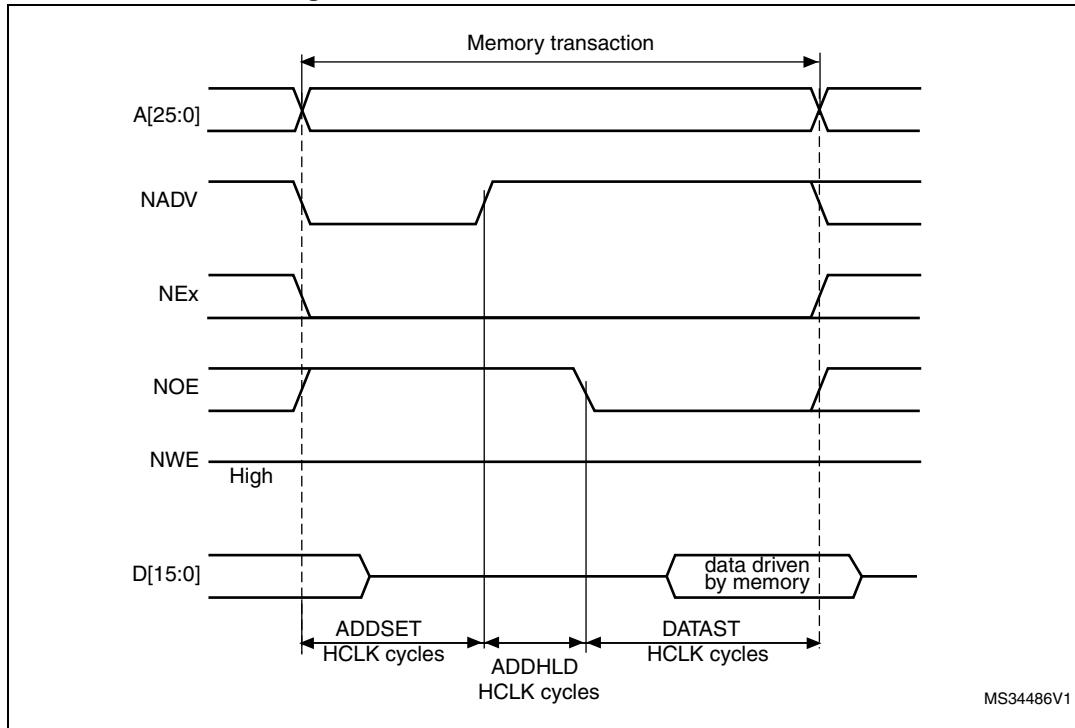
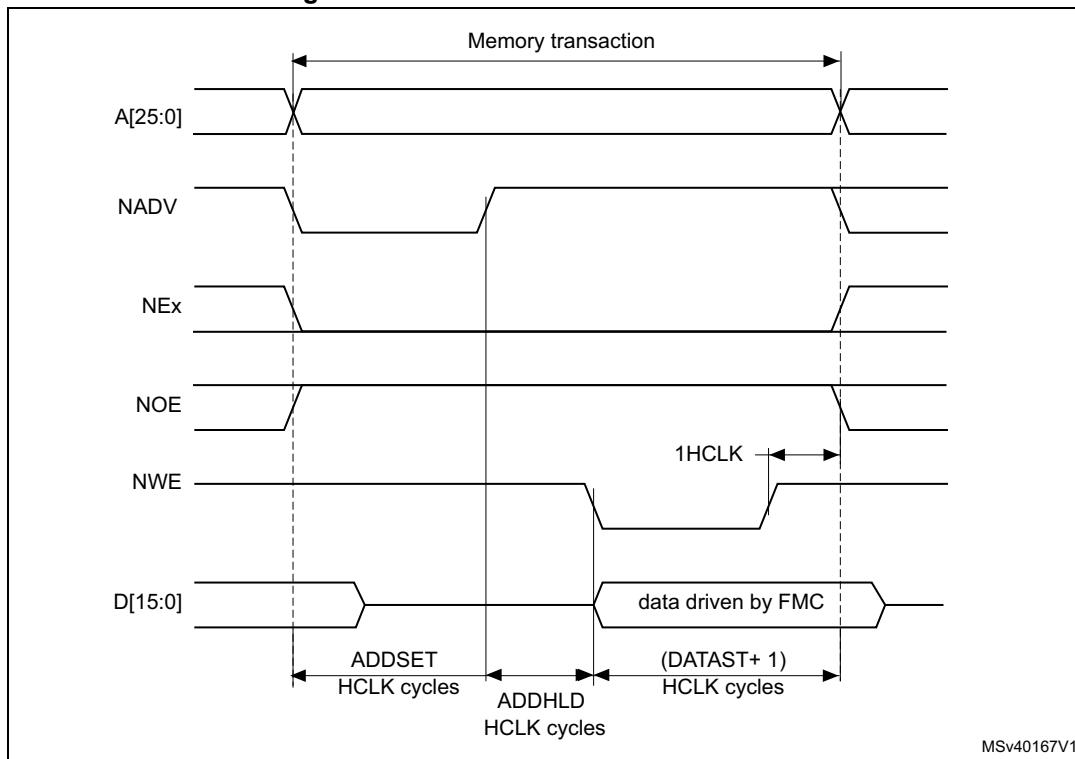


Figure 43. ModeD write access waveforms



The differences with Mode1 are the toggling of NOE that goes on toggling after NADV changes and the independent read and write timings.

**Table 65. FMC\_BCRx bit fields**

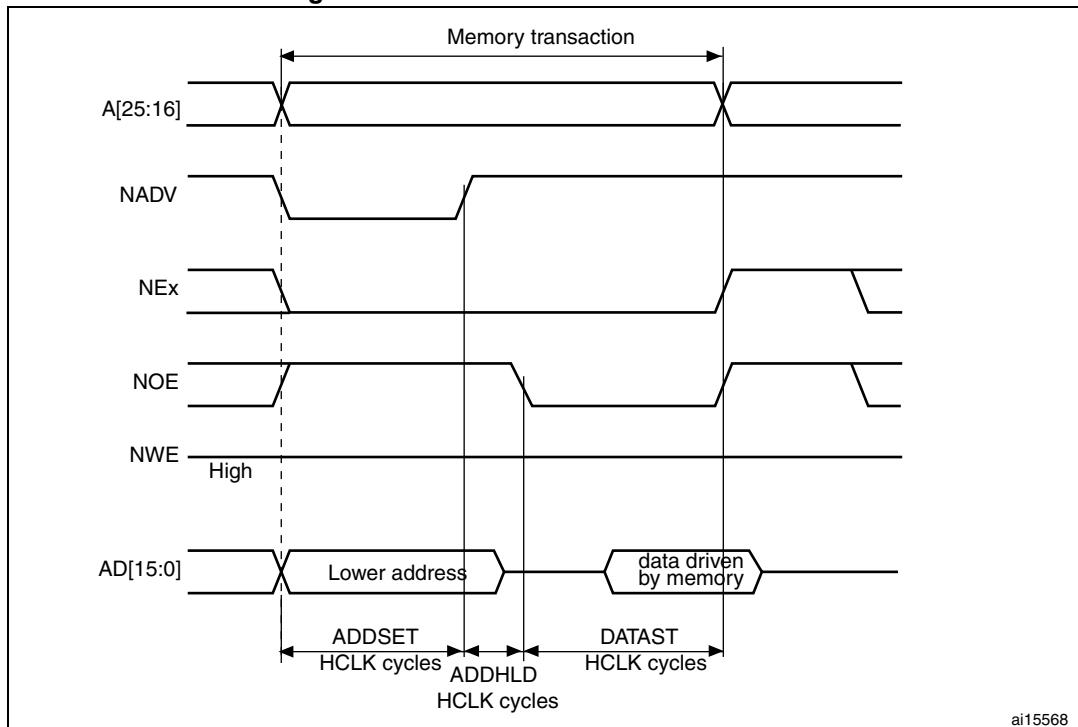
Bit number	Bit name	Value to set
31:22	Reserved	0x000
21	WFDIS	As needed
20	CCLKEN	As needed
19	CBURSTRW	0x0 (no effect in asynchronous mode)
18:16	CPSIZE	0x0 (no effect in asynchronous mode)
15	ASYNCWAIT	Set to 1 if the memory supports this feature. Otherwise keep at 0.
14	EXTMOD	0x1
13	WAITEN	0x0 (no 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	Set according to memory support
5:4	MWID	As needed
3:2	MTYP	As needed
1	MUXEN	0x0
0	MBKEN	0x1

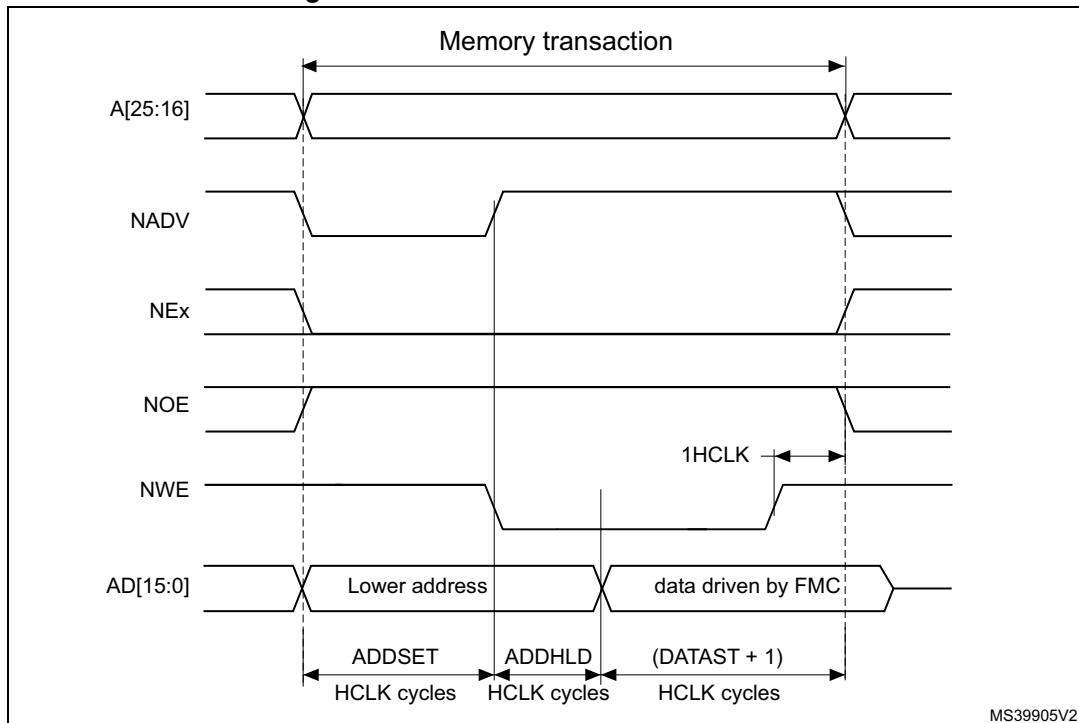
**Table 66. FMC\_BTRx bit fields**

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 67. 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 44. Muxed read access waveforms**

**Figure 45. Muxed write access waveforms**

The difference with ModeD is the drive of the lower address byte(s) on the data bus.

**Table 68. 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 68. 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 69. 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:

$$\text{DATAST} \geq (4 \times \text{HCLK}) + (\text{max\_wait\_assertion\_time} - \text{address\_phase} - \text{hold\_phase})$$

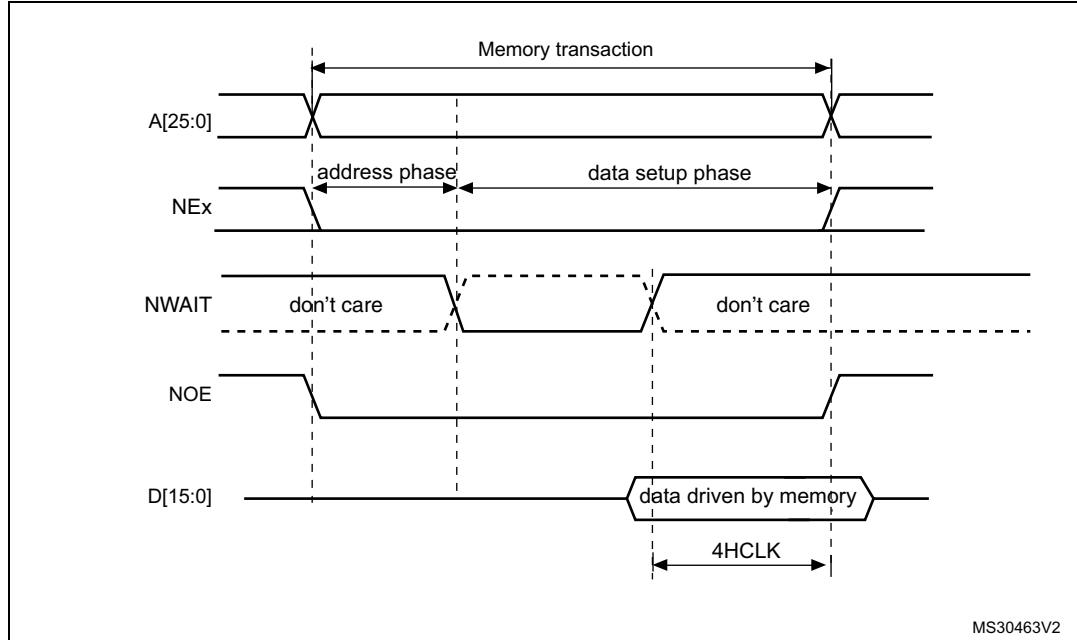
otherwise

$$\text{DATAST} \geq 4 \times \text{HCLK}$$

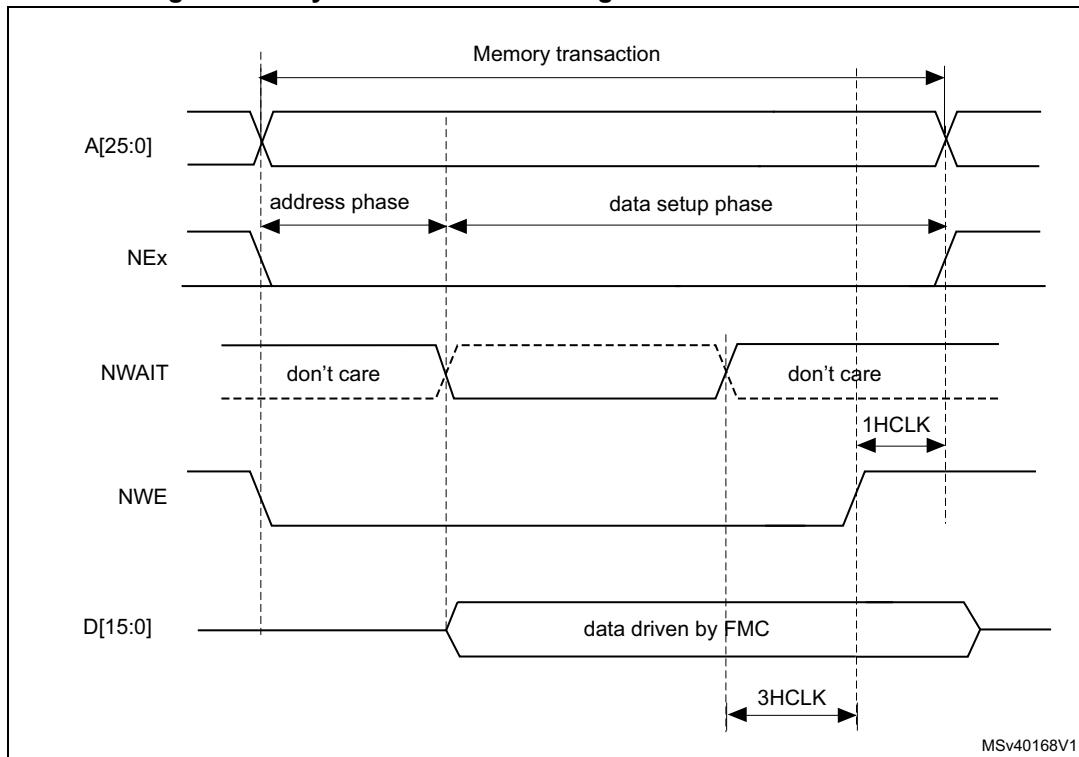
where max\_wait\_assertion\_time is the maximum time taken by the memory to assert the WAIT signal once NEx/NOE/NWE is low.

*Figure 46* and *Figure 47* 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 46. Asynchronous wait during a read access waveforms**



1. NWAIT polarity depends on WAITPOL bit setting in FMC\_BCRx register.

**Figure 47. Asynchronous wait during a write access waveforms**

1. NWAIT polarity depends on WAITPOL bit setting in FMC\_BCRx register.

### 11.5.5 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:

$$\text{FMC\_CLK divider ratio} = \max(\text{CLKDIV} + 1, \text{MWID(AHB data size)})$$

Whatever MWID size: 16 or 8-bit, the FMC\_CLK divider ratio is always defined by the programmed CLKDIV value.

Example:

- 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 Cellular RAM 1.5

Cellular RAM 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).

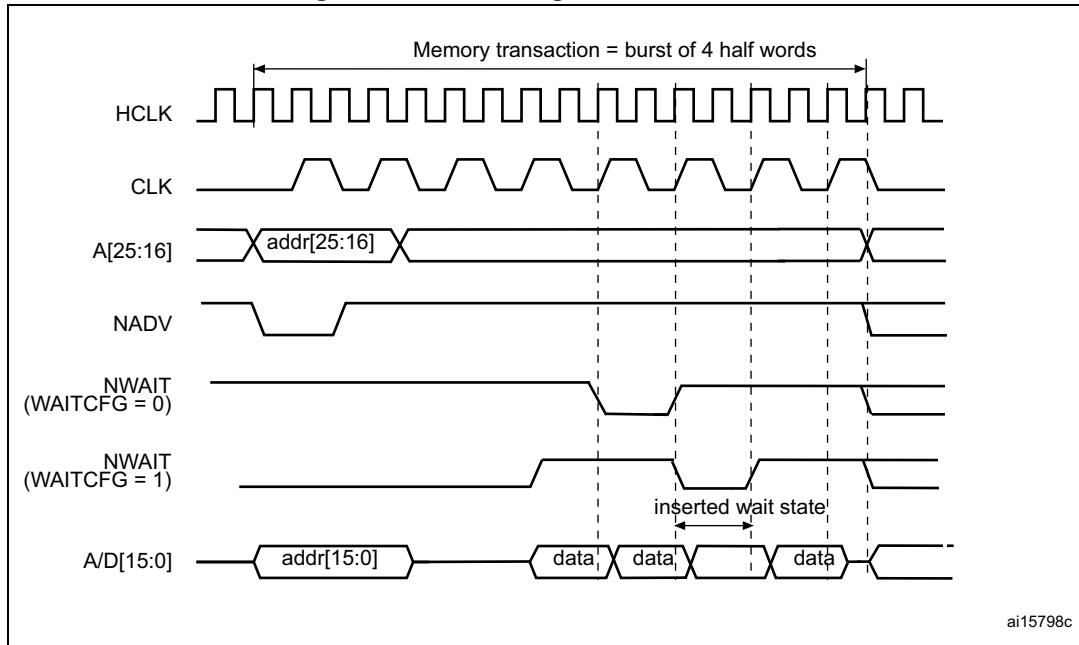
When NWAIT is inactive, the data is considered valid either immediately (bit WAITCFG = 1) or on the next clock edge (bit WAITCFG = 0).

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:

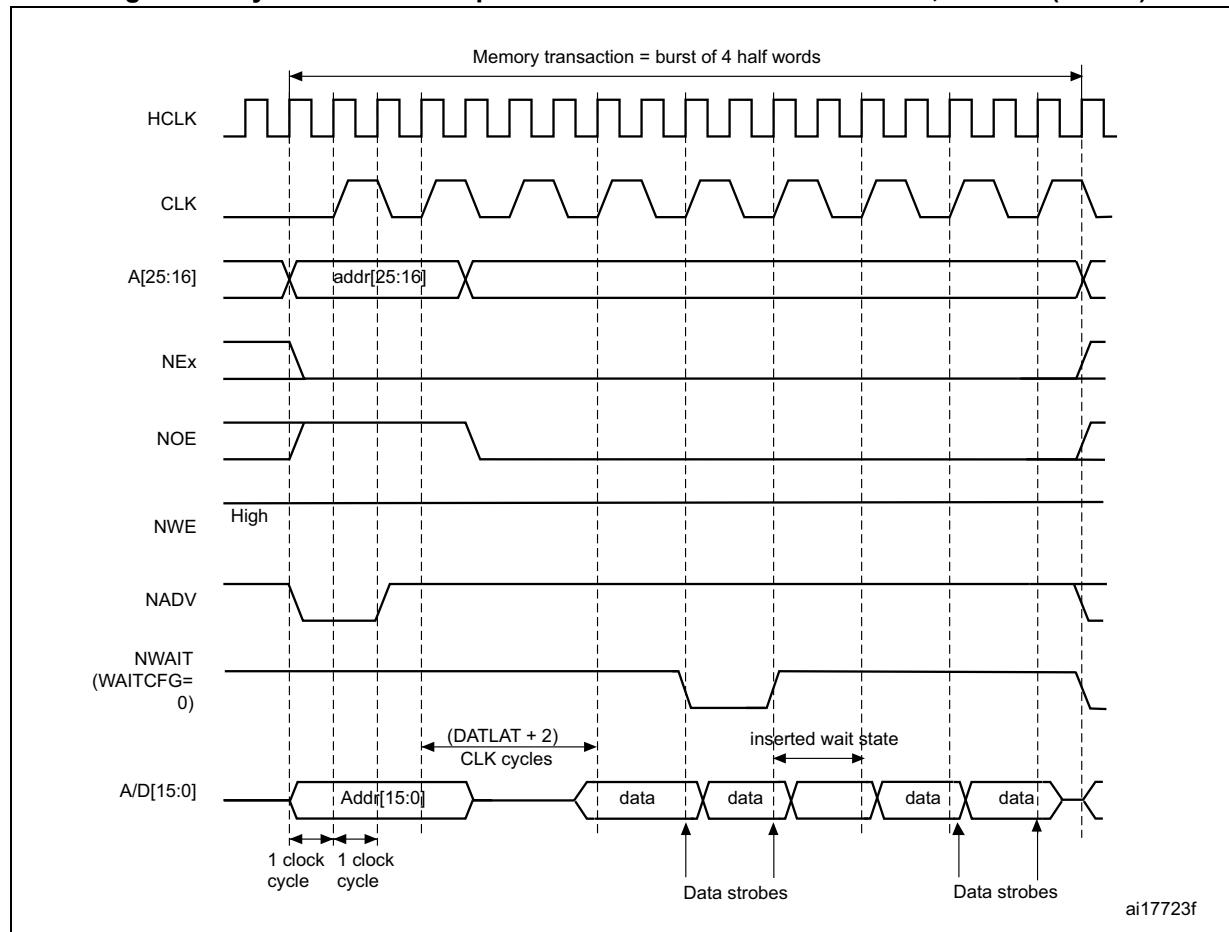
- The Flash memory asserts the NWAIT signal one data cycle before the wait state (default after reset).
- The Flash memory asserts the NWAIT signal during the wait state

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 48. Wait configuration waveforms**

ai15798c

Figure 49. 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 70. 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

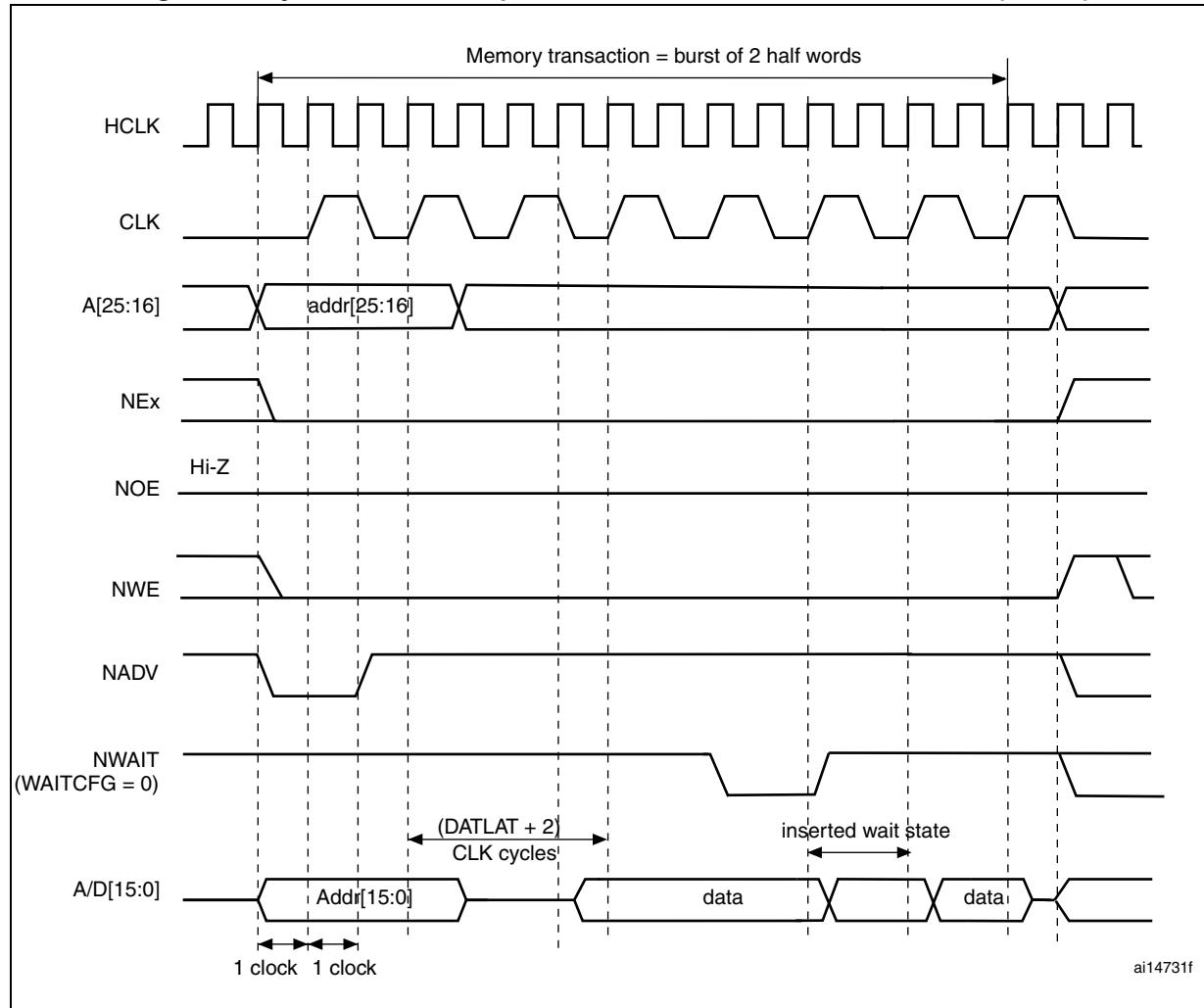
**Table 70. FMC\_BCRx bit fields (continued)**

<b>Bit number</b>	<b>Bit name</b>	<b>Value to set</b>
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 71. FMC\_BTRx bit fields**

<b>Bit number</b>	<b>Bit name</b>	<b>Value to set</b>
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 50. 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 72. 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 72. FMC\_BCRx bit fields (continued)**

<b>Bit number</b>	<b>Bit name</b>	<b>Value to set</b>
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 73. FMC\_BTRx bit fields**

<b>Bit number</b>	<b>Bit name</b>	<b>Value to set</b>
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

### 11.5.6 NOR/PSRAM controller registers

#### SRAM/NOR-Flash chip-select control registers 1..4 (FMC\_BCR1..4)

Address offset:  $8 * (x - 1)$ ,  $x = 1 \dots 4$

Reset value: 0x0000 30DB for Bank1 and 0x0000 30D2 for Bank 2 to 4

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		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 Cellular RAM 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: reserved
- 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 registers 1..4 (FMC\_BTR1..4)**

Address offset:  $0x04 + 8 * (x - 1)$ ,  $x = 1..4$

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		DATLAT				CLKDIV				BUSTURN			
		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								ADDHLD				ADDSET			
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 11.5.5: 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 modes muxed and D.
- 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 modes muxed and D.
  - An asynchronous read to an asynchronous or synchronous write to any static bank or dynamic bank except for modes muxed and D.
  - 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 33](#) to [Figure 45](#)), 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 33](#) to [Figure 45](#)).

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 33](#) to [Figure 45](#)), 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 33](#) to [Figure 45](#)).

*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 33](#) to [Figure 45](#)), 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 33](#) to [Figure 45](#)).

*Note: In synchronous accesses, this value is don't care.*

*In Muxed mode or Mode D, 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:  $0x104 + 8 * (x - 1)$ ,  $x = 1 \dots 4$

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		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	BUSTURN					
		rw	rw									rw	rw	rw	rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
DATAST								ADDHLD				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 modes muxed and D.
- 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 33](#) to [Figure 45](#)), 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 42](#) to [Figure 45](#)), 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 33](#) to [Figure 45](#)), 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.*

## 11.6 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 11.6.7](#)). The programmable memory parameters include access timings (shown in [Table 74](#)) and ECC configuration.

**Table 74. 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

### 11.6.1 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 75. 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
D[7:0]	I/O	8-bit multiplexed, bidirectional address/data bus

**Table 75. 8-bit NAND Flash (continued)**

FMC signal name	I/O	Function
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 76. 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.*

### 11.6.2 NAND Flash supported memories and transactions

*Table 77* shows the supported devices, access modes and transactions. Transactions not allowed (or not supported) by the NAND Flash controller are shown in gray.

**Table 77. 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

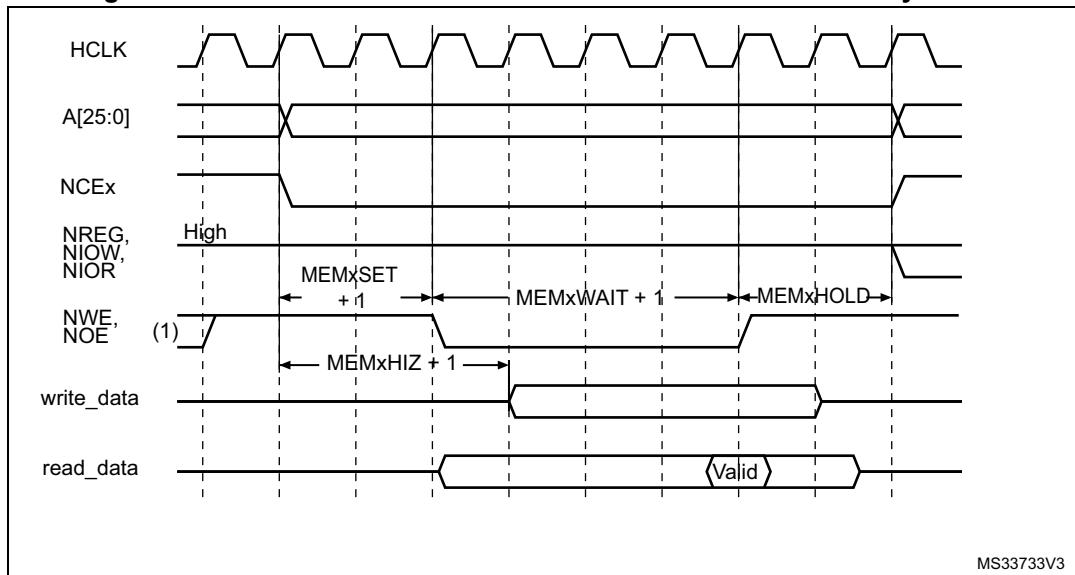
### 11.6.3 Timing diagrams for NAND Flash memory

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 51* shows the timing parameter definitions for common memory accesses, knowing that Attribute memory space access timings are similar.

**Figure 51. 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.

#### 11.6.4 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:

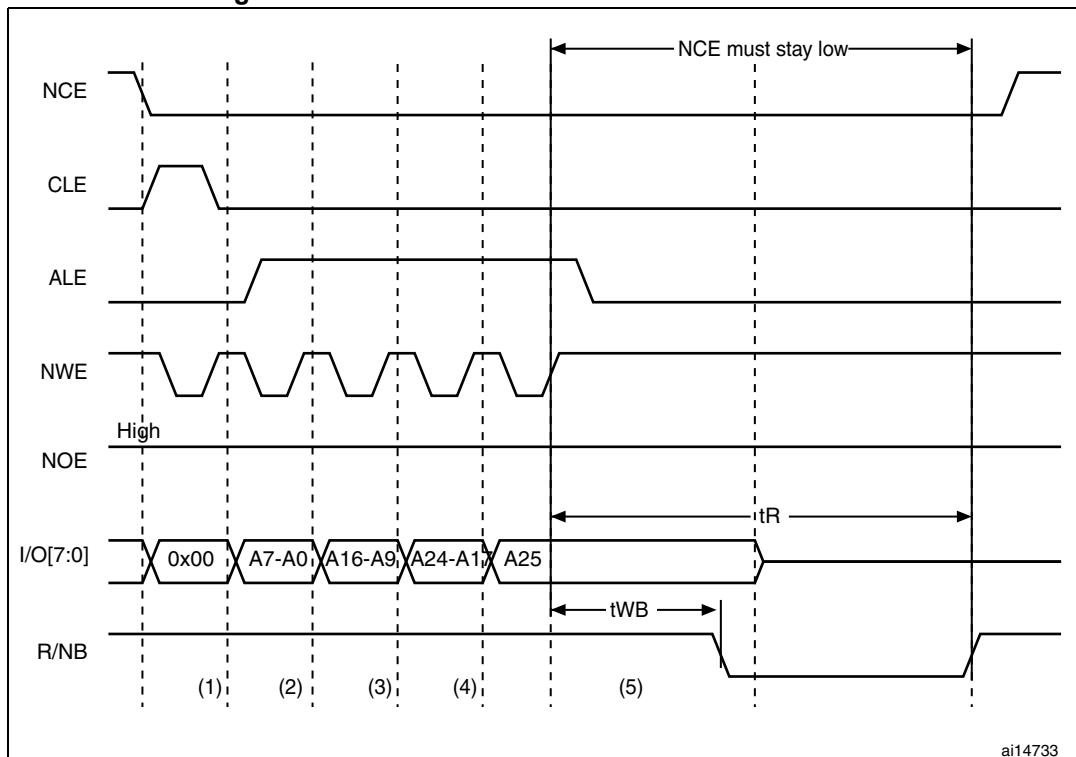
3. Program and enable the corresponding memory bank by configuring the FMC\_PCR and FMC\_PMEM (and for some devices, FMC\_PATT, see [Section 11.6.5: NAND Flash prewait functionality](#)) registers according to the characteristics of the NAND Flash memory (PWID bits for the data bus width of the NAND Flash, PTYP = 1, PWAITEN = 0 or 1 as needed, see [Section 11.4.2: NAND Flash memory address mapping](#) for timing configuration).
4. 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.
5. 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 11.6.5: NAND Flash prewait functionality](#)).
6. 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).
  7. 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.
  8. 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

### 11.6.5 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 52](#)).

**Figure 52. Access to non ‘CE don’t care’ NAND-Flash**



ai14733

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  $\text{ATTTHOLD} \geq 7$  (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.

### 11.6.6 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.

### 11.6.7 NAND Flashcontroller 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.	ECCPS			TAR											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
TAR				TCLR				Res.	Res.	ECCEN	PWID		PTYP	PBKEN	PWAITEN	Res.
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.	Res.	Res.	Res.	Res.	Res.	Res.									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	FEMPT	IFEN	ILEN	IREN	IFS	ILS	IRS								
								r	rw	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
MEMHIZx								MEMHOLDx							
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
MEMWAITx								MEMSETx							
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 11.6.5: NAND Flash prewait functionality](#)).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ATTHIZ								ATTHold							
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								ATTSET							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **ATTHIZ[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 11.6.6: 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
ECCx															
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
ECCx															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **ECC:** ECC result

This field contains the value computed by the ECC computation logic. [Table 78](#) describes the contents of these bit fields.

**Table 78. 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]

## 11.7 SDRAM controller

### 11.7.1 SDRAM controller main features

The main features of the SDRAM controller are the following:

- Two SDRAM banks with independent configuration
- 8-bit, 16-bit data bus width
- 13-bits Address Row, 11-bits Address Column, 4 internal banks: 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)

### 11.7.2 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 79. 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[15:0]	I/O	Bidirectional data bus	FMC_D[15:0]
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[1:0]	O	Output Byte Mask for write accesses (memory signal name: DQM[1:0])	FMC_NBL[1:0]

### 11.7.3 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  $\mu$ 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.

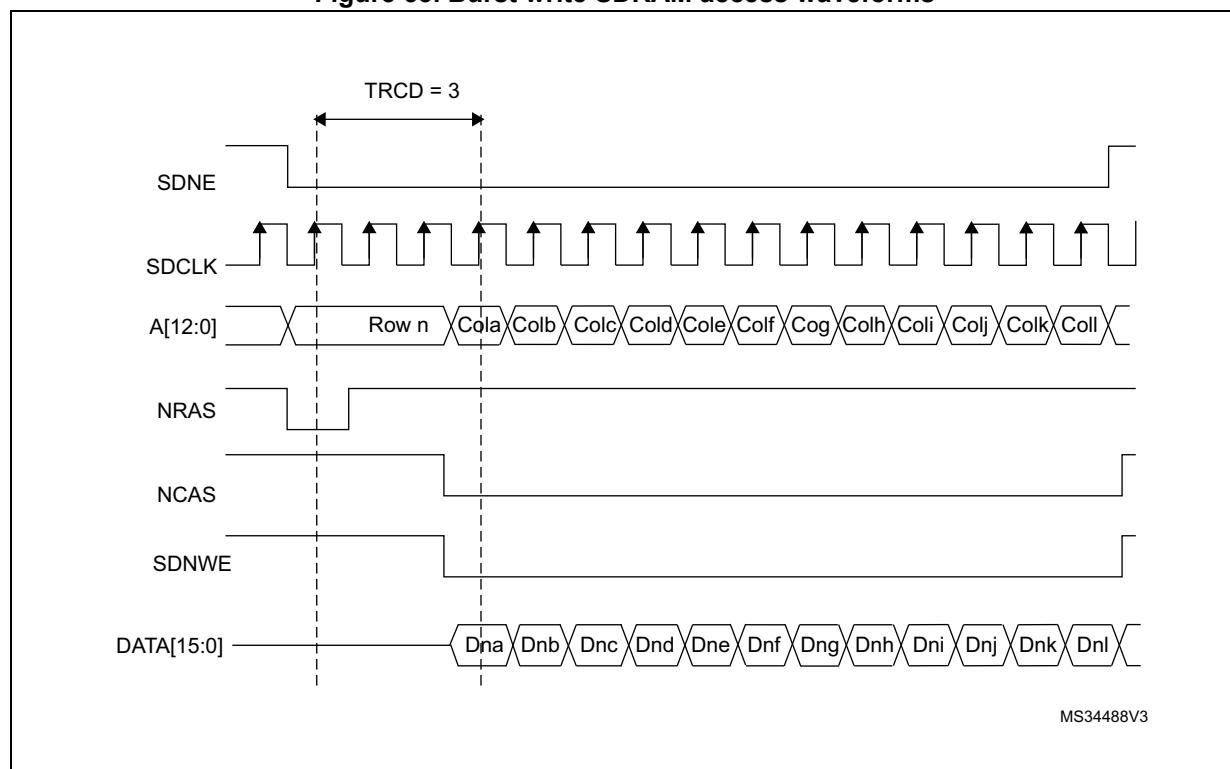
**Note:** If two SDRAM devices are connected to the FMC, all the accesses performed at the same time to both devices by the Command Mode register (Load Mode Register command) are issued using the timing parameters configured for SDRAM Bank 1 (TMRD and TRAS timings) in the FMC\_SDTR1 register.

### 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 53. Burst write SDRAM access waveforms**



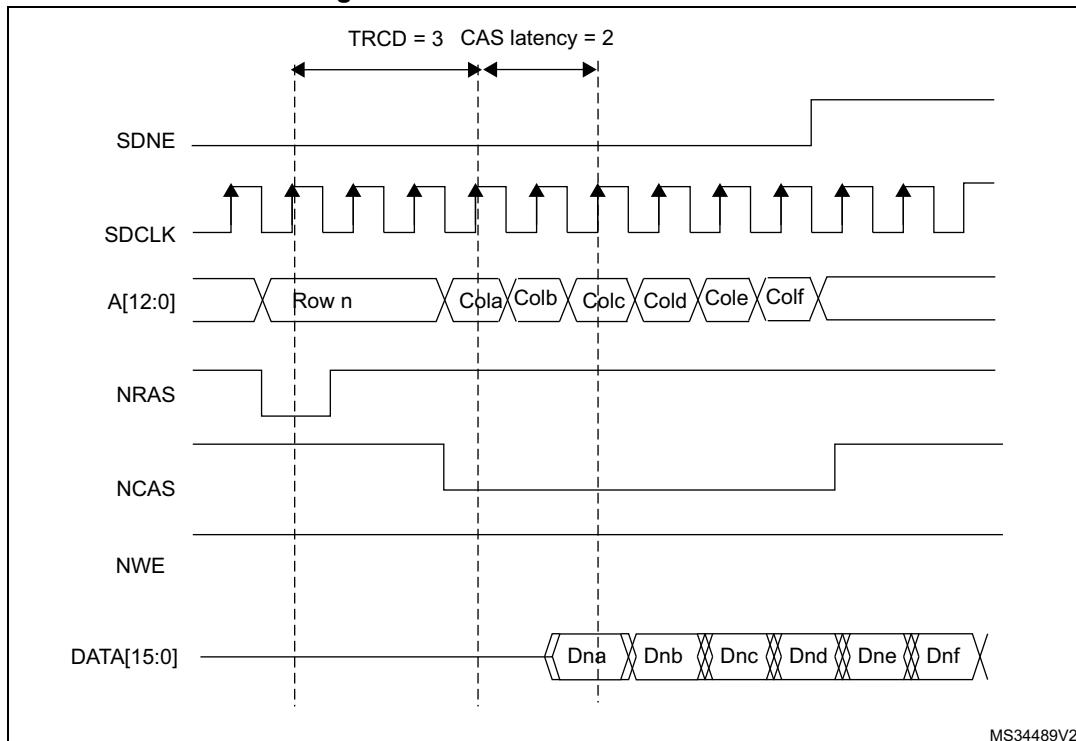
The SDRAM controller always checks the next access.

- 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 54. 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.

$$\text{Number for anticipated data} = \text{CAS latency} + 1 + (\text{RPIPE delay})/2$$

Examples:

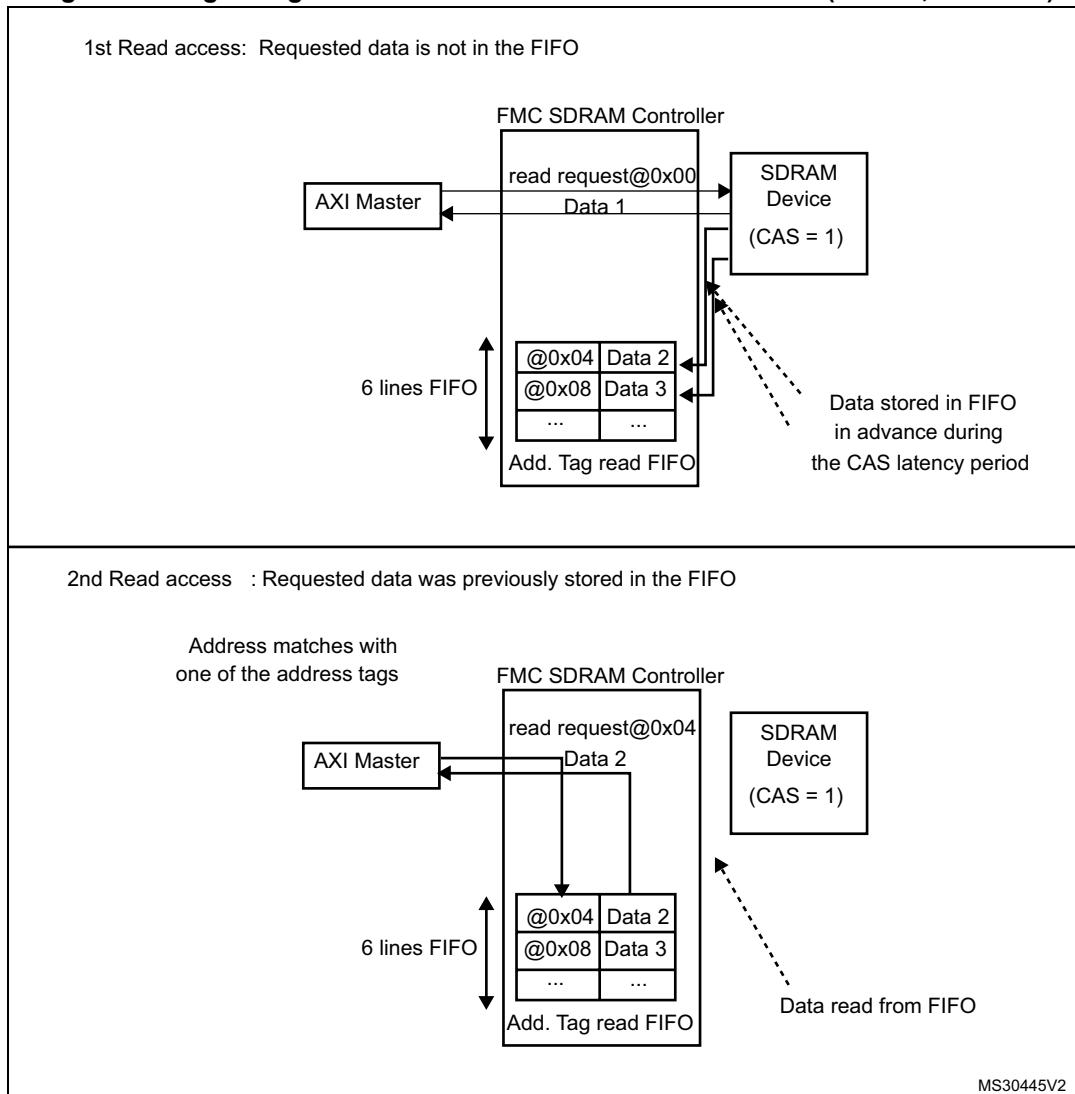
- CAS latency = 3, RPIPE delay = 0: Four data (not committed) are stored in the FIFO.
- CAS latency = 3, RPIPE delay = 0: Five data (not committed) are stored in the FIFO.

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 55. 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.

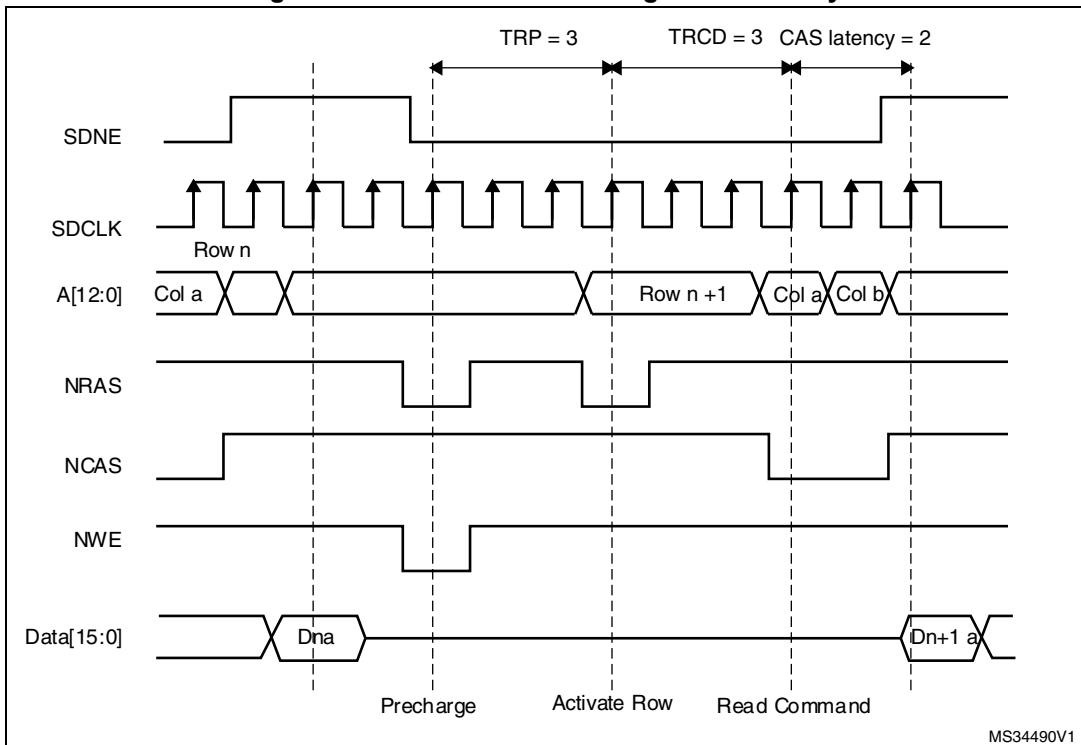
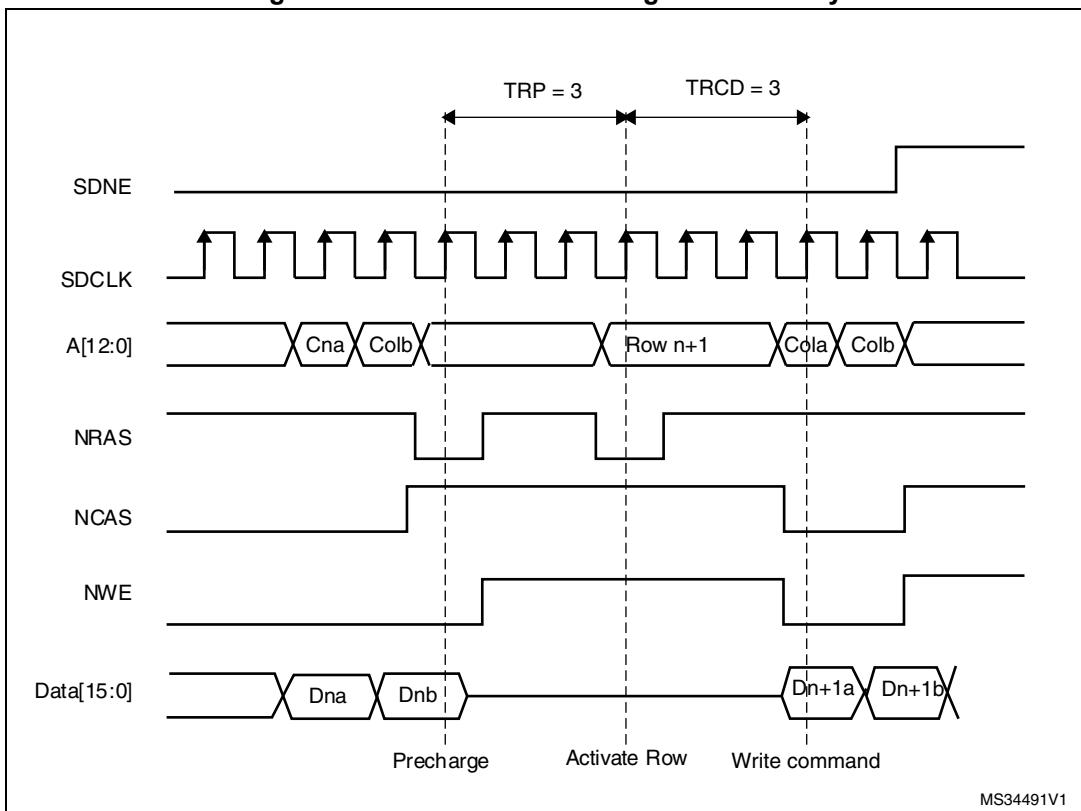
At a row boundary, the automatic activation of the next row is supported for all columns and data bus width configurations.

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 53](#) and [Figure 54](#) for read and burst write access crossing a row boundary.

**Figure 56. Read access crossing row boundary****Figure 57. 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 and the selected SDRAM device is connected to Bank 1, the automatic activation of the next row in device connected to SDRAM Bank2 is not supported. A PALL software command must be issued on Bank1 before any access on Bank2.

### 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.

## 11.7.4 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.

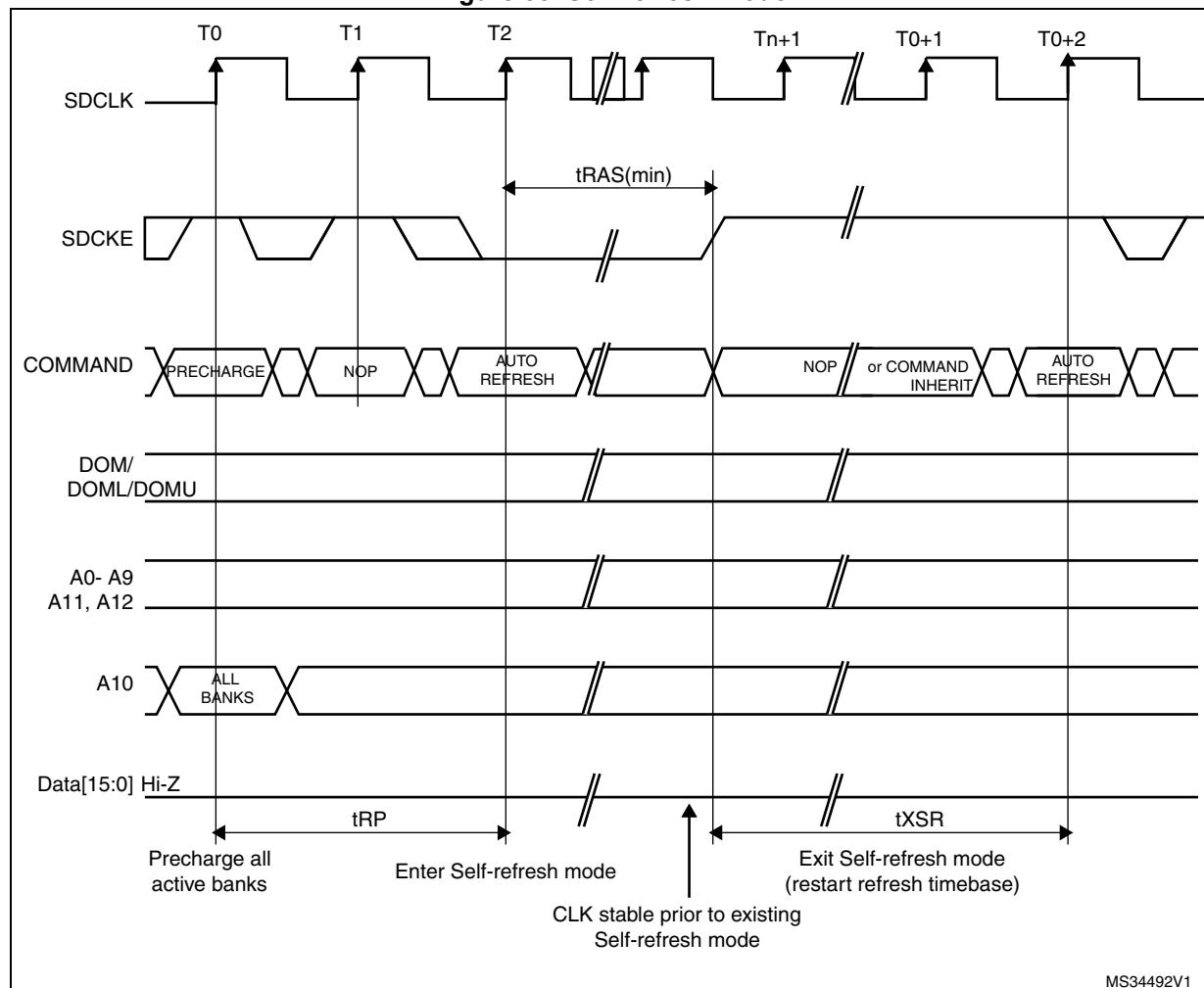
In Self-refresh mode, all SDRAM device inputs become don't care except for SDCKE which remains low.

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 58. Self-refresh mode**

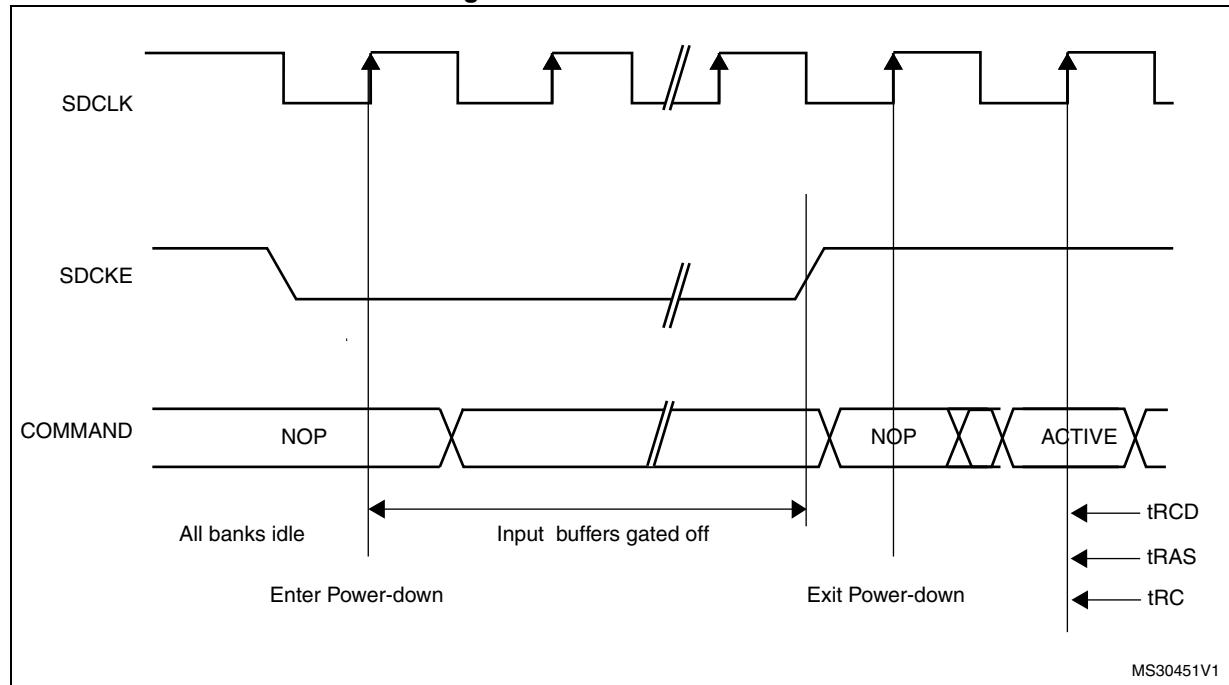


MS34492V1

### 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 59. 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.

### 11.7.5 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 KCK\_FMC clock cycles, for reading data after CAS latency.

00: No KCK\_FMC clock cycle delay

01: One KCK\_FMC clock cycle delay

10: Two KCK\_FMC 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: reserved
- 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.*

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 its associated CTB bit set, the other CTB bit of 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	

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

## 11.8 FMC register map

Table 80. 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																							
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																							
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																							
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																							
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																						
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																						

Table 80. FMC 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	
0x114	FMC_BWTR3	Res.	Res.	Res.	Res.	0	ACCMOD[1:0]	0						BUSTURN[3:0]																				
	Reset value					0	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 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1		
0x11C	FMC_BWTR4	Res.	Res.	Res.	Res.	0	ACCMOD[1:0]	0						BUSTURN[3:0]																				
	Reset value					0	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 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	
0x80	FMC_PCR	Res.	Res.	Res.	Res.	0	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ECCPS[2:0]	TAR[3:0]																			
	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	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0		
0x84	FMC_SR	Res.	Res.	Res.	Res.	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.			
	Reset value					0								1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x88	FMC_PMEM	MEMHIZx[7:0]				MEMHOLDx[7:0]				MEMWAITx[7:0]				MEMSETx[7:0]				MEMWAITx[7:0]				ATTSET[7:0]				ATTSET[7:0]				ATTSET[7:0]				
	Reset value	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	0	0	1	1	1	1	1	0	0
0x8C	FMC_PATT	ATTHIZ[7:0]				ATTHold[7:0]				ATTWait[7:0]				ATTSet[7:0]				ATTSet[7:0]				ATTSet[7:0]				ATTSet[7:0]				ATTSet[7:0]				
	Reset value	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	0	0	1	1	1	1	1	0	0
0x94	FMC_ECCR	ECCx[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	
0x140	FMC_SDCR1	Res.	Res.	Res.	Res.	0	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RPIPE[1:0]	RBURST	SDCLK[1:0]	WP	CAS[1:0]	NB	MWID[1:0]	NR[1:0]	NC												
	Reset value					0								0	0	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0		
0x144	FMC_SDCR2	Res.	Res.	Res.	Res.	0	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SDCLK[1:0]	WP	CAS[1:0]	NB	MWID[1:0]	NR[1:0]	NC											
	Reset value					0								0		1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0		
0x148	FMC_SDTR1	Res.	Res.	Res.	Res.	0	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TRAS[3:0]	TXSR[3:0]	TMRD[3:0]															
	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	1	
0x14C	FMC_SDTR2	Res.	Res.	Res.	Res.	0	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TRAS[3:0]	TXSR[3:0]	TMRD[3:0]															
	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	1	
0x150	FMC_SDCMR	Res.	Res.	Res.	Res.	0	Res.	Res.	Res.	Res.	Res.	Res.	Res.	MRD[12:0]				NRFS[3:0]				MODE[2:0]												
	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	
0x154	FMC_SDRTR	Res.	Res.	Res.	Res.	0	Res.	Res.	Res.	Res.	Res.	Res.	Res.	REIE	0	0	0	0	0	0	0	COUNT[12:0]												
	Reset value					0								0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x158	FMC_SDSR	Res.	Res.	Res.	Res.	0	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value					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 on page 56](#) for the register boundary addresses.

## 12 Quad-SPI interface (QUADSPI)

### 12.1 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.

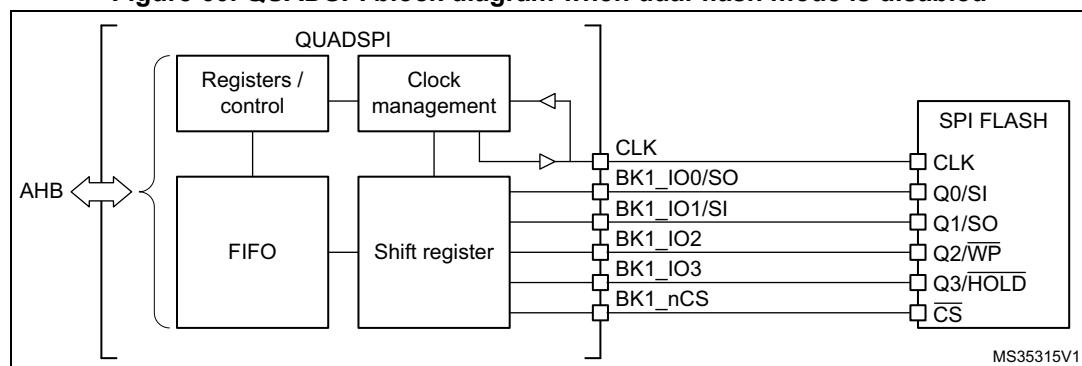
### 12.2 QUADSPI main features

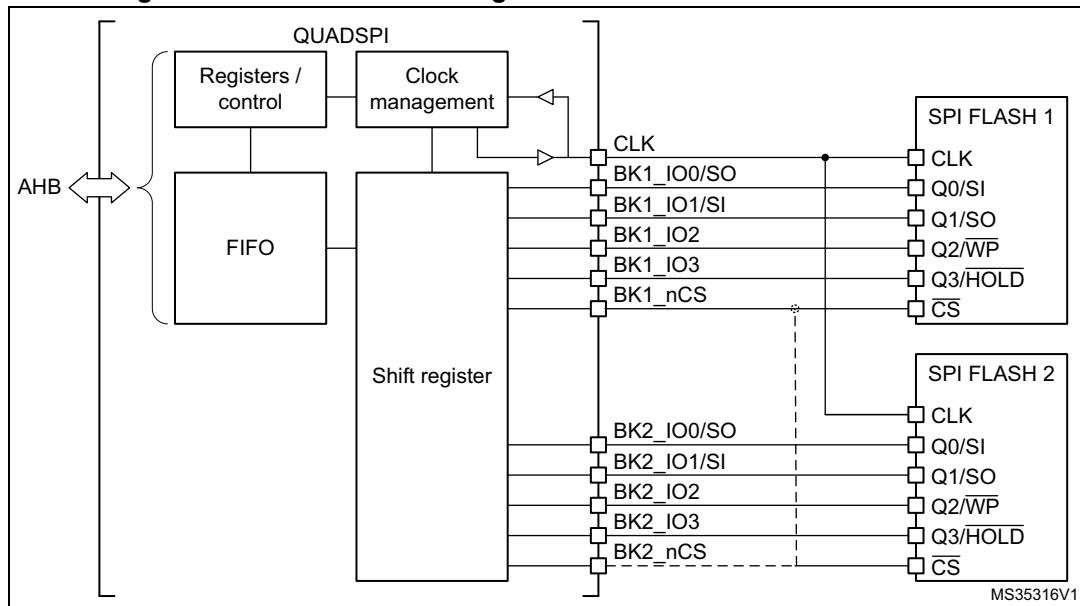
- Three functional modes: indirect, status-polling, and memory-mapped
- Dual-flash mode, where 8 bits can be sent/received simultaneously by accessing two Flash memories in parallel.
- SDR and DDR support
- Fully programmable opcode for both indirect and memory mapped mode
- Fully programmable frame 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

### 12.3 QUADSPI functional description

#### 12.3.1 QUADSPI block diagram

Figure 60. QUADSPI block diagram when dual-flash mode is disabled



**Figure 61. QUADSPI block diagram when dual-flash mode is enabled**

### 12.3.2 QUADSPI pins

*Table 81* 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 81. QUADSPI pins**

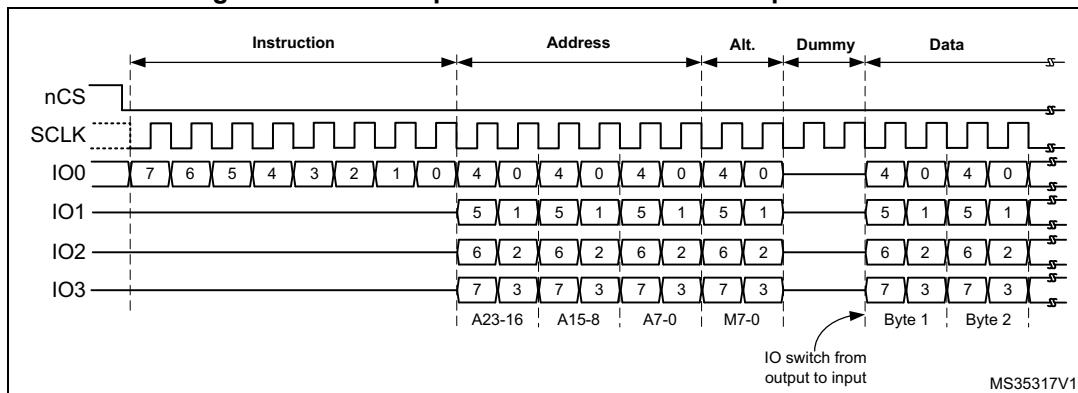
Signal name	Signal type	Description
CLK	Digital output	Clock to FLASH 1 and FLASH 2
BK1_IO0/SO	Digital input/output	Bidirectional IO in dual/quad modes or serial output in single mode, for FLASH 1
BK1_IO1/SI	Digital input/output	Bidirectional IO in dual/quad modes or serial input in single mode, for 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.

### 12.3.3 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 62. 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.

### 12.3.4 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

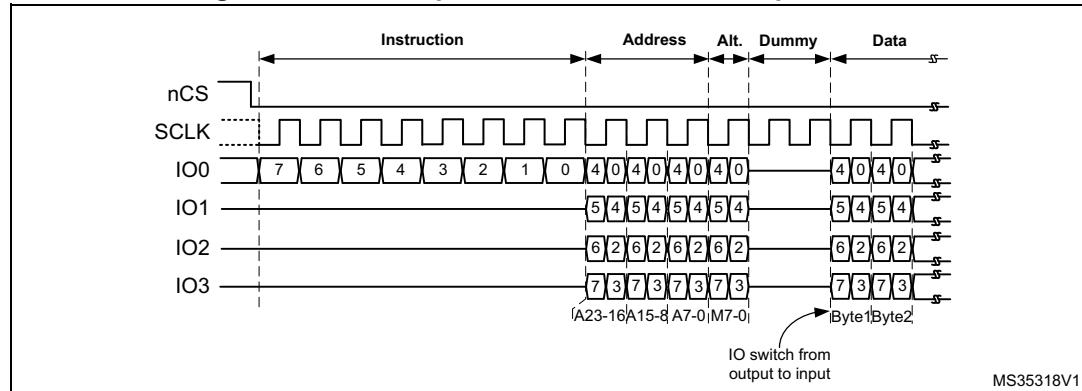
When the DDRM bit (QUADSPI\_CCR[31]) is set to 1, the QUADSPI operates in double data rate (DDR) mode.

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 63. 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.

### 12.3.5 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  $\leq$  11). Thus, when FTHRES  $\geq$  13, 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 11  $<$  FLEVEL  $<$  FTHRES.

### 12.3.6 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.

### 12.3.7 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.

### 12.3.8 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).

### 12.3.9 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.

### 12.3.10 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.

#### 12.3.11 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.

### 12.3.12 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).

### 12.3.13 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.

### 12.3.14 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.

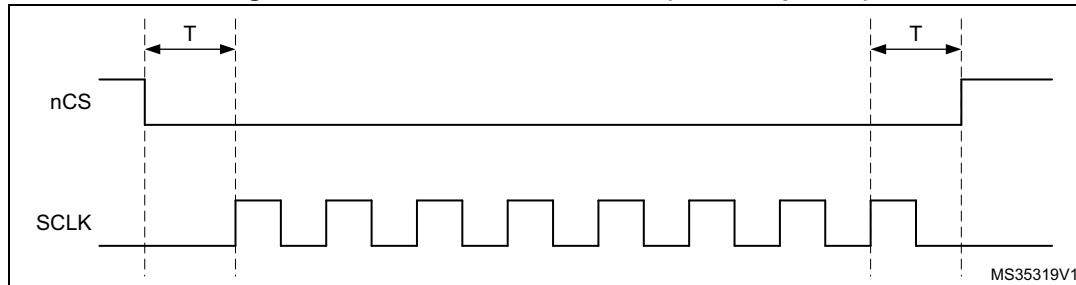
*Note:* Some Flash memories might misbehave if a write operation to a status registers is aborted.

### 12.3.15 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.

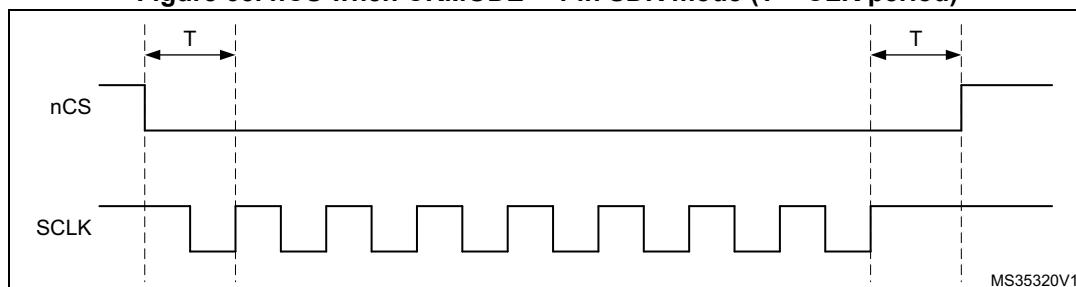
When CKMODE = 0 (“mode0”, where CLK stays low when no operation is in progress) nCS falls one CLK cycle before an operation first rising CLK edge, and nCS rises one CLK cycle after the operation final rising CLK edge, as shown in [Figure 64](#).

**Figure 64. nCS when CKMODE = 0 (T = CLK period)**



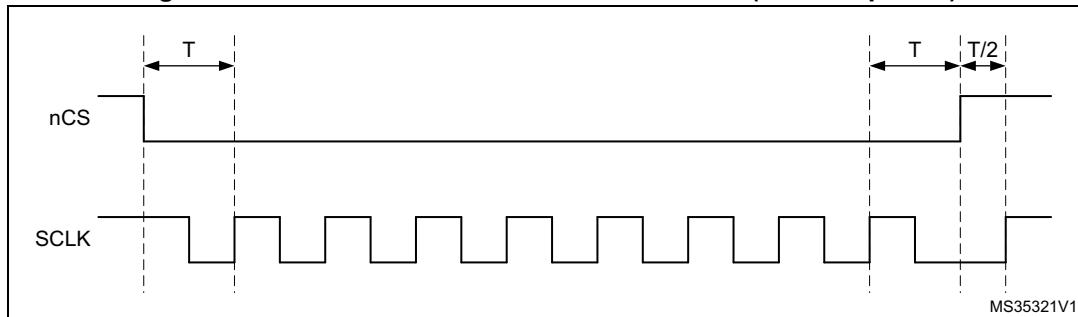
When CKMODE=1 (“mode3”, where CLK goes high when no operation is in progress) and DDRM=0 (SDR mode), nCS still falls one CLK cycle before an operation first rising CLK edge, and nCS rises one CLK cycle after the operation final rising CLK edge, as shown in [Figure 65](#).

**Figure 65. nCS when CKMODE = 1 in SDR mode (T = CLK period)**



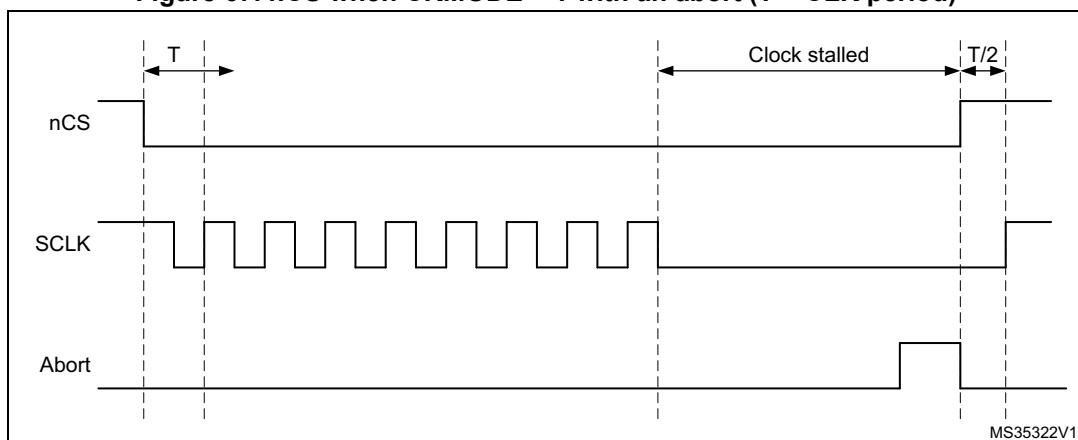
When CKMODE = 1 (“mode3”) and DDRM = 1 (DDR mode), nCS falls one CLK cycle before an operation first rising CLK edge, and nCS rises one CLK cycle after the operation final active rising CLK edge, as shown in [Figure 66](#). Because DDR operations must finish with a falling edge, CLK is low when nCS rises, and CLK rises back up one half of a CLK cycle afterwards.

**Figure 66. 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 67](#).

**Figure 67. 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.

## 12.4 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 82. 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

## 12.5 QUADSPI registers

### 12.5.1 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								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				FSEL	DFM	Res.	SSSHIFT	TCEN	DMAEN	ABORT	EN	
			rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	w1s

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

## 12.5.2 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.	FSIZE																		
															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			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.

### 12.5.3 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.

### 12.5.4 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										
											w1o	w1o		w1o	w1o

Bits 31: 4 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

### 12.5.5 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.

**12.5.6 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		Res.	DCYC[4:0]				ABSIZE			
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		ADSIZE		ADMODE		IMODE		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 12.3.12: Sending the instruction only once on page 336](#). 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.

**12.5.7 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

## 12.5.8 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.

## 12.5.9 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].

### 12.5.10 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.

### 12.5.11 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.

### 12.5.12 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.

### 12.5.13 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.

## 12.5.14 QUADSPI register map

Table 83. QUADSPI 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	
0x0000	QUADSPI_CR																											
		PRESCALER[7:0]																										
0x0004	QUADSPI_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.	
		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
0x0008	QUADSPI_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.	
		Reset value																										
0x000C	QUADSPI_FCR	Res.	Res.	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																										
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	DHIC	Res.	SIO0	FMODE[1:0]	DMODE[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																DCYC[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
0x001C	QUADSPI_ABR																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
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
0x0030	QUADSPI_LPTR																TIMEOUT[15:0]											
		Reset value															0	0	0	0	0	0	0	0	0	0	0	0

Refer to [Section 2.2.2](#) for the register boundary addresses.

## 13 Analog-to-digital converter (ADC)

### 13.1 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.

### 13.2 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

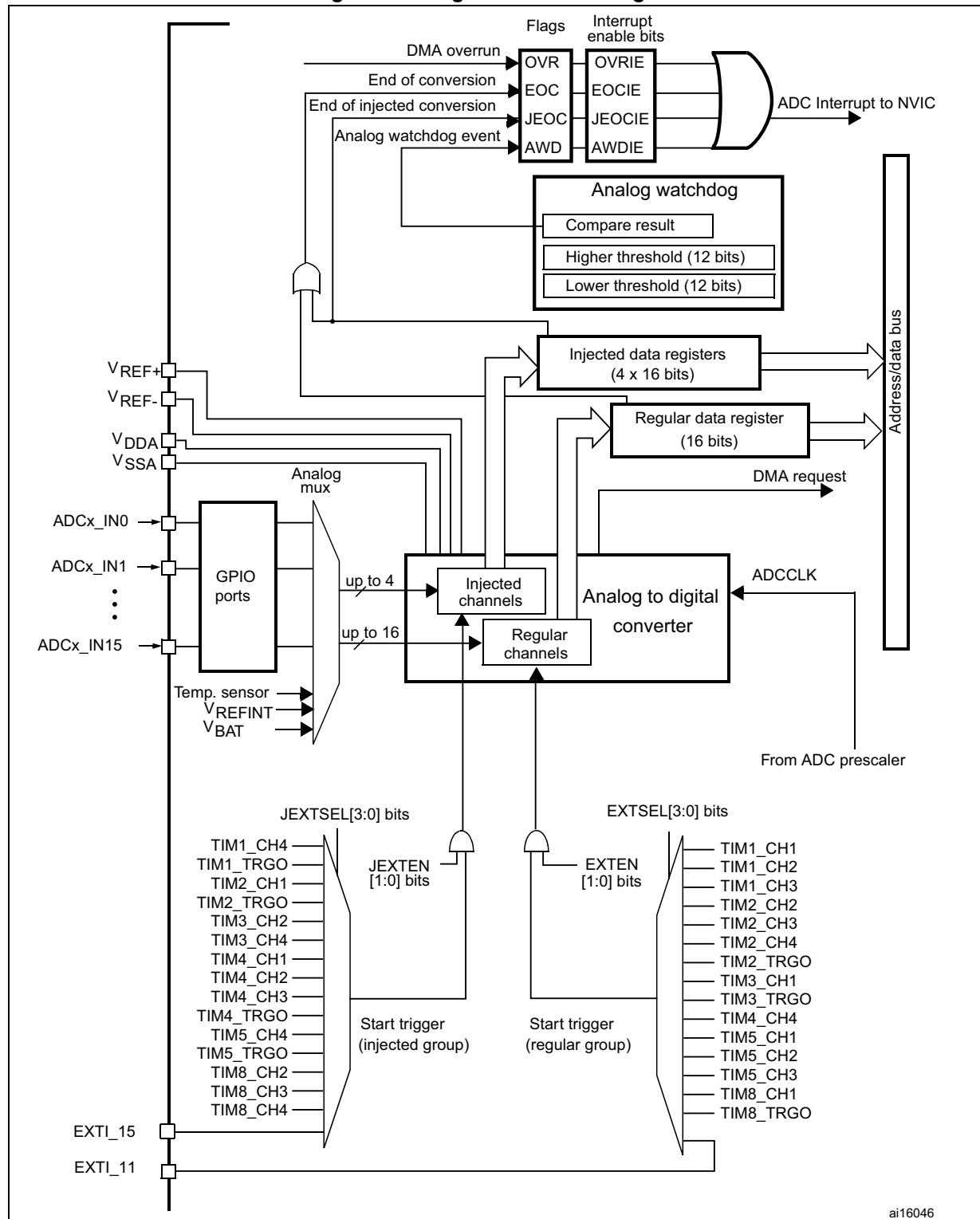
*Figure 68* shows the block diagram of the ADC.

Note:  $V_{REF-}$ , if available (depending on package), must be tied to  $V_{SSA}$ .

### 13.3 ADC functional description

*Figure 68* shows a single ADC block diagram and *Table 84* gives the ADC pin description.

Figure 68. Single ADC block diagram



**Table 84. 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

### 13.3.1 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.

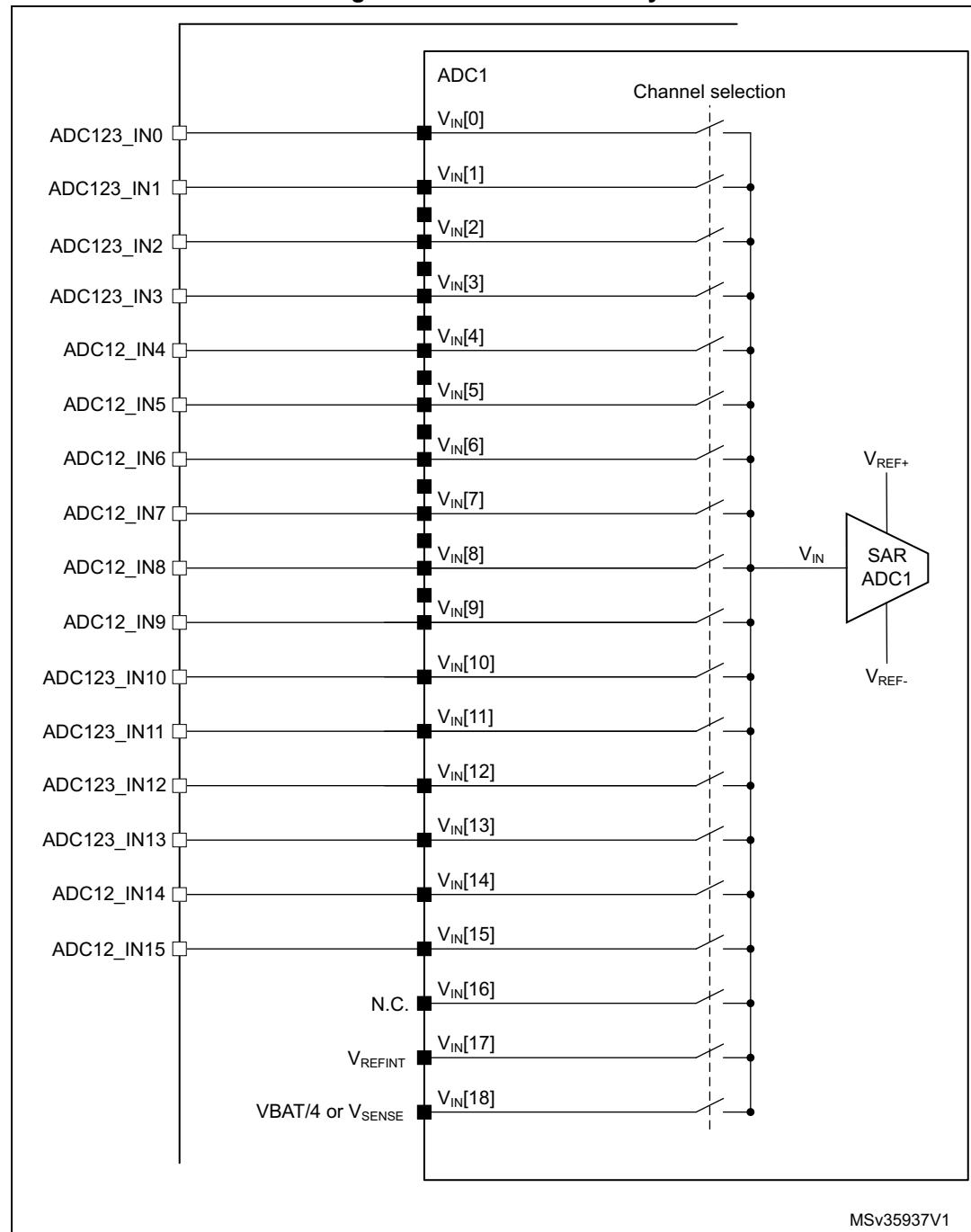
Conversion starts when either the SWSTART or the JSWSTART bit is set.

You 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  $\mu\text{A}$ ).

### 13.3.2 ADC1/2 and ADC3 connectivity

ADC1, ADC2 and ADC3 are tightly coupled and share some external channels as described in [Figure 69](#), [Figure 70](#) and [Figure 71](#).

**Figure 69. ADC1 connectivity**



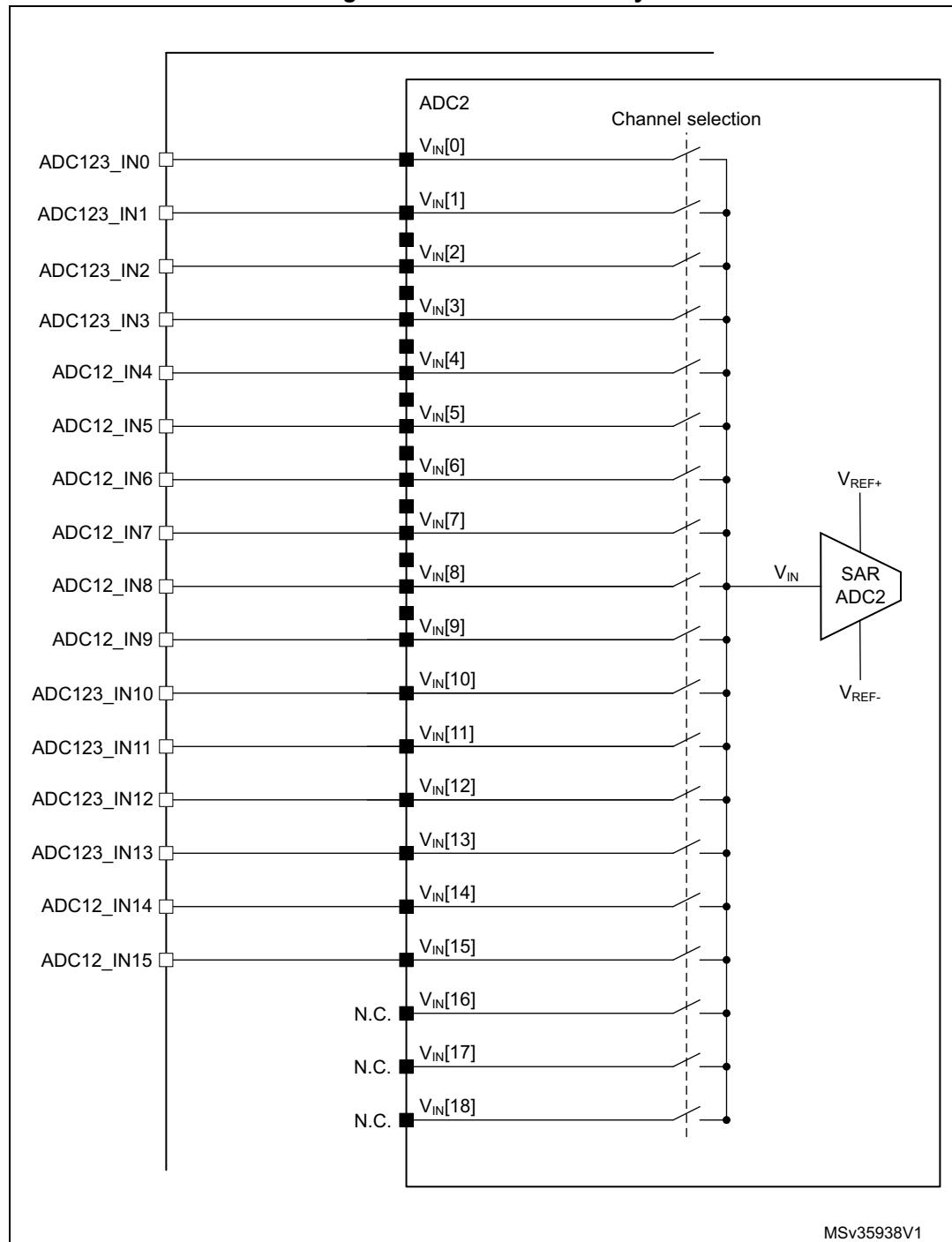
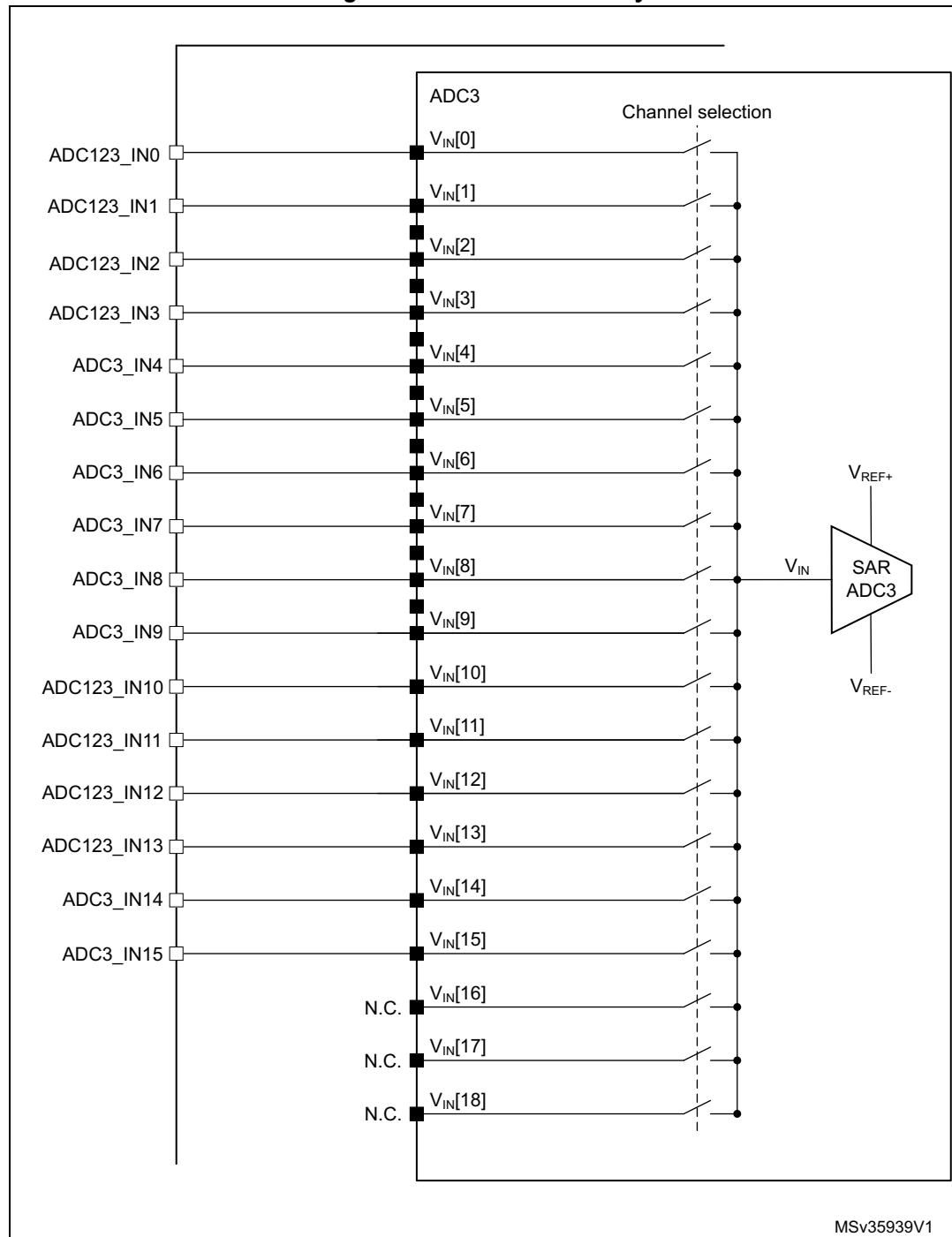
**Figure 70. ADC2 connectivity**

Figure 71. ADC3 connectivity



### 13.3.3 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).

### 13.3.4 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.

### 13.3.5 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.

### 13.3.6 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:

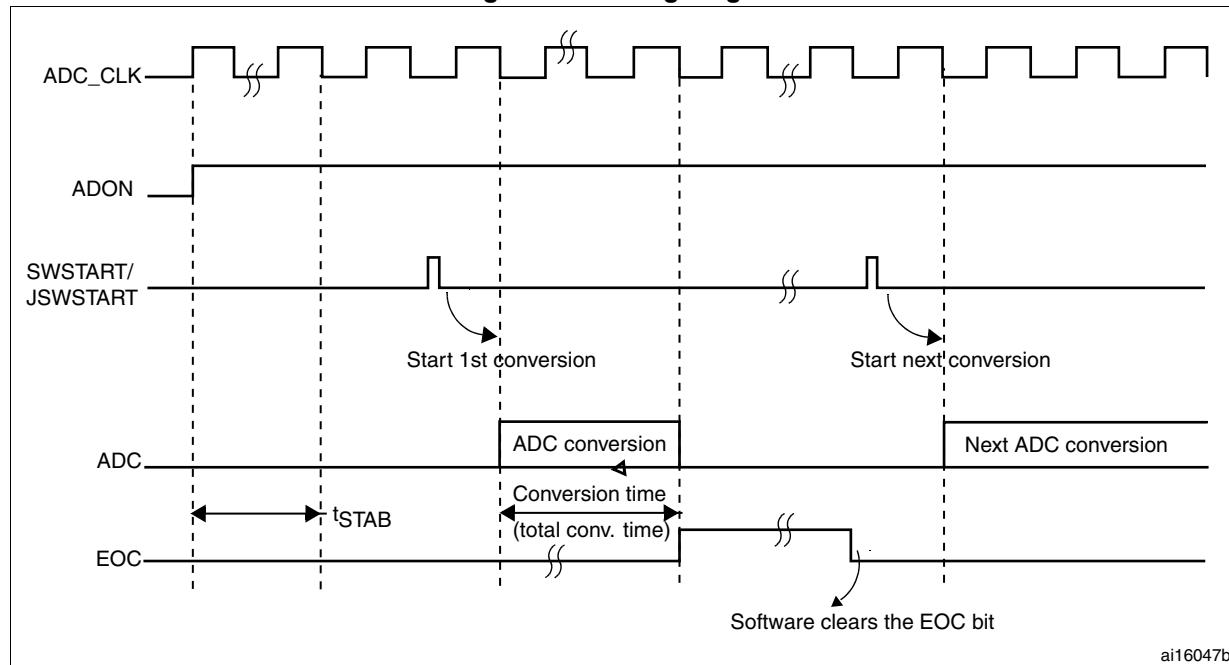
- If a regular group of channels was converted:
  - The last converted data are stored into the 16-bit ADC\_DR register
  - The EOC (end of conversion) flag is set
  - An interrupt is generated if the EOCIE bit is set

**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.*

### 13.3.7 Timing diagram

As shown in [Figure 72](#), 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 72. Timing diagram



### 13.3.8 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 85](#) shows how the ADC\_CR1 register should be configured to enable the analog watchdog on one or more channels.

Figure 73. Analog watchdog's guarded area

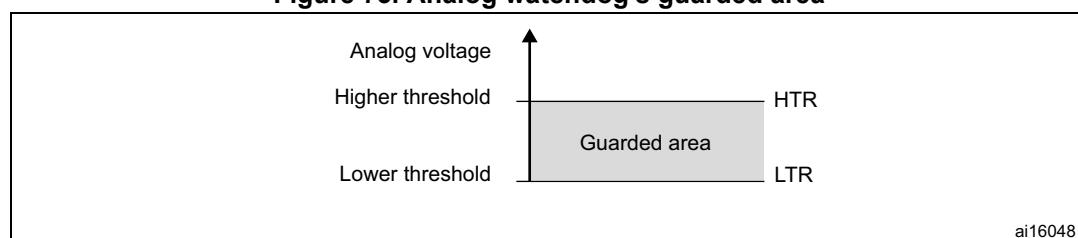


Table 85. Analog watchdog channel selection

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 85. 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 <sup>(1)</sup> injected channel	1	0	1
Single <sup>(1)</sup> regular channel	1	1	0
Single <sup>(1)</sup> regular or injected channel	1	1	1

1. Selected by the AWDCH[4:0] bits

### 13.3.9 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.

### 13.3.10 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 74* 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

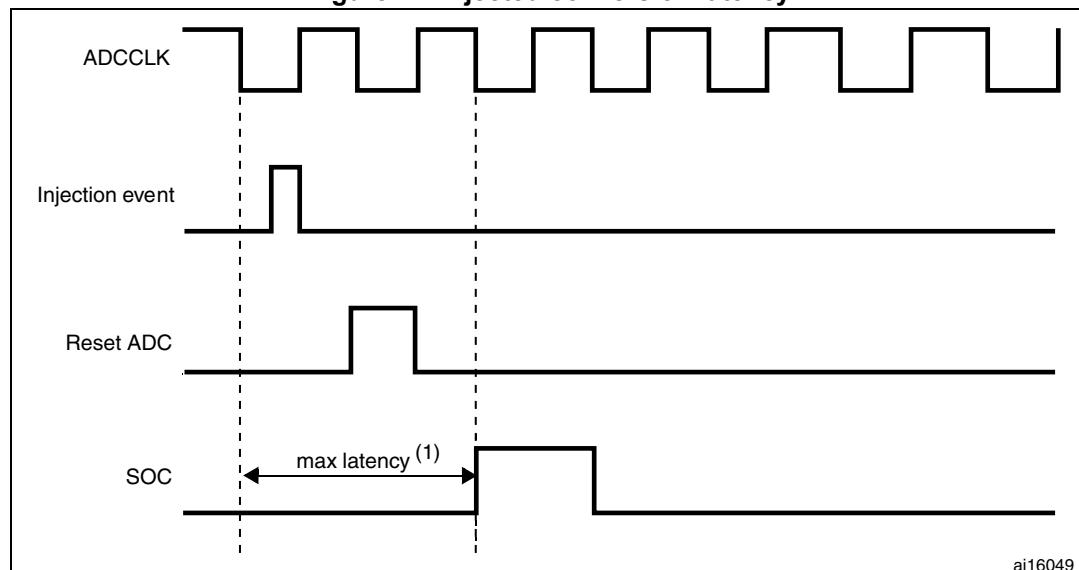
If the JAUTO bit is set, then the channels in the injected group are automatically converted after the regular group of channels. This can be used to convert a sequence of up to 20 conversions programmed in the ADC\_SQRx and ADC\_JSQR registers.

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 74. Injected conversion latency**



ai16049

1. The maximum latency value can be found in the electrical characteristics of the STM32F446xx datasheets.

### 13.3.11 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.*

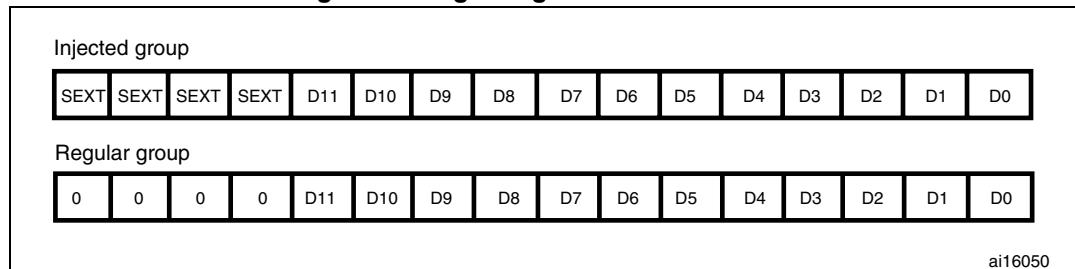
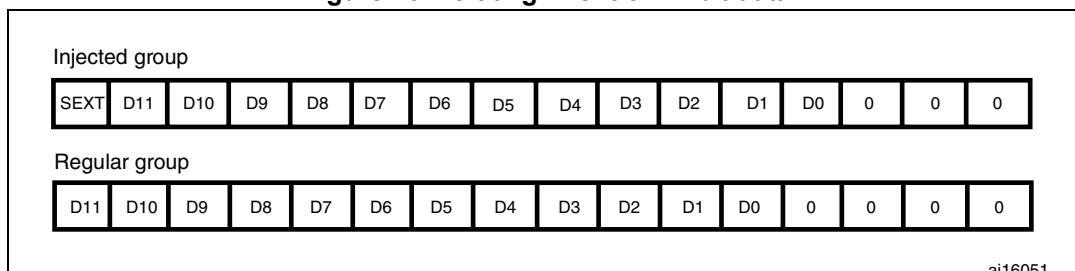
*Discontinuous mode must be enabled only for the conversion of one group.*

## 13.4 Data alignment

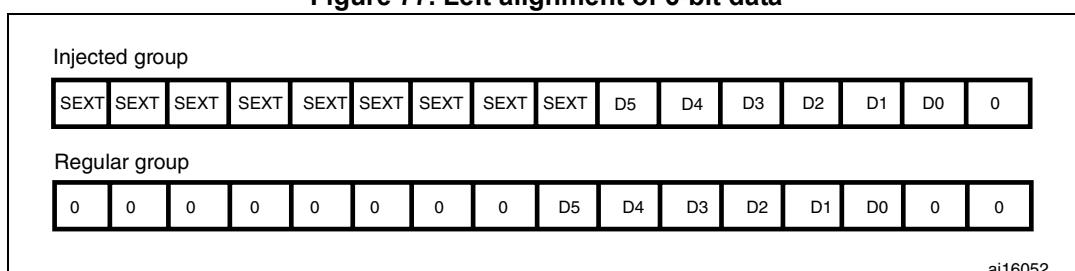
The ALIGN bit in the ADC\_CR2 register selects the alignment of the data stored after conversion. Data can be right- or left-aligned as shown in [Figure 75](#) and [Figure 76](#).

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 75. Right alignment of 12-bit data****Figure 76. 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 77](#).

**Figure 77. Left alignment of 6-bit data**

## 13.5

## 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:

$$T_{\text{conv}} = 3 + 12 = 15 \text{ cycles} = 0.5 \mu\text{s with APB2 at 60 MHz}$$

## 13.6 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 86](#) provides the correspondence between the EXTEN[1:0] and JEXTEN[1:0] values and the trigger polarity.

**Table 86. 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 87](#) gives the possible external trigger for regular conversion.

**Table 87. External trigger for regular channels**

Source	Type	EXTSEL[3:0]
TIM1_CH1 event	Internal signal from on-chip timers	0000
TIM1_CH2 event		0001
TIM1_CH3 event		0010
TIM2_CH2 event		0011
TIM2_CH3 event		0100
TIM2_CH4 event		0101
TIM2_TRGO event		0110
TIM3_CH1 event		0111
TIM3_TRGO event		1000
TIM4_CH4 event		1001
TIM5_CH1 event		1010
TIM5_CH2 event		1011
TIM5_CH3 event		1100
TIM8_CH1 event		1101
TIM8_TRGO event		1110
EXTI line11	External pin	1111

*Table 88* gives the possible external trigger for injected conversion.

**Table 88. External trigger for injected channels**

Source	Connection type	JEXTSEL[3:0]
TIM1_CH4 event	Internal signal from on-chip timers	0000
TIM1_TRGO event		0001
TIM2_CH1 event		0010
TIM2_TRGO event		0011
TIM3_CH2 event		0100
TIM3_CH4 event		0101
TIM4_CH1 event		0110
TIM4_CH2 event		0111
TIM4_CH3 event		1000
TIM4_TRGO event		1001
TIM5_CH4 event		1010
TIM5_TRGO event		1011
TIM8_CH2 event		1100
TIM8_CH3 event		1101
TIM8_CH4 event		1110
EXTI line15	External pin	1111

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.*

## 13.7

## 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

## 13.8 Data management

### 13.8.1 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.

### 13.8.2 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.

### 13.8.3 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.

## 13.9 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 78](#)).

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.*

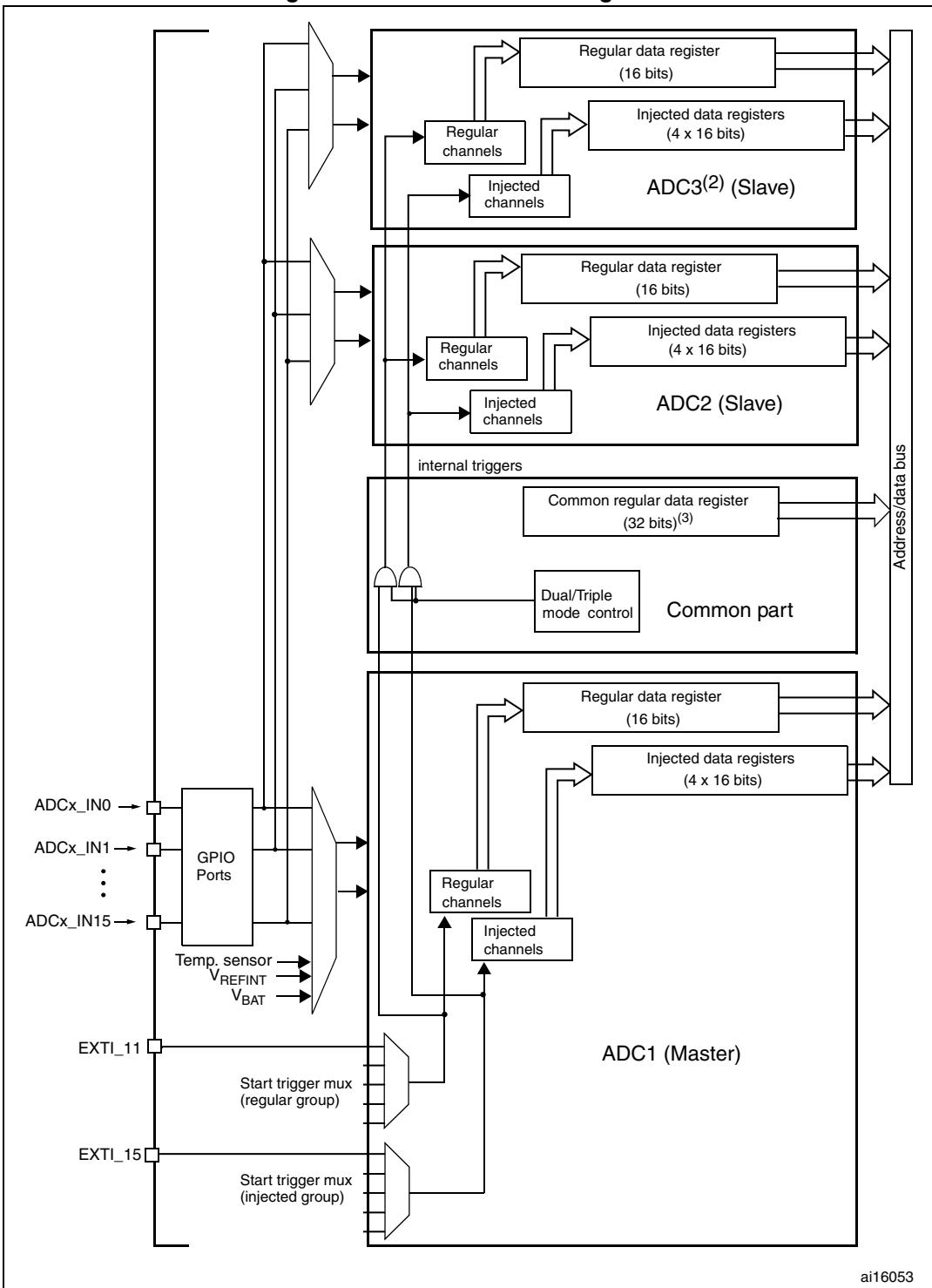
The four possible modes below are implemented:

- 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 78. Multi ADC block diagram<sup>(1)</sup>

ai16053

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.

### 13.9.1 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.*

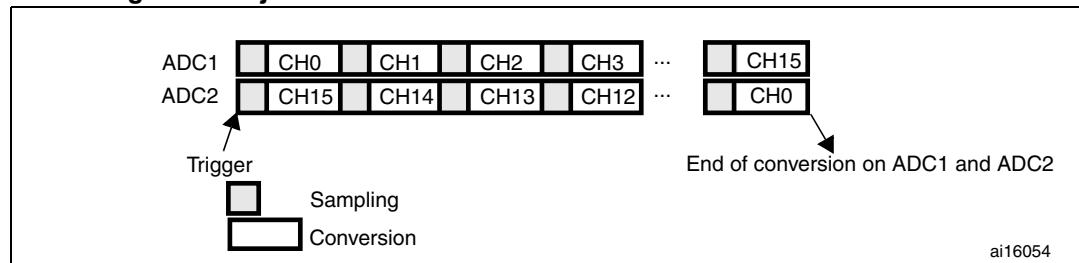
*Regular conversions can be performed on one or all ADCs. In that case, they are independent of each other and are interrupted when an injected event occurs. They are resumed at the end of the injected conversion group.*

### 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 79. Injected simultaneous mode on 4 channels: dual ADC mode**

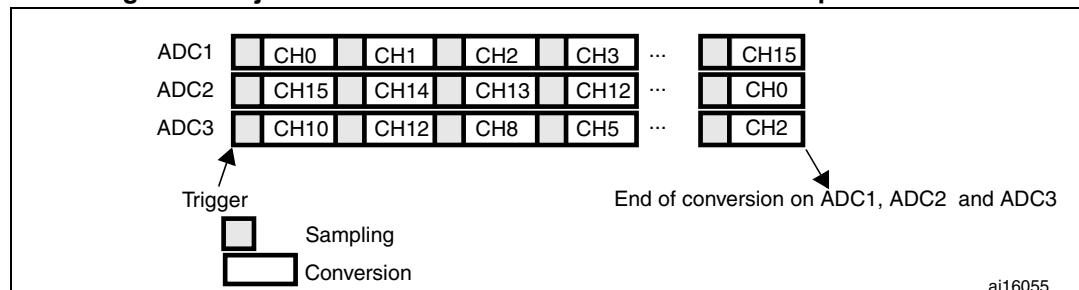


### Triple ADC mode

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 80. Injected simultaneous mode on 4 channels: triple ADC mode**



### 13.9.2 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.*

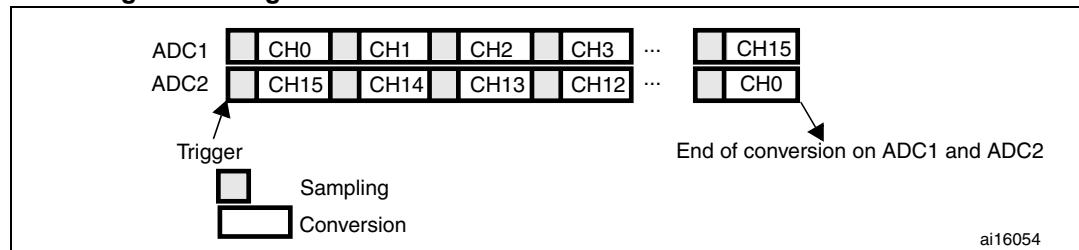
*Injected conversions must be disabled.*

### 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 81. Regular simultaneous mode on 16 channels: dual ADC mode**



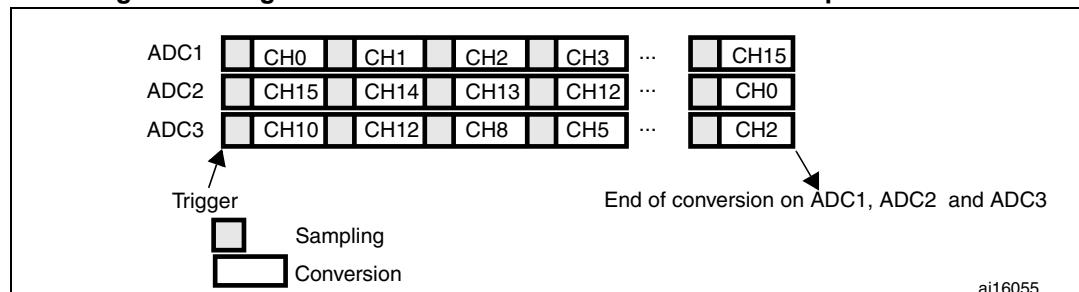
ai16054

### 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 82. Regular simultaneous mode on 16 channels: triple ADC mode**



ai16055

### 13.9.3 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

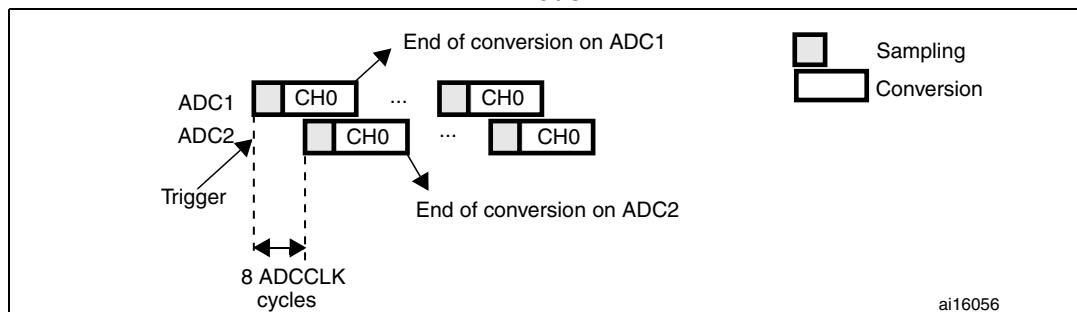
The minimum delay which separates 2 conversions in interleaved mode is configured in the DELAY bits in the ADC\_CCR register. However, an ADC cannot start a conversion if the complementary ADC is still sampling its input (only one ADC can sample the input signal at a given time). In this case, the delay 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 83. 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.

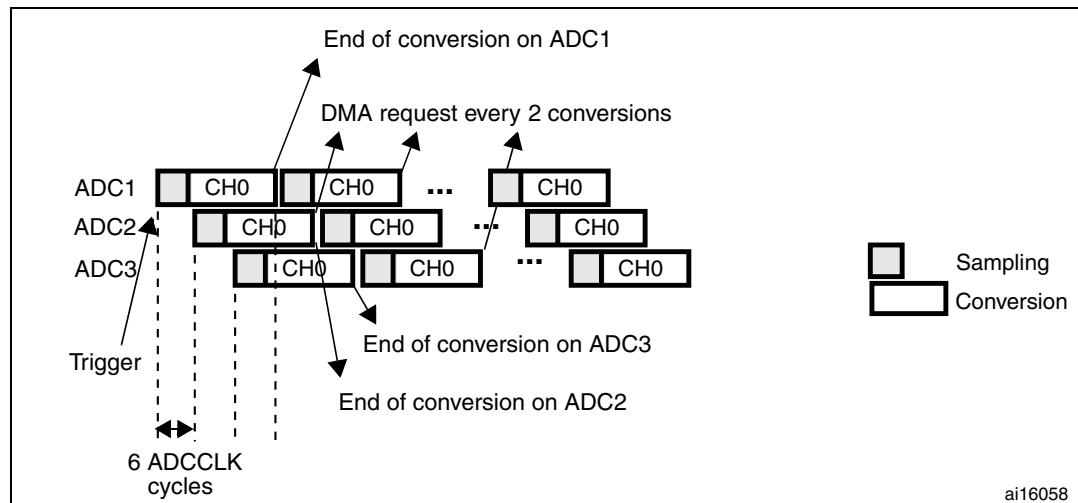
**Note:** *If the conversion sequence is interrupted (for instance when DMA end of transfer occurs), the multi-ADC sequencer must be reset by configuring it in independent mode first (bits DUAL[4:0] = 00000) before reprogramming the interleaved mode.*

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 84. Interleaved mode on 1 channel in continuous conversion mode: triple ADC mode**



### 13.9.4 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

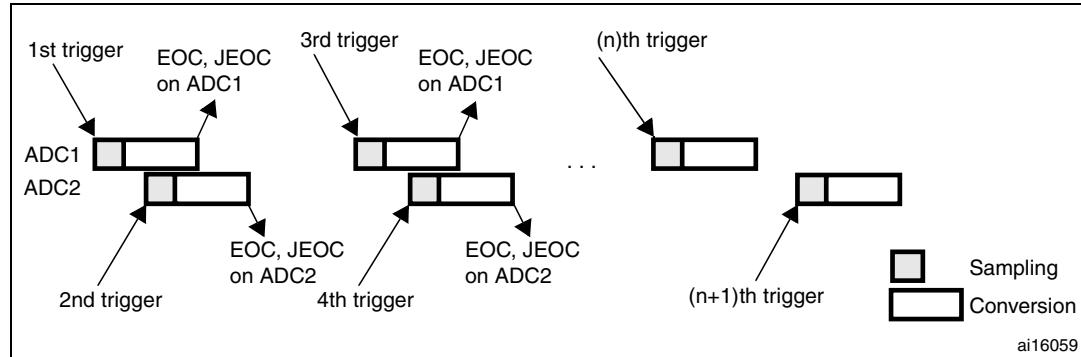
- When the 1st trigger occurs, all injected ADC1 channels in the group are converted
- When the 2nd trigger occurs, all injected ADC2 channels in the group are converted
- and so on

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 85. Alternate trigger: injected group of each ADC**



If the injected discontinuous mode is enabled for both ADC1 and ADC2:

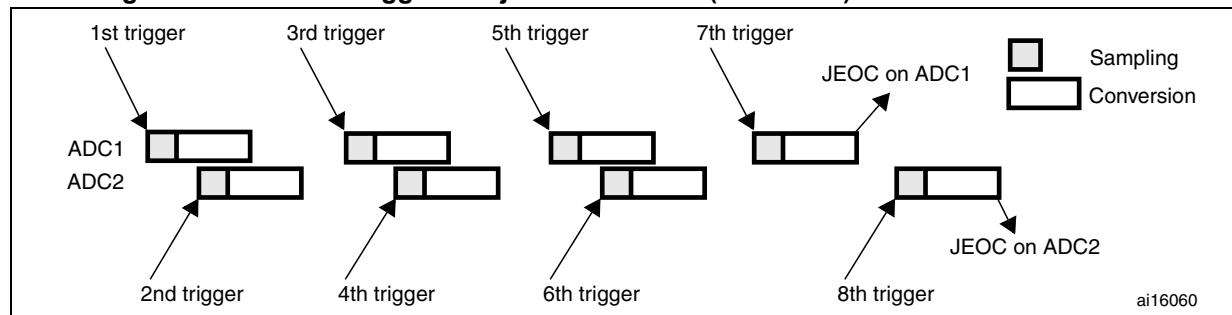
- When the 1st trigger occurs, the first injected ADC1 channel is converted.
- When the 2nd trigger occurs, the first injected ADC2 channel are converted
- and so on

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 86. 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

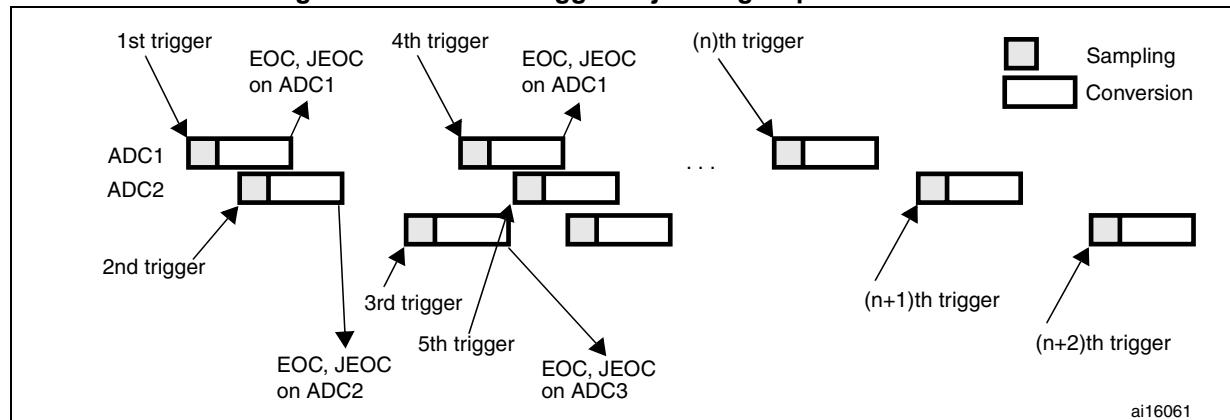
A JEOC interrupt, if enabled, is generated after all injected ADC1 channels in the group have been converted.

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 87. Alternate trigger: injected group of each ADC**



### 13.9.5 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.*

### 13.9.6 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 88](#) shows the behavior of an alternate trigger interrupting a simultaneous regular conversion.

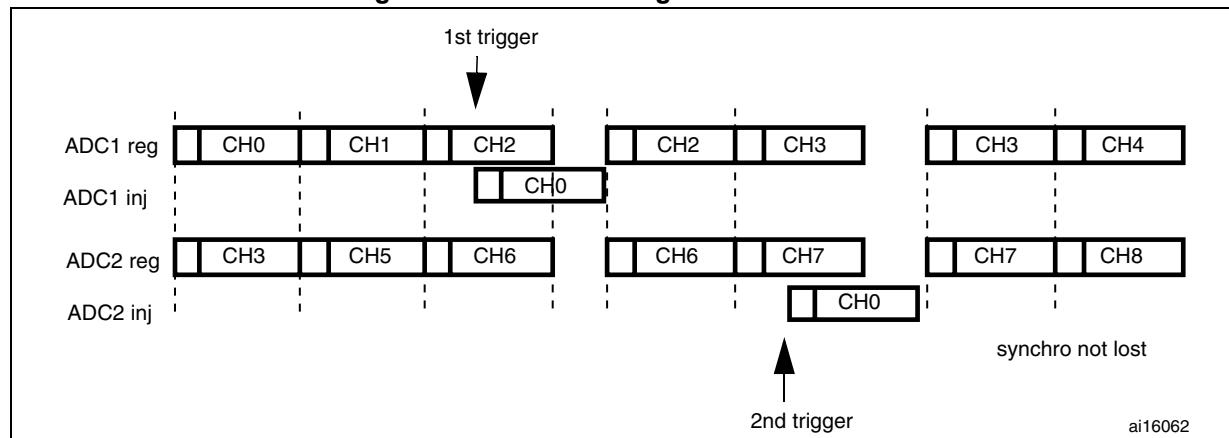
The injected alternate conversion is immediately started after the injected event. If regular conversion is already running, in order to ensure synchronization after the injected conversion, the regular conversion of all (master/slave) ADCs is stopped and resumed synchronously at the end of the injected 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.

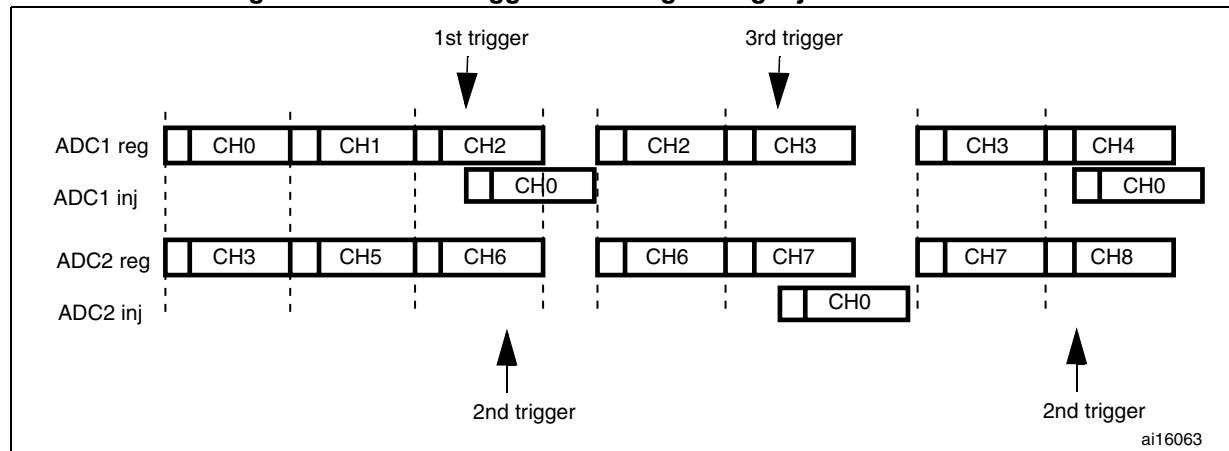
If the conversion sequence is interrupted (for instance when DMA end of transfer occurs), the multi-ADC sequencer must be reset by configuring it in independent mode first (bits DUAL[4:0] = 00000) before reprogramming the interleaved mode.

**Figure 88. Alternate + regular simultaneous**



If a trigger occurs during an injected conversion that has interrupted a regular conversion, it is ignored. [Figure 89](#) shows the behavior in this case (2nd trigger is ignored).

**Figure 89. Case of trigger occurring during injected conversion**



## 13.10 Temperature sensor

The temperature sensor can be used to measure the ambient temperature ( $T_A$ ) of the device.

- On STM32F446xx devices, 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 90](#) shows the block diagram of the temperature sensor.

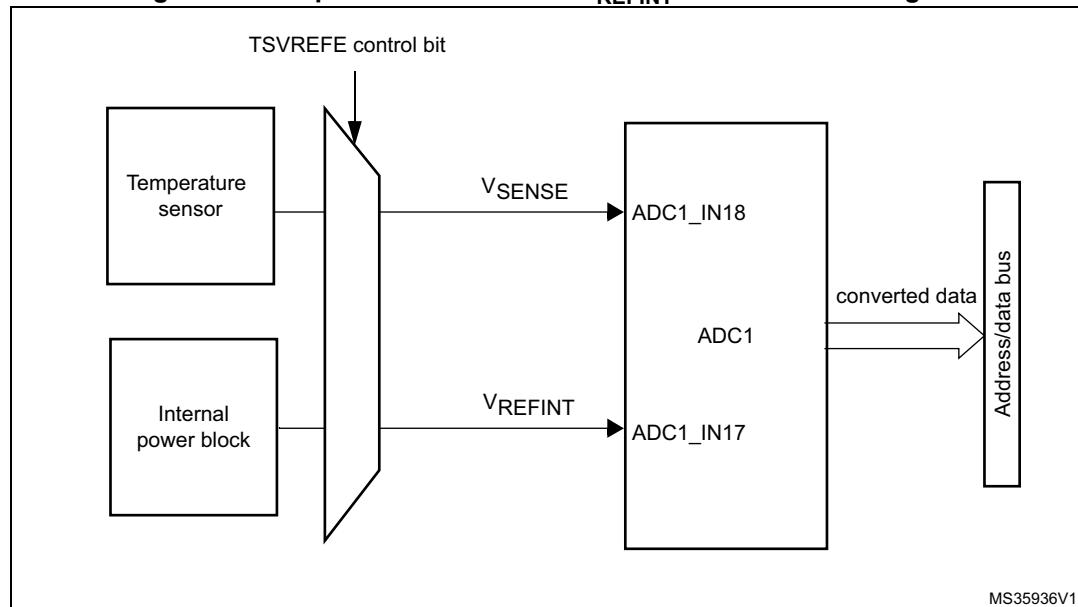
When not in use, the sensor can be put in power down mode.

**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 90. Temperature sensor and V<sub>REFINT</sub> channel block diagram**



1. V<sub>SENSE</sub> 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<sub>SENSE</sub> 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:

- V<sub>25</sub> = V<sub>SENSE</sub> value for 25° C
- Avg\_Slope = average slope of the temperature vs. V<sub>SENSE</sub> curve (given in mV/°C or µV/°C)

Refer to the datasheet electrical characteristics section for the actual values of V<sub>25</sub> 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.

## 13.11 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.*

## 13.12 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 89. 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

## 13.13 ADC registers

Refer to [Section 1.1 on page 51](#) 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).

### 13.13.1 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

### 13.13.2 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

### 13.13.3 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	
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 CC1 event

0001: Timer 1 CC2 event

0010: Timer 1 CC3 event

0011: Timer 2 CC2 event

0100: Timer 2 CC3 event

0101: Timer 2 CC4 event

0110: Timer 2 TRGO event

0111: Timer 3 CC1 event

1000: Timer 3 TRGO event

1001: Timer 4 CC4 event

1010: Timer 5 CC1 event

1011: Timer 5 CC2 event

1100: Timer 5 CC3 event

1101: Timer 8 CC1 event

1110: Timer 8 TRGO event

1111: EXTI line 11

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

*Note: 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 CC4 event

0001: Timer 1 TRGO event

0010: Timer 2 CC1 event

0011: Timer 2 TRGO event

0100: Timer 3 CC2 event

0101: Timer 3 CC4 event

0110: Timer 4 CC1 event

0111: Timer 4 CC2 event

1000: Timer 4 CC3 event

1001: Timer 4 TRGO event

1010: Timer 5 CC4 event

1011: Timer 5 TRGO event

1100: Timer 8 CC2 event

1101: Timer 8 CC3 event

1110: Timer 8 CC4 event

1111: EXTI line15

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 75](#) and [Figure 76](#).

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.

Note: 0: Disable ADC conversion and go to power down mode  
1: Enable ADC

### 13.13.4 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]	
					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
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

### 13.13.5 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]			SMP5[2:1]	
		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

### 13.13.6 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											

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.

### 13.13.7 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											

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.

### 13.13.8 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
LT[11:0]															
Res.	Res.	Res.	Res.	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.

### 13.13.9 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]			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ16_0															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
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

### 13.13.10 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

### 13.13.11 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

### 13.13.12 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.

### 13.13.13 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 75](#) and [Figure 76](#).

### 13.13.14 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.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

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

Bits 15:0 **DATA[15:0]**: 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 75](#) and [Figure 76](#).

### 13.13.15 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
15	14	13	12	11	10	9	8	7	6	r	r	r	r	r	r
ADC3															
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.

### 13.13.16 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<sub>REFINT</sub> enable

This bit is set and cleared by software to enable/disable the temperature sensor and the V<sub>REFINT</sub> channel.

0: Temperature sensor and V<sub>REFINT</sub> channel disabled  
1: Temperature sensor and V<sub>REFINT</sub> 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<sub>BAT</sub> enable

This bit is set and cleared by software to enable/disable the V<sub>BAT</sub> channel.

0: V<sub>BAT</sub> channel disabled  
1: V<sub>BAT</sub> channel enabled

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<sub>ADCCLK</sub>

0001: 6 \* T<sub>ADCCLK</sub>

0010: 7 \* T<sub>ADCCLK</sub>

...

1111: 20 \* T<sub>ADCCLK</sub>

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.*

### 13.13.17 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](#).

### 13.13.18 ADC register map

The following table summarizes the ADC registers.

Table 90. 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 91. 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.						
	Reset value																
0x04	ADC_CR1	Res.	Res.	Res.	Res.	Res.	Res.	RES[1:0]	AWDEN	JAWDEN	Res.	Res.	Res.	Res.	DISC NUM [2:0]	Res.	Res.
	Reset value							0 0 0	0	0					0 0 0	0	0

**Table 91. ADC register map and reset values for each ADC (continued)**

**Table 92. 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	<b>ADC_CSR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OVR	STRT	JSTRT	JEOC	EOC	AWD	Res.	Res.	Res.	STRT	JSTRT	JEOC	EOC	AWD	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				
0x04	<b>ADC_CCR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TSVREFE	VBATE	Res.	Res.	Res.	Res.	DMA[1:0]	DDS	OVR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	MULTI [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
0x08	<b>ADC_CDR</b>	Regular DATA2[15:0]										Regular DATA1[15:0]										DELAY [3:0]																
	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			

Refer to [Section 2.2.2 on page 56](#) for the register boundary addresses.

## 14 Digital-to-analog converter (DAC)

### 14.1 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.

### 14.2 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 91* shows the block diagram of a DAC channel and *Table 93* gives the pin description.

Figure 91. DAC channel block diagram

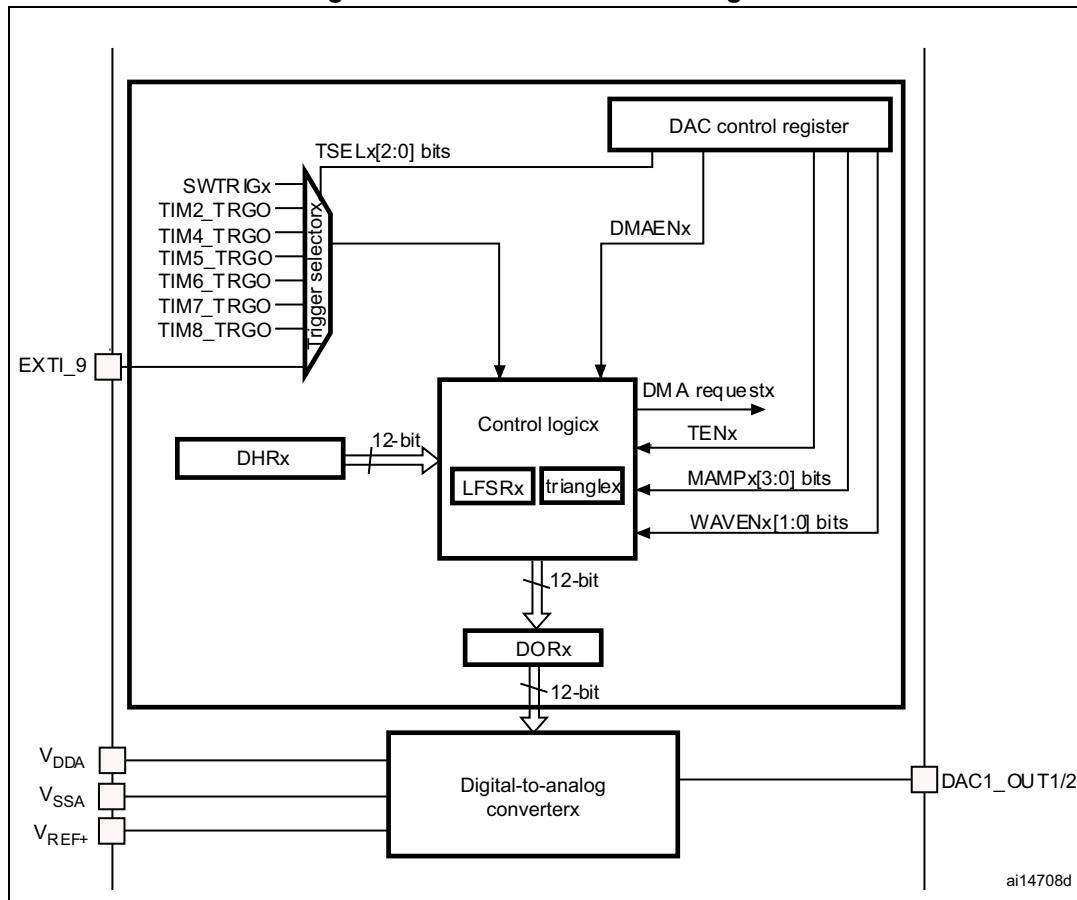


Table 93. DAC pins

Name	Signal type	Remarks
V <sub>REF+</sub>	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 <sub>DDA</sub>	Input, analog supply	Analog power supply
V <sub>SSA</sub>	Input, analog supply ground	Ground for analog power supply
DAC <sub>_OUTx</sub>	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<sub>\_OUTx</sub>). In order to avoid parasitic consumption, the PA4 or PA5 pin should first be configured to analog (AIN).

## 14.3 DAC functional description

### 14.3.1 DAC channel enable

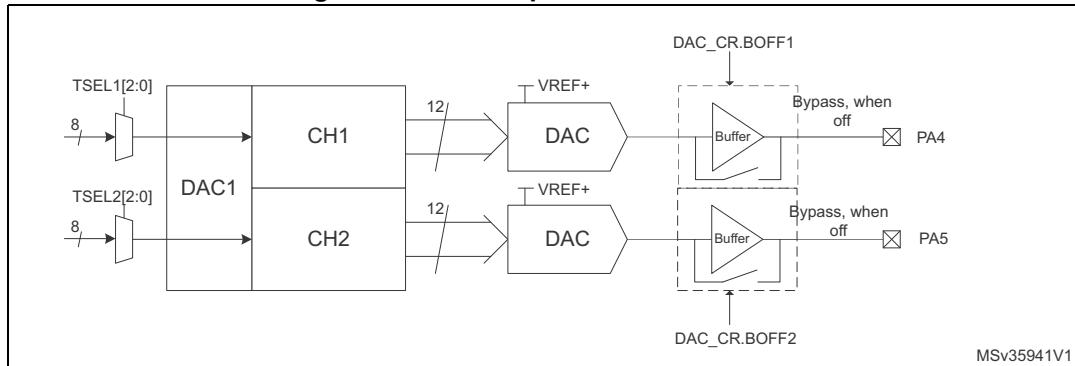
Each DAC channel can be powered on by setting its corresponding ENx bit in the DAC\_CR register. The DAC channel is then enabled after a startup time  $t_{\text{WAKEUP}}$ .

**Note:** The ENx bit enables the analog DAC Channelx macrocell only. The DAC Channelx digital interface is enabled even if the ENx bit is reset.

### 14.3.2 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 92. DAC output buffer connection**

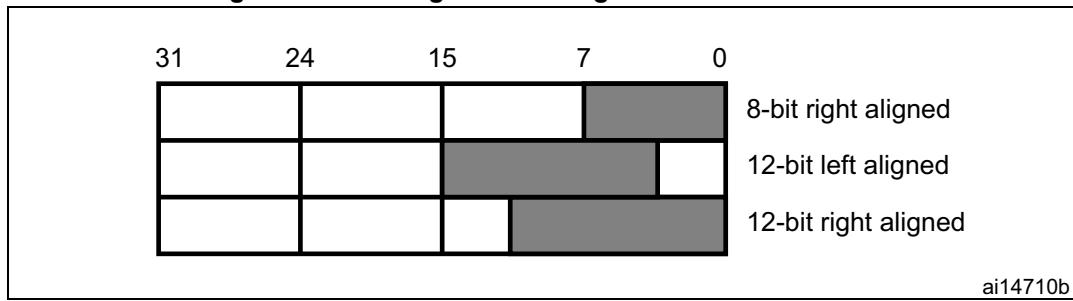


### 14.3.3 DAC data format

Depending on the selected configuration mode, the data have to be written into the specified register as described below:

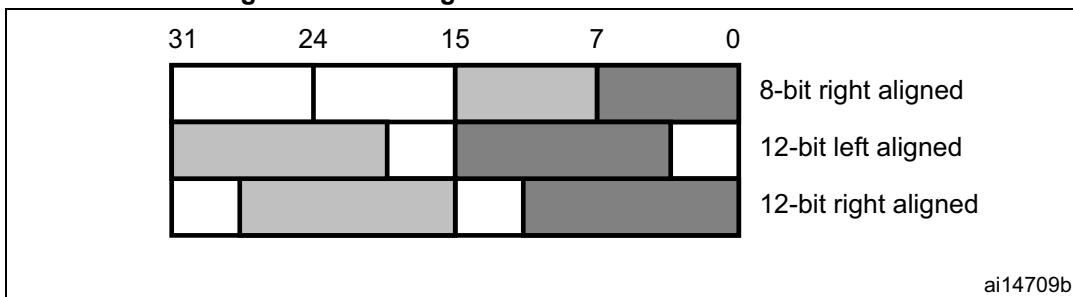
- Single DAC channelx, there are three possibilities:
  - 8-bit right alignment: the software has to load data into the DAC\_DHR8Rx [7:0] bits (stored into the DHRx[11:4] bits)
  - 12-bit left alignment: the software has to load data into the DAC\_DHR12Lx [15:4] bits (stored into the DHRx[11:0] bits)
  - 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 93. Data registers in single DAC channel mode**

- 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 94. Data registers in dual DAC channel mode**

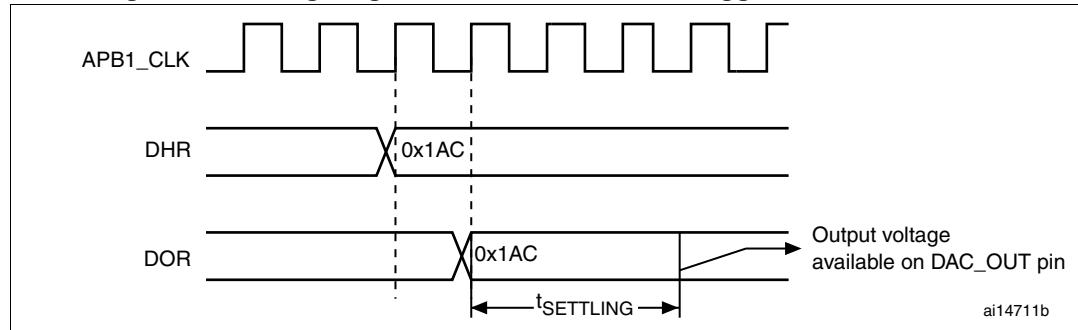
#### 14.3.4 DAC conversion

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 95. Timing diagram for conversion with trigger disabled TEN = 0**



#### 14.3.5 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:

$$\text{DACOutput} = V_{REF} \times \frac{\text{DOR}}{4096}$$

#### 14.3.6 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 94](#).

**Table 94. 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.*

#### 14.3.7 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.

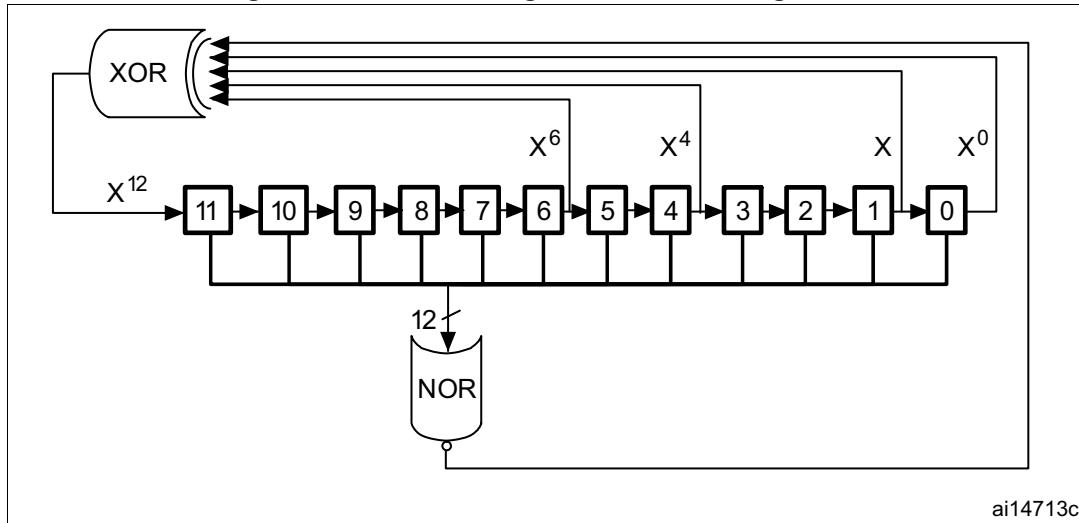
The software should clear the DMAUDRx flag by writing “1”, clear the DMAEN bit of the used DMA stream and re-initialize both DMA and DAC channelx to restart the transfer correctly. The software should modify the DAC trigger conversion frequency or lighten the DMA workload to avoid a new DMA underrun. 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.

#### 14.3.8 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 96. DAC LFSR register calculation algorithm

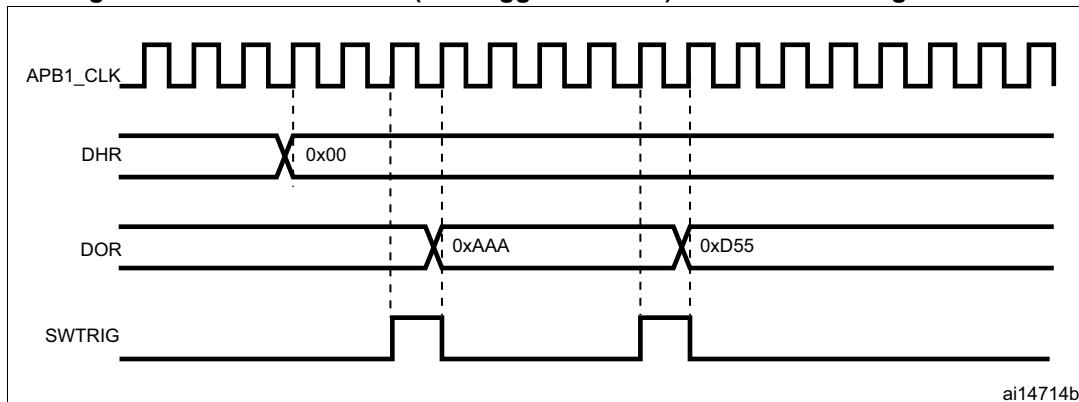


The LFSR value, that may be masked partially or totally by means of the MAMPx[3:0] bits in the DAC\_CR register, is added up to the DAC\_DHRx contents without overflow and this value is then stored into the DAC\_DORx register.

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 97. DAC conversion (SW trigger enabled) with LFSR wave generation



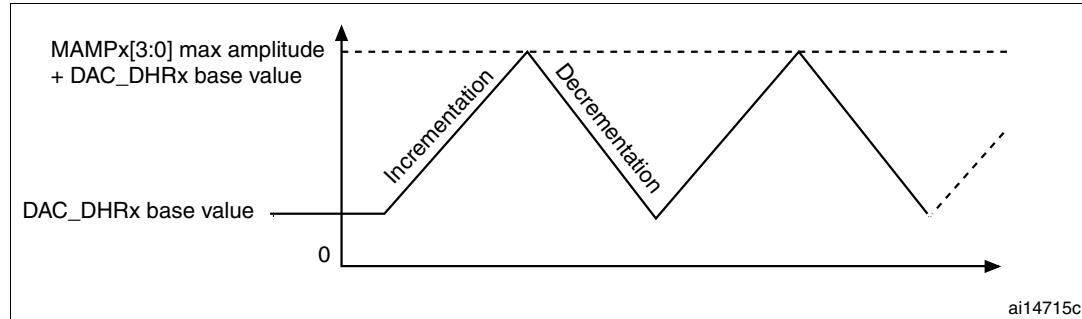
**Note:** The DAC trigger must be enabled for noise generation by setting the TENx bit in the DAC\_CR register.

#### 14.3.9 Triangle-wave generation

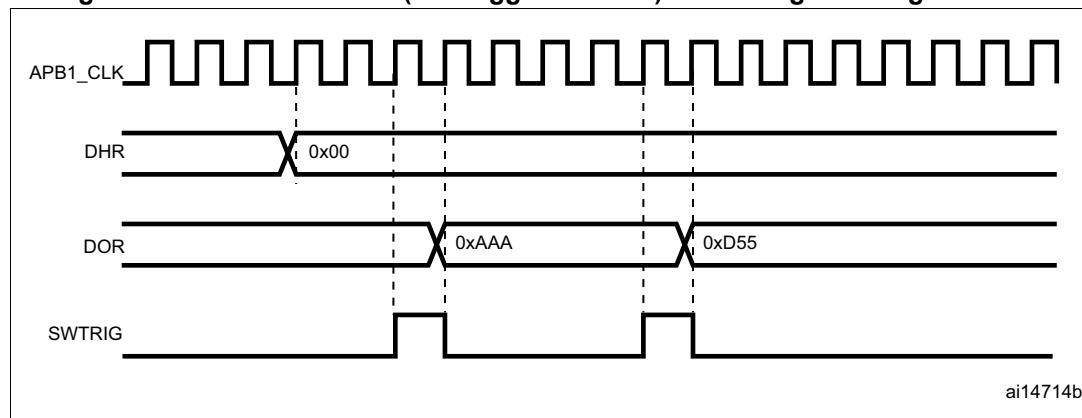
It is possible to add a small-amplitude triangular waveform on a DC or slowly varying signal. DAC triangle-wave generation is selected by setting WAVE[1:0] to “10”. The amplitude is configured through the MAMPx[3:0] bits in the DAC\_CR register. An internal triangle counter is incremented three APB1 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 98. DAC triangle wave generation**



**Figure 99. 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.

## 14.4 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.

#### 14.4.1 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).

#### 14.4.2 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.

#### 14.4.3 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.

#### 14.4.4 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.

#### 14.4.5 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.

#### 14.4.6 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.

#### 14.4.7 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).

#### 14.4.8 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.

#### 14.4.9 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.

#### 14.4.10 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.

#### 14.4.11 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.

## 14.5 DAC registers

Refer to [Section 1 on page 51](#) for a list of abbreviations used in register descriptions.

The peripheral registers have to be accessed by words (32 bits).

### 14.5.1 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
- $\geq 1011$ : 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

### 14.5.2 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.

### 14.5.3 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.

#### 14.5.4 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.

#### 14.5.5 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.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.															
								rw							

Bits 31:8 Reserved, must be kept at reset value.

Bits 7:0 **DACC1DHR[7:0]**: DAC channel1 8-bit right-aligned data

These bits are written by software which specifies 8-bit data for DAC channel1.

#### 14.5.6 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.

#### 14.5.7 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.

#### 14.5.8 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.

#### 14.5.9 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]												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	Res.	Res.	Res.	DACC1DHR[11:0]												
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 27:16 **DACC2DHR[11:0]**: DAC channel2 12-bit right-aligned data

These bits are written by software which specifies 12-bit data for DAC channel2.

Bits 15:12 Reserved, must be kept at reset value.

Bits 11:0 **DACC1DHR[11:0]**: DAC channel1 12-bit right-aligned data

These bits are written by software which specifies 12-bit data for DAC channel1.

#### 14.5.10 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.

#### 14.5.11 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.

#### 14.5.12 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.

#### 14.5.13 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.

#### 14.5.14 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.

#### **14.5.15 DAC register map**

*Table 95* summarizes the DAC registers.

**Table 95. DAC register map**

Refer to [Section 2.2.2 on page 56](#) for the register boundary addresses.

## 15 Digital camera interface (DCMI)

### 15.1 DCMI introduction

The digital camera is a synchronous parallel interface able to receive a high-speed data flow from an external 8-, 10-, 12- or 14-bit CMOS camera module. It supports different data formats: YCbCr4:2:2/RGB565 progressive video and compressed data (JPEG).

This interface is for use with black & white cameras, X24 and X5 cameras, and it is assumed that all preprocessing like resizing is performed in the camera module.

This interface is also able to transmit a parallel data flow, allowing it to emulate a camera module interfacing with another camera interface.

It may also be used as a generic synchronous parallel interface ensuring a high data rate transfer, in receive or in transmit mode. It is a slave interface, the external clock and data flow control being ensured externally (by another device or by other resources of the STM32, e.g. timers).

### 15.2 DCMI main features

- 8-, 10-, 12- or 14-bit parallel interface
- Embedded/external line and frame synchronization
- Continuous or snapshot mode
- Crop feature
- Supports the following data formats:
  - 8/10/12/14- bit progressive video: either monochrome or raw bayer
  - YCbCr 4:2:2 progressive video
  - RGB 565 progressive video
  - Compressed data: JPEG

### 15.3 DCMI clocks

The digital camera interface uses two clock domains, DCMI\_PIXCLK and HCLK. The signals generated with DCMI\_PIXCLK are sampled on the rising edge of HCLK once they are stable. An enable signal is generated in the HCLK domain, to indicate that data coming from the camera are stable and can be sampled. The maximum DCMI\_PIXCLK period must be higher than 2.5 HCLK periods.

## 15.4 DCMI functional overview

The digital camera interface is a synchronous parallel interface that can receive high-speed (up to 54 Mbytes/s) data flows. It consists of up to 14 data lines (D13-D0) and a pixel clock line (DCMI\_PIXCLK). The pixel clock has a programmable polarity, so that data can be captured on either the rising or the falling edge of the pixel clock.

The data are packed into a 32-bit data register (DCMI\_DR) and then transferred through a general-purpose DMA channel. The image buffer is managed by the DMA, not by the camera interface.

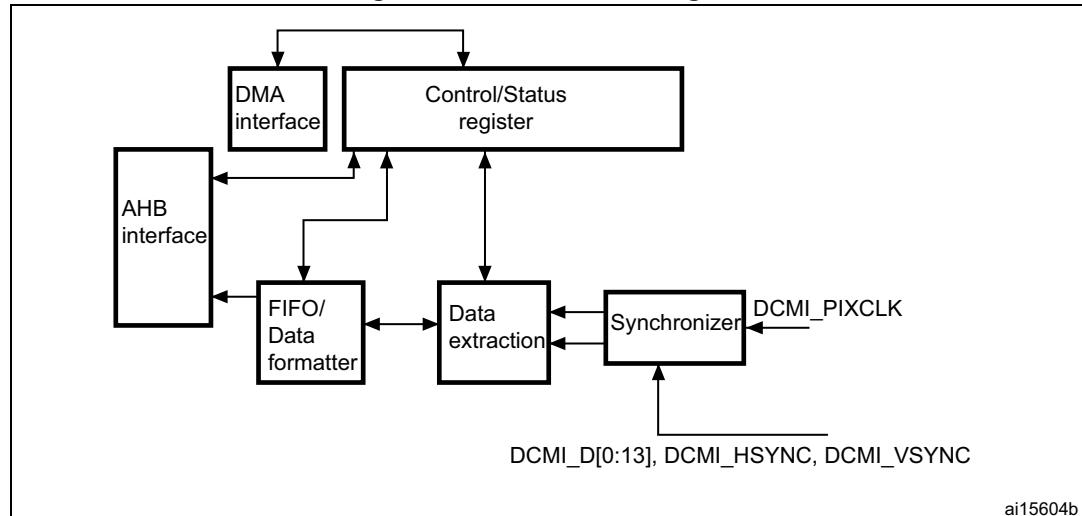
The data received from the camera can be organized in lines/frames (raw YUB/RGB/Bayer modes) or can be a sequence of JPEG images. To enable JPEG image reception, the JPEG bit (bit 3 of DCMI\_CR register) must be set.

The data flow is synchronized either by hardware using the optional DCMI\_HSYNC (horizontal synchronization) and DCMI\_VSYNC (vertical synchronization) signals or by synchronization codes embedded in the data flow.

### 15.4.1 DCMI block diagram

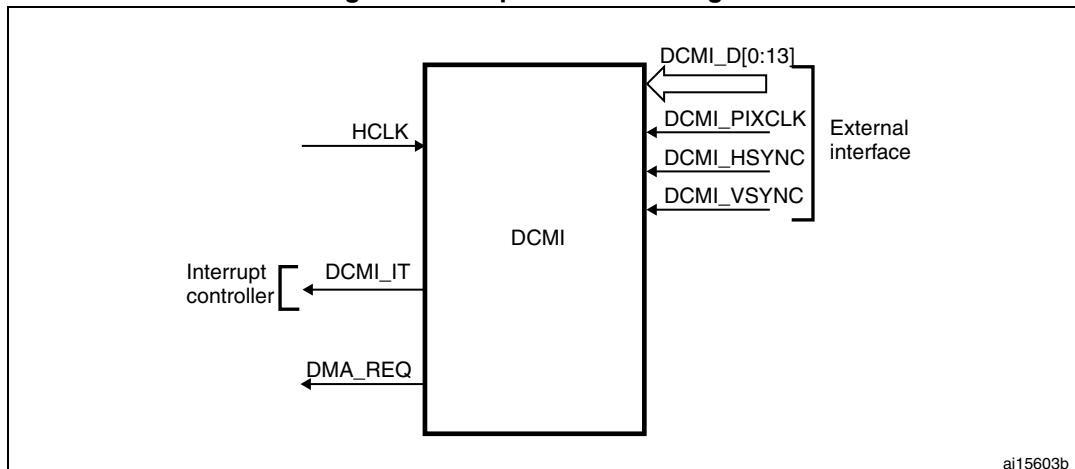
*Figure 100* shows the DCMI block diagram.

**Figure 100. DCMI block diagram**



ai15604b

Figure 101. Top-level block diagram



#### 15.4.2 DMA interface

The DMA interface is active when the CAPTURE bit in the DCMI\_CR register is set. A DMA request is generated each time the camera interface receives a complete 32-bit data block in its register.

#### 15.4.3 DCMI physical interface

The interface is composed of 11/13/15/17 inputs. Only the Slave mode is supported.

The camera interface can capture 8-bit, 10-bit, 12-bit or 14-bit data depending on the EDM[1:0] bits in the DCMI\_CR register. If less than 14 bits are used, the unused input pins must be connected to ground.

*Table 96* shows the DCMI pins.

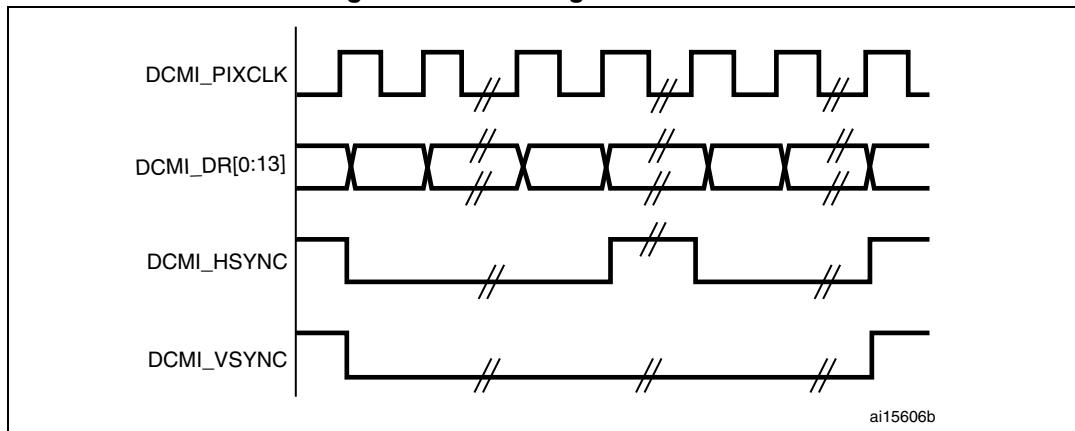
Table 96. DCMI external signals

Signal name	Signal type	Signal description		
8 bits	DCMI_D[0..7]	Digital inputs	DCMI data	
10 bits	DCMI_D[0..9]			
12 bits	DCMI_D[0..11]			
14 bits	DCMI_D[0..13]			
DCMI_PIXCLK	Digital input	Pixel clock		
DCMI_HSYNC	Digital input	Horizontal synchronization / Data valid		
DCMI_VSYNC	Digital input	Vertical synchronization		

The data are synchronous with DCMI\_PIXCLK and change on the rising/falling edge of the pixel clock depending on the polarity.

The DCMI\_HSYNC signal indicates the start/end of a line.

The DCMI\_VSYNC signal indicates the start/end of a frame

**Figure 102. DCMI signal waveforms**

1. The capture edge of DCMI\_PIXCLK is the falling edge, the active state of DCMI\_HSYNC and DCMI\_VSYNC is 1.
2. DCMI\_HSYNC and DCMI\_VSYNC can change states at the same time.

### 8-bit data

When EDM[1:0] in DCMI\_CR are programmed to “00” the interface captures 8 LSBs at its input (DCMI\_D[0:7]) and stores them as 8-bit data. The DCMI\_D[13:8] inputs are ignored. In this case, to capture a 32-bit word, the camera interface takes four pixel clock cycles.

The first captured data byte is placed in the LSB position in the 32-bit word and the 4<sup>th</sup> captured data byte is placed in the MSB position in the 32-bit word. [Table 97](#) gives an example of the positioning of captured data bytes in two 32-bit words.

**Table 97. Positioning of captured data bytes in 32-bit words (8-bit width)**

Byte address	31:24	23:16	15:8	7:0
0	D <sub>n+3</sub> [7:0]	D <sub>n+2</sub> [7:0]	D <sub>n+1</sub> [7:0]	D <sub>n</sub> [7:0]
4	D <sub>n+7</sub> [7:0]	D <sub>n+6</sub> [7:0]	D <sub>n+5</sub> [7:0]	D <sub>n+4</sub> [7:0]

### 10-bit data

When EDM[1:0] in DCMI\_CR are programmed to “01”, the camera interface captures 10-bit data at its input DCMI\_D[0..9] and stores them as the 10 least significant bits of a 16-bit word. The remaining most significant bits in the DCMI\_DR register (bits 11 to 15) are cleared to zero. So, in this case, a 32-bit data word is made up every two pixel clock cycles.

The first captured data are placed in the LSB position in the 32-bit word and the 2<sup>nd</sup> captured data are placed in the MSB position in the 32-bit word as shown in [Table 98](#).

**Table 98. Positioning of captured data bytes in 32-bit words (10-bit width)**

Byte address	31:26	25:16	15:10	9:0
0	0	D <sub>n+1</sub> [9:0]	0	D <sub>n</sub> [9:0]
4	0	D <sub>n+3</sub> [9:0]	0	D <sub>n+2</sub> [9:0]

### 12-bit data

When EDM[1:0] in DCMI\_CR are programmed to “10”, the camera interface captures the 12-bit data at its input DCMI\_D[0..11] and stores them as the 12 least significant bits of a 16-bit word. The remaining most significant bits are cleared to zero. So, in this case a 32-bit data word is made up every two pixel clock cycles.

The first captured data are placed in the LSB position in the 32-bit word and the 2<sup>nd</sup> captured data are placed in the MSB position in the 32-bit word as shown in [Table 99](#).

**Table 99. Positioning of captured data bytes in 32-bit words (12-bit width)**

Byte address	31:28	27:16	15:12	11:0
0	0	D <sub>n+1</sub> [11:0]	0	D <sub>n</sub> [11:0]
4	0	D <sub>n+3</sub> [11:0]	0	D <sub>n+2</sub> [11:0]

### 14-bit data

When EDM[1:0] in DCMI\_CR are programmed to “11”, the camera interface captures the 14-bit data at its input DCMI\_D[0..13] and stores them as the 14 least significant bits of a 16-bit word. The remaining most significant bits are cleared to zero. So, in this case a 32-bit data word is made up every two pixel clock cycles.

The first captured data are placed in the LSB position in the 32-bit word and the 2<sup>nd</sup> captured data are placed in the MSB position in the 32-bit word as shown in [Table 100](#).

**Table 100. Positioning of captured data bytes in 32-bit words (14-bit width)**

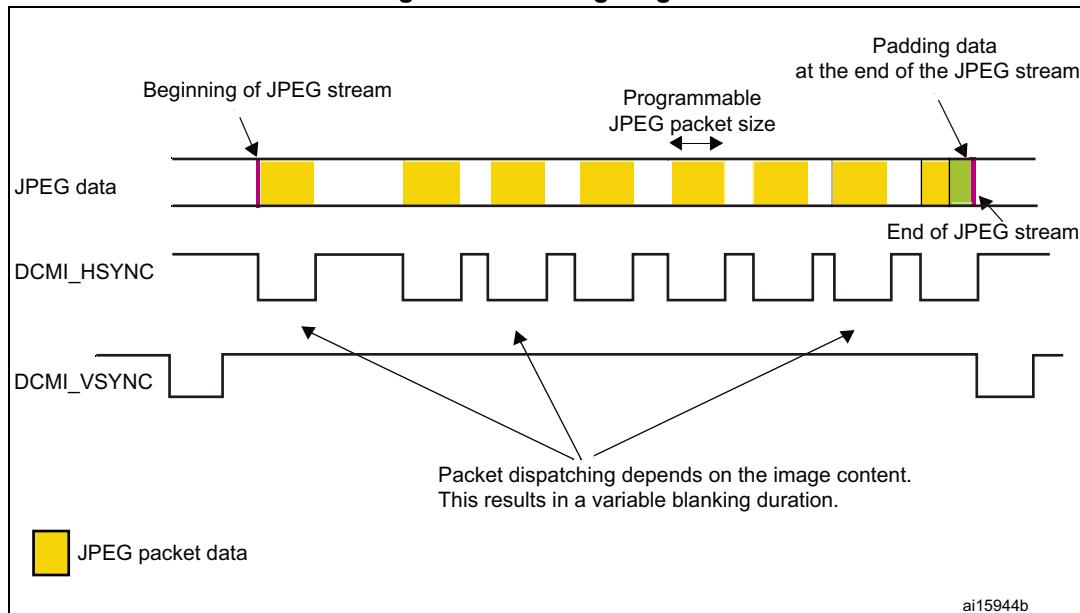
Byte address	31:30	29:16	15:14	13:0
0	0	D <sub>n+1</sub> [13:0]	0	D <sub>n</sub> [13:0]
4	0	D <sub>n+3</sub> [13:0]	0	D <sub>n+2</sub> [13:0]

#### 15.4.4 Synchronization

The digital camera interface supports embedded or hardware (DCMI\_HSYNC and DCMI\_VSYNC) synchronization. When embedded synchronization is used, it is up to the digital camera module to make sure that the 0x00 and 0xFF values are used ONLY for synchronization (not in data). Embedded synchronization codes are supported only for the 8-bit parallel data interface width (that is, in the DCMI\_CR register, the EDM[1:0] bits should be cleared to “00”).

For compressed data, the DCMI supports only the hardware synchronization mode. In this case, DCMI\_VSYNC is used as a start/end of the image, and DCMI\_HSYNC is used as a Data Valid signal. [Figure 103](#) shows the corresponding timing diagram.

Figure 103. Timing diagram



ai15944b

### Hardware synchronization mode

In hardware synchronization mode, the two synchronization signals (DCMI\_HSYNC/DCMI\_VSYNC) are used.

Depending on the camera module/module, data may be transmitted during horizontal/vertical synchronization periods. The DCMI\_HSYNC/DCMI\_VSYNC signals act like blanking signals since all the data received during DCMI\_HSYNC/DCMI\_VSYNC active periods are ignored.

In order to correctly transfer images into the DMA/RAM buffer, data transfer is synchronized with the DCMI\_VSYNC signal. When the hardware synchronization mode is selected, and capture is enabled (CAPTURE bit set in DCMI\_CR), data transfer is synchronized with the deactivation of the DCMI\_VSYNC signal (next start of frame).

Transfer can then be continuous, with successive frames transferred by DMA to successive buffers or the same/circular buffer. To allow the DMA management of successive frames, a VSIF (Vertical synchronization interrupt flag) is activated at the end of each frame.

### Embedded data synchronization mode

In this synchronization mode, the data flow is synchronized using 32-bit codes embedded in the data flow. These codes use the 0x00/0xFF values that are *not* used in data anymore. There are 4 types of codes, all with a 0xFF0000XY format. The embedded synchronization codes are supported only in 8-bit parallel data width capture (in the DCMI\_CR register, the EDM[1:0] bits should be programmed to "00"). For other data widths, this mode generates unpredictable results and must not be used.

**Note:** Camera modules can have 8 such codes (in interleaved mode). For this reason, the interleaved mode is not supported by the camera interface (otherwise, every other half-frame would be discarded).

- Mode 2

Four embedded codes signal the following events

- Frame start (FS)
- Frame end (FE)
- Line start (LS)
- Line end (LE)

The XY values in the 0xFF0000XY format of the four codes are programmable (see [Section 15.7.7: DCMI embedded synchronization code register \(DCMI\\_ESCR\)](#)).

A 0xFF value programmed as a “frame end” means that all the unused codes are interpreted as valid frame end codes.

In this mode, once the camera interface has been enabled, the frame capture starts after the first occurrence of the frame end (FE) code followed by a frame start (FS) code.

- Mode 1

An alternative coding is the camera mode 1. This mode is ITU656 compatible.

The codes signal another set of events:

- SAV (active line) - line start
- EAV (active line) - line end
- SAV (blanking) - end of line during interframe blanking period
- EAV (blanking) - end of line during interframe blanking period

This mode can be supported by programming the following codes:

- FS ≤ 0xFF
- FE ≤ 0xFF
- LS ≤ SAV (active)
- LE ≤ EAV (active)

An embedded unmask code is also implemented for frame/line start and frame/line end codes. Using it, it is possible to compare only the selected unmasked bits with the programmed code. You can therefore select a bit to compare in the embedded code and detect a frame/line start or frame/line end. This means that there can be different codes for the frame/line start and frame/line end with the unmasked bit position remaining the same.

### Example

FS = 0xA5

Unmask code for FS = 0x10

In this case the frame start code is embedded in the bit 4 of the frame start code.

### 15.4.5 Capture modes

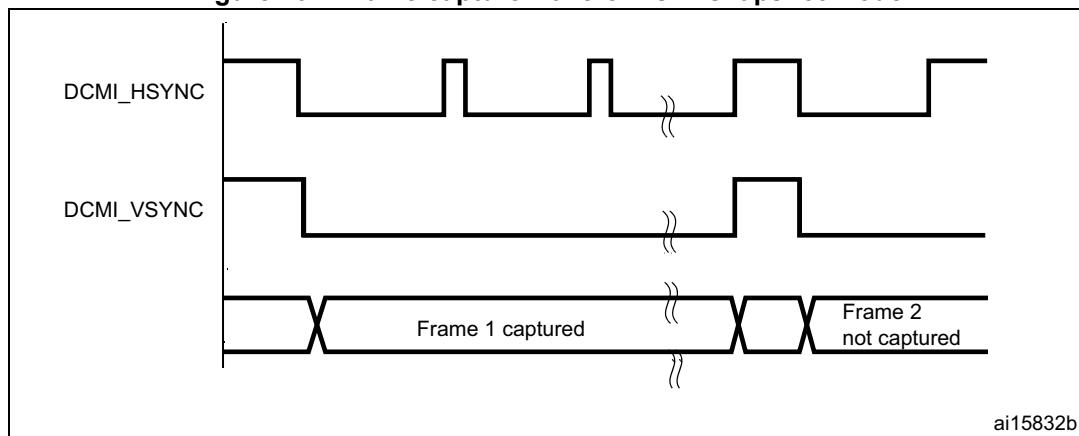
This interface supports two types of capture: snapshot (single frame) and continuous grab.

#### Snapshot mode (single frame)

In this mode, a single frame is captured (CM = '1' in the DCMI\_CR register). After the CAPTURE bit is set in DCMI\_CR, the interface waits for the detection of a start of frame before sampling the data. The camera interface is automatically disabled (CAPTURE bit cleared in DCMI\_CR) after receiving the first complete frame. An interrupt is generated (IT\_FRAME) if it is enabled.

In case of an overrun, the frame is lost and the CAPTURE bit is cleared.

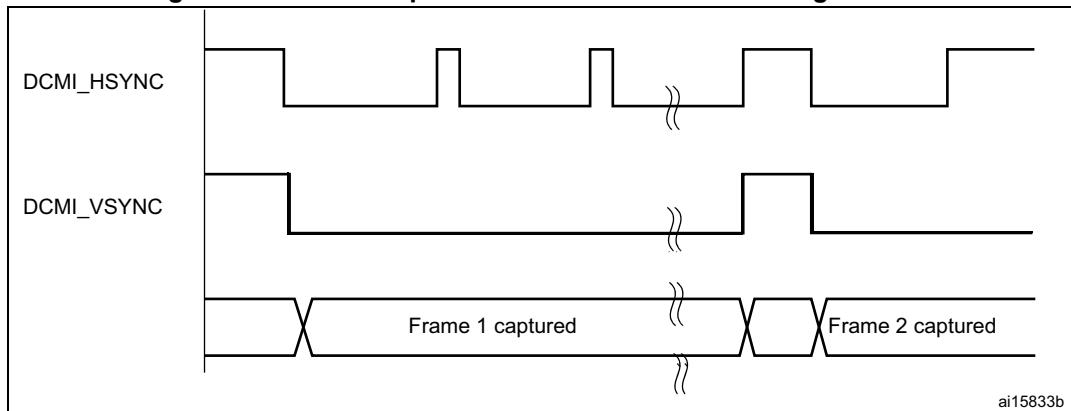
**Figure 104. Frame capture waveforms in snapshot mode**



1. Here, the active state of DCMI\_HSYNC and DCMI\_VSYNC is 1.
2. DCMI\_HSYNC and DCMI\_VSYNC can change states at the same time.

#### Continuous grab mode

In this mode (CM bit = '0' in DCMI\_CR), once the CAPTURE bit has been set in DCMI\_CR, the grabbing process starts on the next DCMI\_VSYNC or embedded frame start depending on the mode. The process continues until the CAPTURE bit is cleared in DCMI\_CR. Once the CAPTURE bit has been cleared, the grabbing process continues until the end of the current frame.

**Figure 105. Frame capture waveforms in continuous grab mode**

1. Here, the active state of DCMI\_HSYNC and DCMI\_VSYNC is 1.
2. DCMI\_HSYNC and DCMI\_VSYNC can change states at the same time.

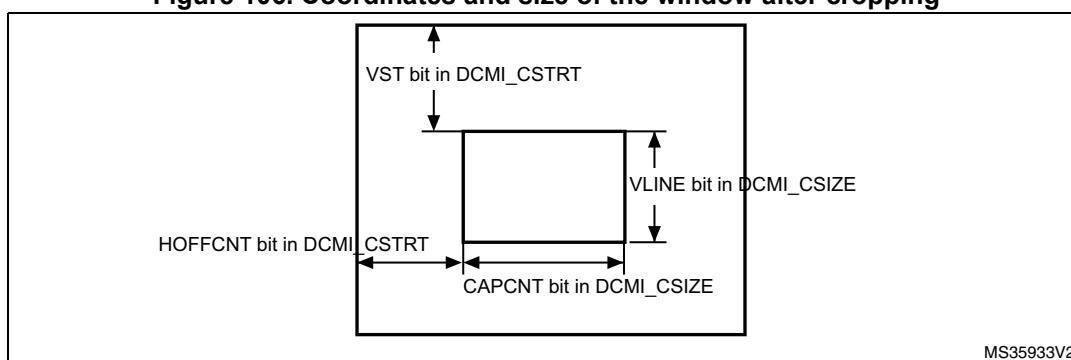
In continuous grab mode, you can configure the FCRC bits in DCMI\_CR to grab all pictures, every second picture or one out of four pictures to decrease the frame capture rate.

**Note:**

*In the hardware synchronization mode (ESS = '0' in DCMI\_CR), the IT\_VSYNC interrupt is generated (if enabled) even when CAPTURE = '0' in DCMI\_CR so, to reduce the frame capture rate even further, the IT\_VSYNC interrupt can be used to count the number of frames between 2 captures in conjunction with the Snapshot mode. This is not allowed by embedded data synchronization mode.*

#### 15.4.6 Crop feature

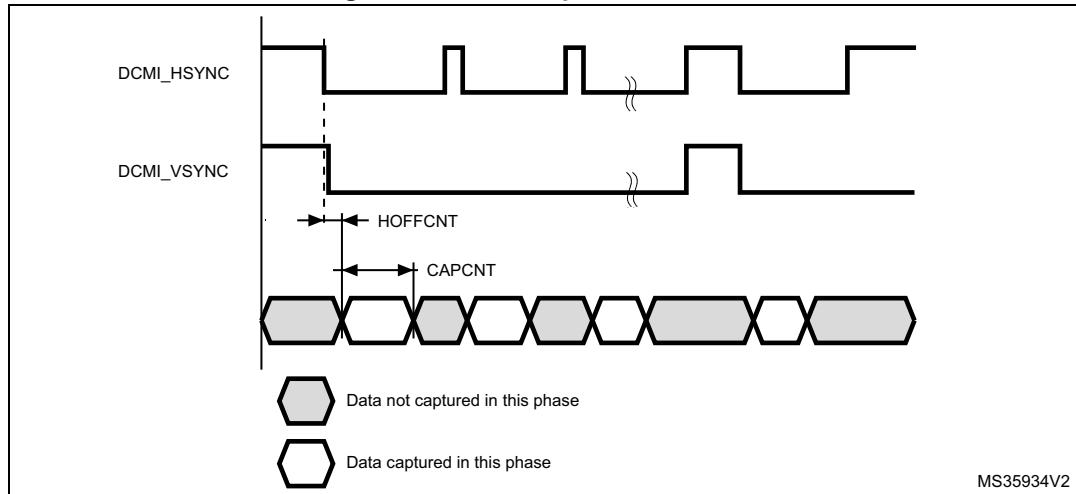
With the crop feature, the camera interface can select a rectangular window from the received image. The start (upper left corner) coordinates and size (horizontal dimension in number of pixel clocks and vertical dimension in number of lines) are specified using two 32-bit registers (DCMI\_CWSTRT and DCMI\_CWSIZE). The size of the window is specified in number of pixel clocks (horizontal dimension) and in number of lines (vertical dimension).

**Figure 106. Coordinates and size of the window after cropping**

These registers specify the coordinates of the starting point of the capture window as a line number (in the frame, starting from 0) and a number of pixel clocks (on the line, starting from 0), and the size of the window as a line number and a number of pixel clocks. The CAPCNT value can only be a multiple of 4 (two least significant bits are forced to 0) to allow the correct transfer of data through the DMA.

If the VSYNC signal goes active before the number of lines is specified in the DCMI\_CWSIZE register, then the capture stops and an IT\_FRAME interrupt is generated when enabled.

**Figure 107. Data capture waveforms**



1. Here, the active state of DCMI\_HSYNC and DCMI\_VSYNC is 1.
2. DCMI\_HSYNC and DCMI\_VSYNC can change states at the same time.

#### 15.4.7 JPEG format

To allow JPEG image reception, it is necessary to set the JPEG bit in the DCMI\_CR register. JPEG images are not stored as lines and frames, so the DCMI\_VSYNC signal is used to start the capture while DCMI\_HSYNC serves as a data enable signal. The number of bytes in a line may not be a multiple of 4, you should therefore be careful when handling this case since a DMA request is generated each time a complete 32-bit word has been constructed from the captured data. When an end of frame is detected and the 32-bit word to be transferred has not been completely received, the remaining data are padded with '0s' and a DMA request is generated.

The crop feature and embedded synchronization codes cannot be used in the JPEG format.

#### 15.4.8 FIFO

##### Input mode

A four-word FIFO is implemented to manage data rate transfers on the AHB. The DCMI features a simple FIFO controller with a read pointer incremented each time the camera interface reads from the AHB, and a write pointer incremented each time the camera interface writes to the FIFO. There is no overrun protection to prevent the data from being overwritten if the AHB interface does not sustain the data transfer rate.

In case of overrun or errors in the synchronization signals, the FIFO is reset and the DCMI interface waits for a new start of frame.

## 15.5 Data format description

### 15.5.1 Data formats

Three types of data are supported:

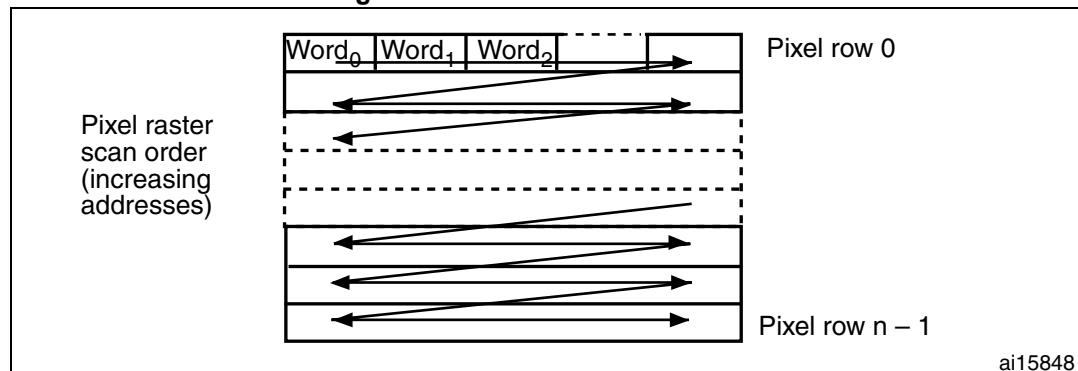
- 8-bit progressive video: either monochrome or raw Bayer format
- YCbCr 4:2:2 progressive video
- RGB565 progressive video. A pixel coded in 16 bits (5 bits for blue, 5 bits for red, 6 bits for green) takes two clock cycles to be transferred.

Compressed data: JPEG

For B&W, YCbCr or RGB data, the maximum input size is  $2048 \times 2048$  pixels. No limit in JPEG compressed mode.

For monochrome, RGB & YCbCr, the frame buffer is stored in raster mode. 32-bit words are used. Only the little endian format is supported.

**Figure 108. Pixel raster scan order**



ai15848

### 15.5.2 Monochrome format

Characteristics:

- Raster format
- 8 bits per pixel

*Table 101* shows how the data are stored.

**Table 101. Data storage in monochrome progressive video format**

Byte address	31:24	23:16	15:8	7:0
0	n + 3	n + 2	n + 1	n
4	n + 7	n + 6	n + 5	n + 4

### 15.5.3 RGB format

Characteristics:

- Raster format
- RGB
- Interleaved: one buffer: R, G & B interleaved: BRGBRGBRG, etc.
- Optimized for display output

The RGB planar format is compatible with standard OS frame buffer display formats.

Only 16 BPP (bits per pixel): RGB565 (2 pixels per 32-bit word) is supported.

The 24 BPP (palletized format) and grayscale formats are not supported. Pixels are stored in a raster scan order, that is from top to bottom for pixel rows, and from left to right within a pixel row. Pixel components are R (red), G (green) and B (blue). All components have the same spatial resolution (4:4:4 format). A frame is stored in a single part, with the components interleaved on a pixel basis.

*Table 102* shows how the data are stored.

**Table 102. Data storage in RGB progressive video format**

Byte address	31:27	26:21	20:16	15:11	10:5	4:0
0	Red n + 1	Green n + 1	Blue n + 1	Red n	Green n	Blue n
4	Red n + 4	Green n + 3	Blue n + 3	Red n + 2	Green n + 2	Blue n + 2

### 15.5.4 YCbCr format

Characteristics:

- Raster format
- YCbCr 4:2:2
- Interleaved: one Buffer: Y, Cb & Cr interleaved: CbYCrYCbYCr, etc.

Pixel components are Y (luminance or “luma”), Cb and Cr (chrominance or “chroma” blue and red). Each component is encoded in 8 bits. Luma and chroma are stored together (interleaved) as shown in *Table 103*.

**Table 103. Data storage in YCbCr progressive video format**

Byte address	31:24	23:16	15:8	7:0
0	Y n + 1	Cr n	Y n	Cb n
4	Y n + 3	Cr n + 2	Y n + 2	Cb n + 2

### 15.5.5 YCbCr format - Y only

Characteristics:

- Raster format
- YCbCr 4:2:2
- The buffer only contains Y information - monochrome image

Pixel components are Y (luminance or “luma”), Cb and Cr (chrominance or “chroma” blue and red). In this mode, the chroma information is dropped. Only Luma component of each

pixel , encoded in 8 bits, is stored as shown in [Table 104](#).

The result is a monochrome image having the same resolution as the original YCbCr data.

**Table 104. Data storage in YCbCr progressive video format - Y extraction mode**

Byte address	31:24	23:16	15:8	7:0
0	Y n + 3	Y n + 2	Y n + 1	Y n
4	Y n + 7	Y n + 6	Y n + 5	Y n + 4

### 15.5.6 Half resolution image extraction

This is a modification of the previous reception modes, being applicable to monochrome, RGB or Y extraction modes.

This mode allows to only store a half resolution image. It is selected through OELS and LSM control bits.

## 15.6 DCMI interrupts

Five interrupts are generated. All interrupts are maskable by software. The global interrupt (IT\_DCMI) is the OR of all the individual interrupts. [Table 105](#) gives the list of all interrupts.

**Table 105. DCMI interrupts**

Interrupt name	Interrupt event
IT_LINE	Indicates the end of line
IT_FRAME	Indicates the end of frame capture
IT_OVR	indicates the overrun of data reception
IT_VSYNC	Indicates the synchronization frame
IT_ERR	Indicates the detection of an error in the embedded synchronization frame detection
IT_DCMI	Logic OR of the previous interrupts

## 15.7 DCMI register description

All DCMI registers have to be accessed as 32-bit words, otherwise a bus error occurs.

### 15.7.1 DCMI control register (DCMI\_CR)

Address offset: 0x00

Reset value: 0x0000 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OEELS	LSM	OEBS	BSM	
											rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	ENABLE	Res.	Res.	EDM		FCRC		VSPOL	HSPOL	PCKPOL	ESS	JPEG	CROP	CM	CAPTURE
	rw			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:21 Reserved, must be kept at reset value.

Bit 20 **OEELS:** Odd/Even Line Select (Line Select Start)

This bit works in conjunction with LSM field (LSM = 1)

0: Interface captures first line after the frame start, second one being dropped

1: Interface captures second line from the frame start, first one being dropped

Bit 19 **LSM:** Line Select mode

0: Interface captures all received lines

1: Interface captures one line out of two.

Bit 18 **OEBS:** Odd/Even Byte Select (Byte Select Start)

This bit works in conjunction with BSM field (BSM <> 00)

0: Interface captures first data (byte or double byte) from the frame/line start, second one being dropped

1: Interface captures second data (byte or double byte) from the frame/line start, first one being dropped

Bits 17:16 **BSM[1:0]:** Byte Select mode

00: Interface captures all received data

01: Interface captures every other byte from the received data

10: Interface captures one byte out of four

11: Interface captures two bytes out of four

*Note: This mode only work for EDM[1:0]=00. For all other EDM values, this bit field must be programmed to the reset value.*

Bit 15 Reserved, must be kept at reset value.

Bit 14 **ENABLE:** DCMI enable

0: DCMI disabled

1: DCMI enabled

*Note: The DCMI configuration registers should be programmed correctly before enabling this Bit*

Bits 13:12 Reserved, must be kept at reset value.

Bits 11:10 **EDM[1:0]:** Extended data mode

00: Interface captures 8-bit data on every pixel clock

01: Interface captures 10-bit data on every pixel clock

10: Interface captures 12-bit data on every pixel clock

11: Interface captures 14-bit data on every pixel clock

Bits 9:8 **FCRC[1:0]:** Frame capture rate control

These bits define the frequency of frame capture. They are meaningful only in Continuous grab mode. They are ignored in snapshot mode.

00: All frames are captured

01: Every alternate frame captured (50% bandwidth reduction)

10: One frame in 4 frames captured (75% bandwidth reduction)

11: reserved

**Bit 7 VSPOL:** Vertical synchronization polarity

This bit indicates the level on the DCMI\_VSYNC pin when the data are not valid on the parallel interface.

- 0: DCMI\_VSYNC active low
- 1: DCMI\_VSYNC active high

**Bit 6 HSPOL:** Horizontal synchronization polarity

This bit indicates the level on the DCMI\_HSYNC pin when the data are not valid on the parallel interface.

- 0: DCMI\_HSYNC active low
- 1: DCMI\_HSYNC active high

**Bit 5 PCKPOL:** Pixel clock polarity

This bit configures the capture edge of the pixel clock

- 0: Falling edge active.
- 1: Rising edge active.

**Bit 4 ESS:** Embedded synchronization select

0: Hardware synchronization data capture (frame/line start/stop) is synchronized with the DCMI\_HSYNC/DCMI\_VSYNC signals.

1: Embedded synchronization data capture is synchronized with synchronization codes embedded in the data flow.

*Note: Valid only for 8-bit parallel data. HSPOL/VSPOL are ignored when the ESS bit is set.*

This bit is disabled in JPEG mode.

**Bit 3 JPEG:** JPEG format

0: Uncompressed video format

1: This bit is used for JPEG data transfers. The DCMI\_HSYNC signal is used as data enable. The crop and embedded synchronization features (ESS bit) cannot be used in this mode.

**Bit 2 CROP:** Crop feature

0: The full image is captured. In this case the total number of bytes in an image frame should be a multiple of 4

1: Only the data inside the window specified by the crop register will be captured. If the size of the crop window exceeds the picture size, then only the picture size is captured.

**Bit 1 CM:** Capture mode

0: Continuous grab mode - The received data are transferred into the destination memory through the DMA. The buffer location and mode (linear or circular buffer) is controlled through the system DMA.

1: Snapshot mode (single frame) - Once activated, the interface waits for the start of frame and then transfers a single frame through the DMA. At the end of the frame, the CAPTURE bit is automatically reset.

**Bit 0 CAPTURE:** Capture enable

0: Capture disabled.

1: Capture enabled.

The camera interface waits for the first start of frame, then a DMA request is generated to transfer the received data into the destination memory.

In snapshot mode, the CAPTURE bit is automatically cleared at the end of the 1st frame received.

In continuous grab mode, if the software clears this bit while a capture is ongoing, the bit will be effectively cleared after the frame end.

*Note: The DMA controller and all DCMI configuration registers should be programmed correctly before enabling this bit.*

### 15.7.2 DCMI status register (DCMI\_SR)

Address offset: 0x04

Reset value: 0x0000 0x0000

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

Bits 31:3 Reserved, must be kept at reset value.

#### Bit 2 FNE: FIFO not empty

This bit gives the status of the FIFO

1: FIFO contains valid data

0: FIFO empty

#### Bit 1 VSYNC

This bit gives the state of the DCMI\_VSYNC pin with the correct programmed polarity.

When embedded synchronization codes are used, the meaning of this bit is the following:

0: active frame

1: synchronization between frames

In case of embedded synchronization, this bit is meaningful only if the CAPTURE bit in DCMI\_CR is set.

#### Bit 0 HSYNC

This bit gives the state of the DCMI\_HSYNC pin with the correct programmed polarity.

When embedded synchronization codes are used, the meaning of this bit is the following:

0: active line

1: synchronization between lines

In case of embedded synchronization, this bit is meaningful only if the CAPTURE bit in DCMI\_CR is set.

### 15.7.3 DCMI raw interrupt status register (DCMI\_RIS)

Address offset: 0x08

Reset value: 0x0000 0x0000

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

DCMI\_RIS gives the raw interrupt status and is accessible in read only. When read, this register returns the status of the corresponding interrupt before masking with the DCMI\_IER register value.

Bits 31:5 Reserved, must be kept at reset value.

#### Bit 4 LINE\_RIS: Line raw interrupt status

This bit gets set when the DCMI\_HSYNC signal changes from the inactive state to the active state. It goes high even if the line is not valid.

In the case of embedded synchronization, this bit is set only if the CAPTURE bit in DCMI\_CR is set.

It is cleared by writing a '1' to the LINE\_ISC bit in DCMI\_ICR.

#### Bit 3 VSYNC\_RIS: DCMI\_VSYNC raw interrupt status

This bit is set when the DCMI\_VSYNC signal changes from the inactive state to the active state.

In the case of embedded synchronization, this bit is set only if the CAPTURE bit is set in DCMI\_CR.

It is cleared by writing a '1' to the VSYNC\_ISC bit in DCMI\_ICR.

#### Bit 2 ERR\_RIS: Synchronization error raw interrupt status

0: No synchronization error detected

1: Embedded synchronization characters are not received in the correct order.

This bit is valid only in the embedded synchronization mode. It is cleared by writing a '1' to the ERR\_ISC bit in DCMI\_ICR.

*Note: This bit is available only in embedded synchronization mode.*

#### Bit 1 OVR\_RIS: Overrun raw interrupt status

0: No data buffer overrun occurred

1: A data buffer overrun occurred and the data FIFO is corrupted.

This bit is cleared by writing a '1' to the OVR\_ISC bit in DCMI\_ICR.

#### Bit 0 FRAME\_RIS: Capture complete raw interrupt status

0: No new capture

1: A frame has been captured.

This bit is set when a frame or window has been captured.

In case of a cropped window, this bit is set at the end of line of the last line in the crop. It is set even if the captured frame is empty (e.g. window cropped outside the frame).

This bit is cleared by writing a '1' to the FRAME\_ISC bit in DCMI\_ICR.

### 15.7.4 DCMI interrupt enable register (DCMI\_IER)

Address offset: 0x0C

Reset value: 0x0000 0x0000

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

The DCMI\_IER register is used to enable interrupts. When one of the DCMI\_IER bits is set, the corresponding interrupt is enabled. This register is accessible in both read and write.

Bits 31:5 Reserved, must be kept at reset value.

Bit 4 **LINE\_IE**: Line interrupt enable

0: No interrupt generation when the line is received

1: An Interrupt is generated when a line has been completely received

Bit 3 **VSYNC\_IE**: DCMI\_VSYNC interrupt enable

0: No interrupt generation

1: An interrupt is generated on each DCMI\_VSYNC transition from the inactive to the active state

The active state of the DCMI\_VSYNC signal is defined by the VSPOL bit.

Bit 2 **ERR\_IE**: Synchronization error interrupt enable

0: No interrupt generation

1: An interrupt is generated if the embedded synchronization codes are not received in the correct order.

*Note:* This bit is available only in embedded synchronization mode.

Bit 1 **OVR\_IE**: Overrun interrupt enable

0: No interrupt generation

1: An interrupt is generated if the DMA was not able to transfer the last data before new data (32-bit) are received.

Bit 0 **FRAME\_IE**: Capture complete interrupt enable

0: No interrupt generation

1: An interrupt is generated at the end of each received frame/crop window (in crop mode).

### 15.7.5 DCMI masked interrupt status register (DCMI\_MIS)

This DCMI\_MIS register is a read-only register. When read, it returns the current masked status value (depending on the value in DCMI\_IER) of the corresponding interrupt. A bit in this register is set if the corresponding enable bit in DCMI\_IER is set and the corresponding bit in DCMI\_RIS is set.

Address offset: 0x10

Reset value: 0x0000

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

Bits 31:5 Reserved, must be kept at reset value.

#### Bit 4 **LINE\_MIS**: Line masked interrupt status

This bit gives the status of the masked line interrupt

0: No interrupt generation when the line is received

1: An interrupt is generated when a line has been completely received and the LINE\_IE bit is set in DCMI\_IER.

#### Bit 3 **VSYNC\_MIS**: VSYNC masked interrupt status

This bit gives the status of the masked VSYNC interrupt

0: No interrupt is generated on DCMI\_VSYNC transitions

1: An interrupt is generated on each DCMI\_VSYNC transition from the inactive to the active state and the VSYNC\_IE bit is set in DCMI\_IER.

The active state of the DCMI\_VSYNC signal is defined by the VSPOL bit.

#### Bit 2 **ERR\_MIS**: Synchronization error masked interrupt status

This bit gives the status of the masked synchronization error interrupt

0: No interrupt is generated on a synchronization error

1: An interrupt is generated if the embedded synchronization codes are not received in the correct order and the ERR\_IE bit in DCMI\_IER is set.

*Note: This bit is available only in embedded synchronization mode.*

#### Bit 1 **OVR\_MIS**: Overrun masked interrupt status

This bit gives the status of the masked overflow interrupt

0: No interrupt is generated on overrun

1: An interrupt is generated if the DMA was not able to transfer the last data before new data (32-bit) are received and the OVR\_IE bit is set in DCMI\_IER.

#### Bit 0 **FRAME\_MIS**: Capture complete masked interrupt status

This bit gives the status of the masked capture complete interrupt

0: No interrupt is generated after a complete capture

1: An interrupt is generated at the end of each received frame/crop window (in crop mode) and the FRAME\_IE bit is set in DCMI\_IER.

### 15.7.6 DCMI interrupt clear register (DCMI\_ICR)

Address offset: 0x14

Reset value: 0x0000 0x0000

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

The DCMI\_ICR register is write-only. Writing a ‘1’ into a bit of this register clears the corresponding bit in the DCMI\_RIS and DCMI\_MIS registers. Writing a ‘0’ has no effect.

Bits 15:5 Reserved, must be kept at reset value.

Bit 4 **LINE\_ISC:** line interrupt status clear

Writing a ‘1’ into this bit clears LINE\_RIS in the DCMI\_RIS register

Bit 3 **VSYNC\_ISC:** Vertical Synchronization interrupt status clear

Writing a ‘1’ into this bit clears the VSYNC\_RIS bit in DCMI\_RIS

Bit 2 **ERR\_ISC:** Synchronization error interrupt status clear

Writing a ‘1’ into this bit clears the ERR\_RIS bit in DCMI\_RIS

*Note: This bit is available only in embedded synchronization mode.*

Bit 1 **OVR\_ISC:** Overrun interrupt status clear

Writing a ‘1’ into this bit clears the OVR\_RIS bit in DCMI\_RIS

Bit 0 **FRAME\_ISC:** Capture complete interrupt status clear

Writing a ‘1’ into this bit clears the FRAME\_RIS bit in DCMI\_RIS

### 15.7.7 DCMI embedded synchronization code register (DCMI\_ESCR)

Address offset: 0x18

Reset value: 0x0000 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FEC								LEC							
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
LSC								FSC							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **FEC**: Frame end delimiter code

This byte specifies the code of the frame end delimiter. The code consists of 4 bytes in the form of 0xFF, 0x00, 0x00, FEC.

If FEC is programmed to 0xFF, all the unused codes (0xFF0000XY) are interpreted as frame end delimiters.

Bits 23:16 **LEC**: Line end delimiter code

This byte specifies the code of the line end delimiter. The code consists of 4 bytes in the form of 0xFF, 0x00, 0x00, LEC.

Bits 15:8 **LSC**: Line start delimiter code

This byte specifies the code of the line start delimiter. The code consists of 4 bytes in the form of 0xFF, 0x00, 0x00, LSC.

Bits 7:0 **FSC**: Frame start delimiter code

This byte specifies the code of the frame start delimiter. The code consists of 4 bytes in the form of 0xFF, 0x00, 0x00, FSC.

If FSC is programmed to 0xFF, no frame start delimiter is detected. But, the 1<sup>st</sup> occurrence of LSC after an FEC code will be interpreted as a start of frame delimiter.

### 15.7.8 DCMI embedded synchronization unmask register (DCMI\_ESUR)

Address offset: 0x1C

Reset value: 0x0000 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FEU								LEU							
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
LSU								FSU							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

#### Bits 31:24 **FEU**: Frame end delimiter unmask

This byte specifies the mask to be applied to the code of the frame end delimiter.

0: The corresponding bit in the FEC byte in DCMI\_ESCR is masked while comparing the frame end delimiter with the received data.

1: The corresponding bit in the FEC byte in DCMI\_ESCR is compared while comparing the frame end delimiter with the received data

#### Bits 23:16 **LEU**: Line end delimiter unmask

This byte specifies the mask to be applied to the code of the line end delimiter.

0: The corresponding bit in the LEC byte in DCMI\_ESCR is masked while comparing the line end delimiter with the received data

1: The corresponding bit in the LEC byte in DCMI\_ESCR is compared while comparing the line end delimiter with the received data

#### Bits 15:8 **LSU**: Line start delimiter unmask

This byte specifies the mask to be applied to the code of the line start delimiter.

0: The corresponding bit in the LSC byte in DCMI\_ESCR is masked while comparing the line start delimiter with the received data

1: The corresponding bit in the LSC byte in DCMI\_ESCR is compared while comparing the line start delimiter with the received data

#### Bits 7:0 **FSU**: Frame start delimiter unmask

This byte specifies the mask to be applied to the code of the frame start delimiter.

0: The corresponding bit in the FSC byte in DCMI\_ESCR is masked while comparing the frame start delimiter with the received data

1: The corresponding bit in the FSC byte in DCMI\_ESCR is compared while comparing the frame start delimiter with the received data

### 15.7.9 DCMI crop window start (DCMI\_CWSTRT)

Address offset: 0x20

Reset value: 0x0000 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16		
Res.	Res.	Res.	VST[12:0]														
			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.	HOFFCNT[13:0]															
		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 28:16 **VST[12:0]**: Vertical start line count

The image capture starts with this line number. Previous line data are ignored.

0x0000 => line 1

0x0001 => line 2

0x0002 => line 3

....

Bits 15:14 Reserved, must be kept at reset value.

Bits 13:0 **HOFFCNT[13:0]**: Horizontal offset count

This value gives the number of pixel clocks to count before starting a capture.

### 15.7.10 DCMI crop window size (DCMI\_CWSIZE)

Address offset: 0x24

Reset value: 0x0000 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Res.	Res.	VLINEx13:0]														
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	Res.	CAPCNT[13:0]														
		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 29:16 **VLINEx13:0]**: Vertical line count

This value gives the number of lines to be captured from the starting point.

0x0000 => 1 line

0x0001 => 2 lines

0x0002 => 3 lines

....

Bits 15:14 Reserved, must be kept at reset value.

Bits 13:0 **CAPCNT[13:0]**: Capture count

This value gives the number of pixel clocks to be captured from the starting point on the same line. Its value should correspond to word-aligned data for different widths of parallel interfaces.

0x0000 => 1 pixel

0x0001 => 2 pixels

0x0002 => 3 pixels

....

### 15.7.11 DCMI data register (DCMI\_DR)

Address offset: 0x28

Reset value: 0x0000 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Byte3								Byte2							
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
Byte1								Byte0							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:24 Data byte 3

Bits 23:16 Data byte 2

Bits 15:8 Data byte 1

Bits 7:0 Data byte 0

The digital camera Interface packages all the received data in 32-bit format before requesting a DMA transfer. A 4-word deep FIFO is available to leave enough time for DMA transfers and avoid DMA overrun conditions.

### 15.7.12 DCMI register map

*Table 106* summarizes the DCMI registers.

**Table 106. DCMI register map and reset values**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5			
0x00	DCMI_CR	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			
	Reset value																														
0x04	DCMI_SR	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			
	Reset value																														
0x08	DCMI_RIS	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			
	Reset value																														
0x0C	DCMI_IER	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			
	Reset value																														
0x10	DCMI_MIS	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			
	Reset value																														
0x14	DCMI_ICR	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			
	Reset value																														
0x18	DCMI_ESCR	FEC				LEC				LSC				FSC				FSU				HOFFCNT[13:0]				VST[12:0]					
	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		
0x1C	DCMI_ESUR	FEU				LEU				LSU				FSU				CAPCNT[13:0]				VLINE13:0]				LINE_MIS					
	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		
0x20	DCMI_CWSTRT	Res.	Res.	Res.	Res.	VST[12:0]												HOFFCNT[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		
0x24	DCMI_CWSIZE	Res.	Res.	VLINE13:0]												CAPCNT[13:0]												ERR_ISC			
	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	DCMI_DR	Byte3				Byte2				Byte1				Byte0				PCKPOL				HSPOL				VSPOL					
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Refer to [Section 2.2.2 on page 56](#) for the register boundary addresses.

## 16 Advanced-control timers (TIM1&TIM8)

### 16.1 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 length 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.

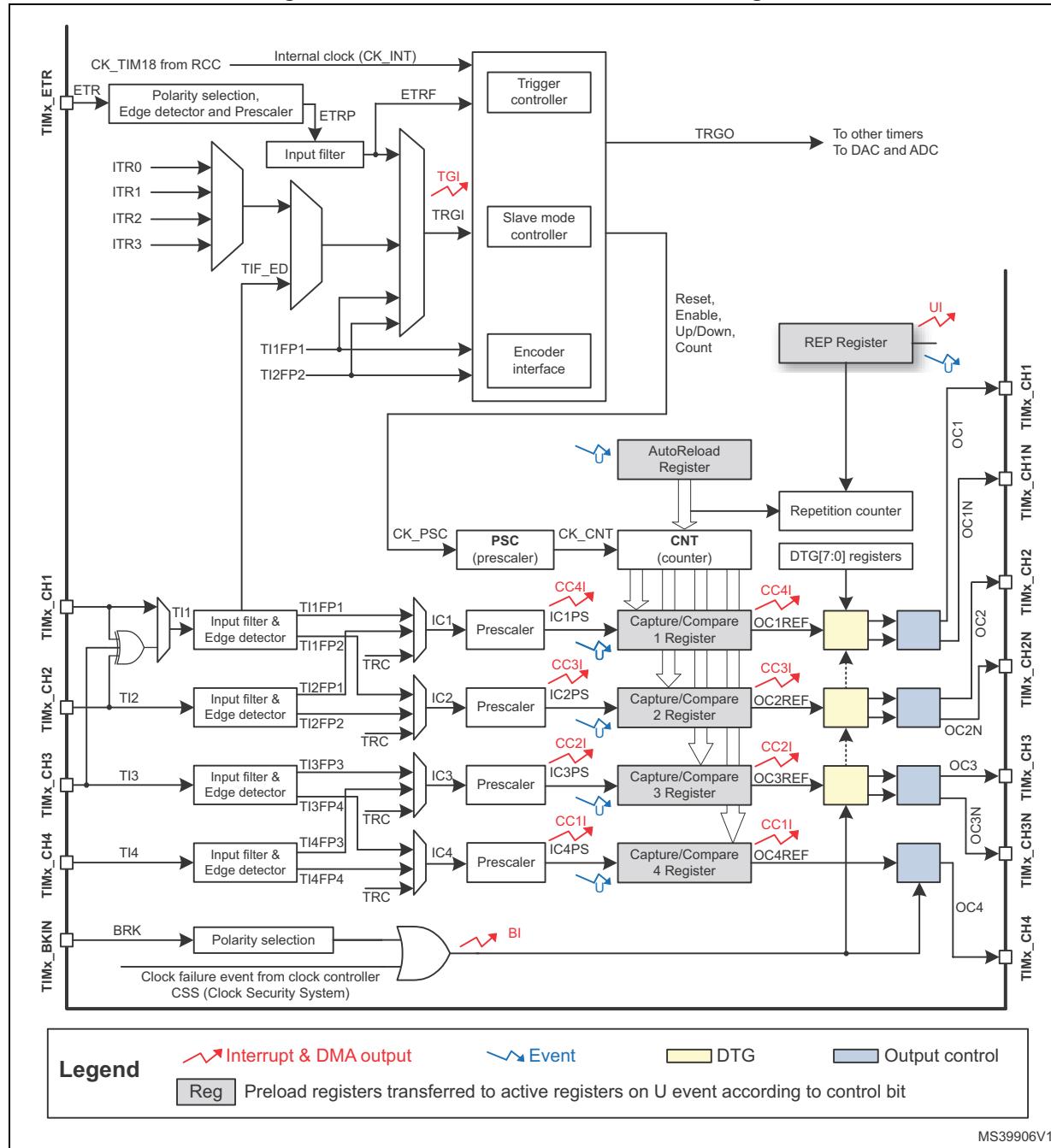
The advanced-control (TIM1&TIM8) and general-purpose (TIMx) timers are completely independent, and do not share any resources. They can be synchronized together as described in [Section 16.3.20](#).

### 16.2 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 4 independent channels for:
  - Input Capture
  - Output Compare
  - PWM generation (Edge and Center-aligned Mode)
  - One-pulse mode output
- Complementary outputs with programmable dead-time
- Synchronization circuit to control the timer with external signals and to interconnect several timers together.
- Repetition counter to update the timer registers only after a given number of cycles of the counter.
- Break input to put the timer’s output signals in reset state or in a known state.
- Interrupt/DMA generation on the following events:
  - Update: counter overflow/underflow, counter initialization (by software or internal/external trigger)
  - Trigger event (counter start, stop, initialization or count by internal/external trigger)
  - Input capture
  - Output compare
  - Break input
- Supports incremental (quadrature) encoder and Hall-sensor circuitry for positioning purposes
- Trigger input for external clock or cycle-by-cycle current management

Figure 109. Advanced-control timer block diagram



## 16.3 TIM1&TIM8 functional description

### 16.3.1 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.

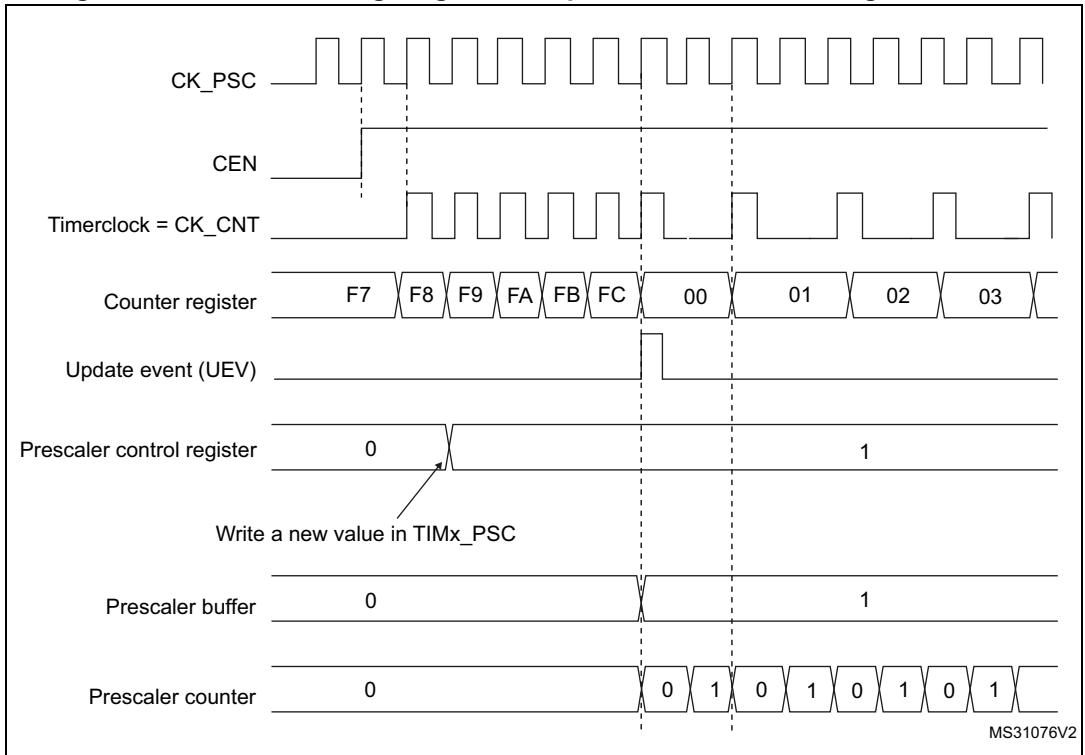
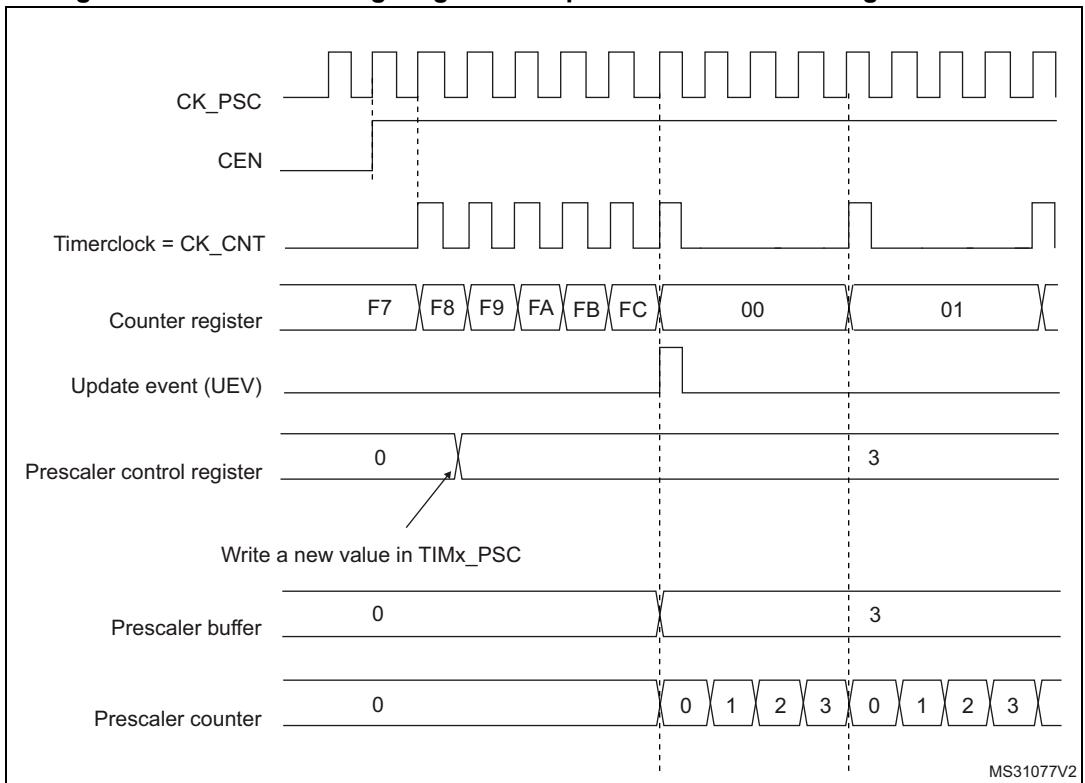
The counter is clocked by the prescaler output CK\_CNT, which is enabled only when the counter enable bit (CEN) in TIMx\_CR1 register is set (refer also to the slave mode controller description to get more details on counter enabling).

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 110* and *Figure 111* give some examples of the counter behavior when the prescaler ratio is changed on the fly:

**Figure 110. Counter timing diagram with prescaler division change from 1 to 2****Figure 111. Counter timing diagram with prescaler division change from 1 to 4**

### 16.3.2 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 plus one (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.

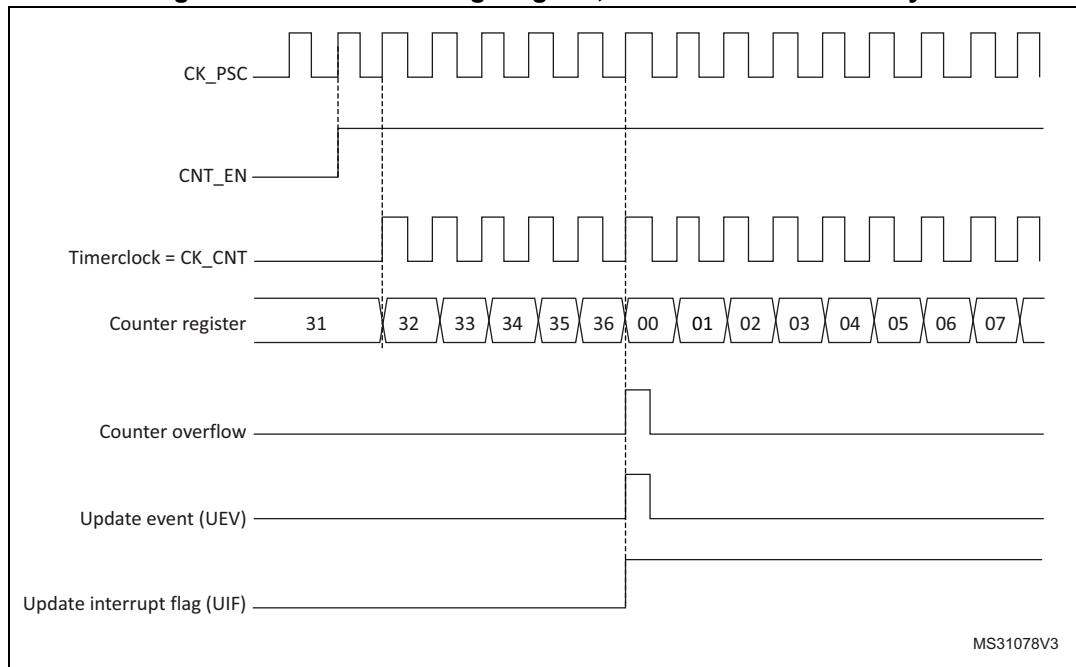
The UEV event can be disabled by software by setting the UDIS bit in the TIMx\_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until the UDIS bit has been written to 0. However, the counter restarts from 0, as 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.

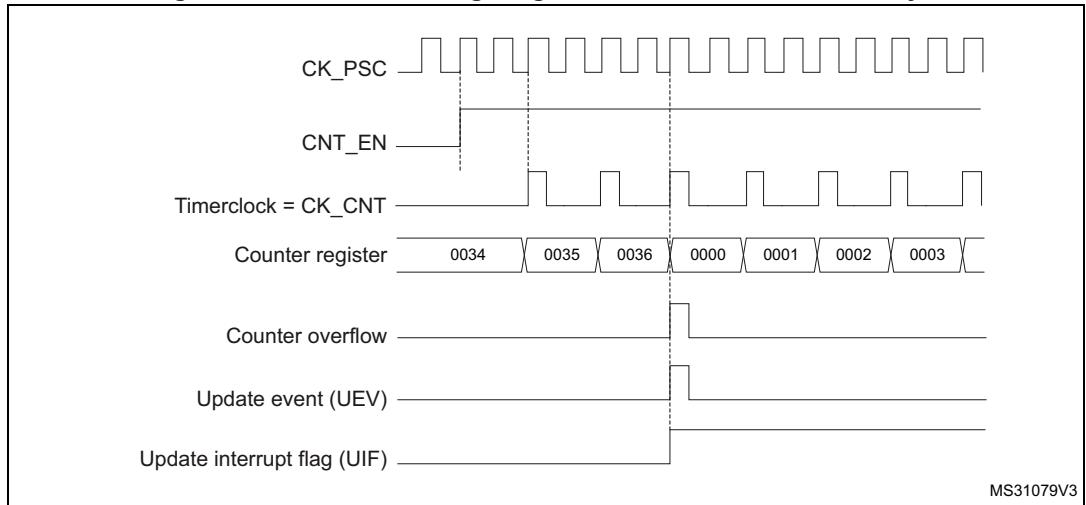
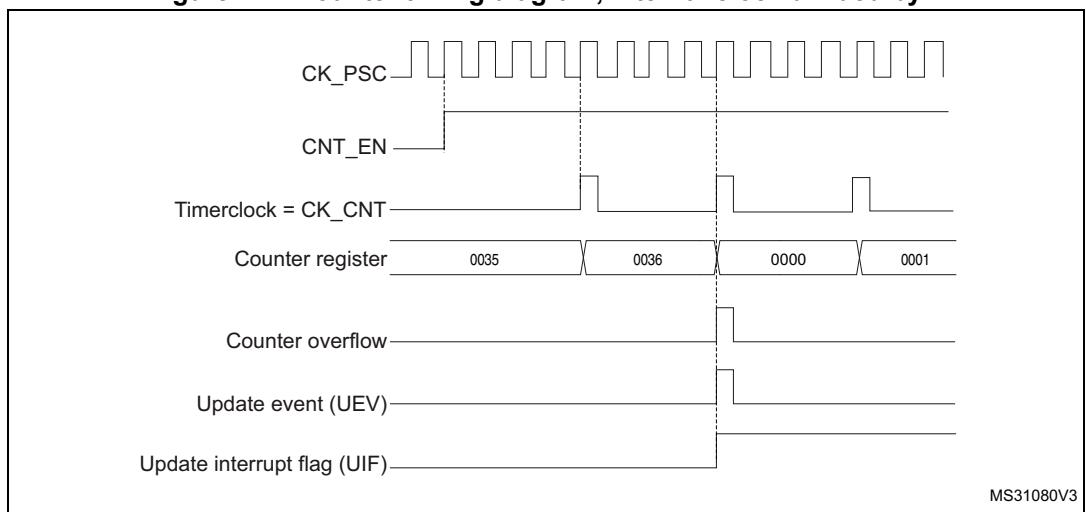
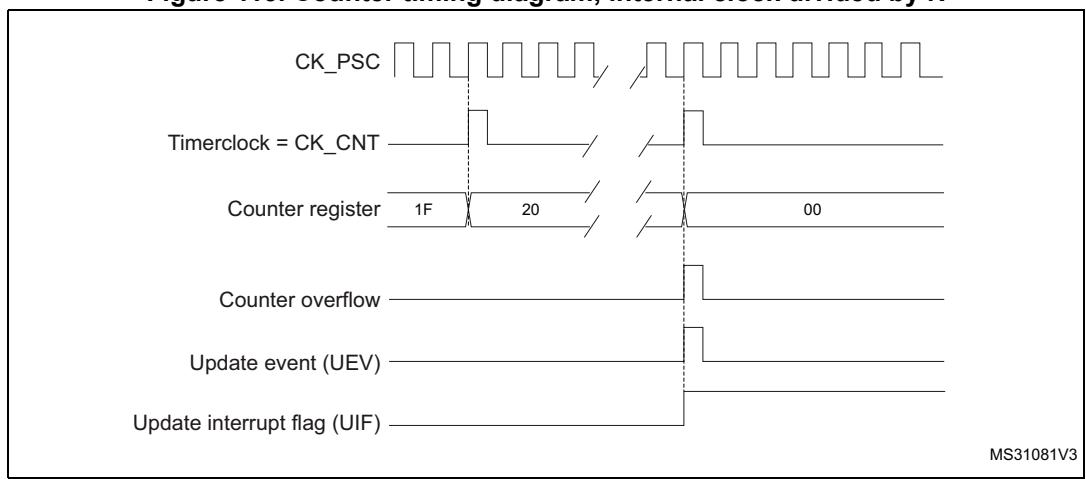
When an update event occurs, all the registers are updated and the update flag (UIF bit in TIMx\_SR register) is set (depending on the URS bit):

- 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).

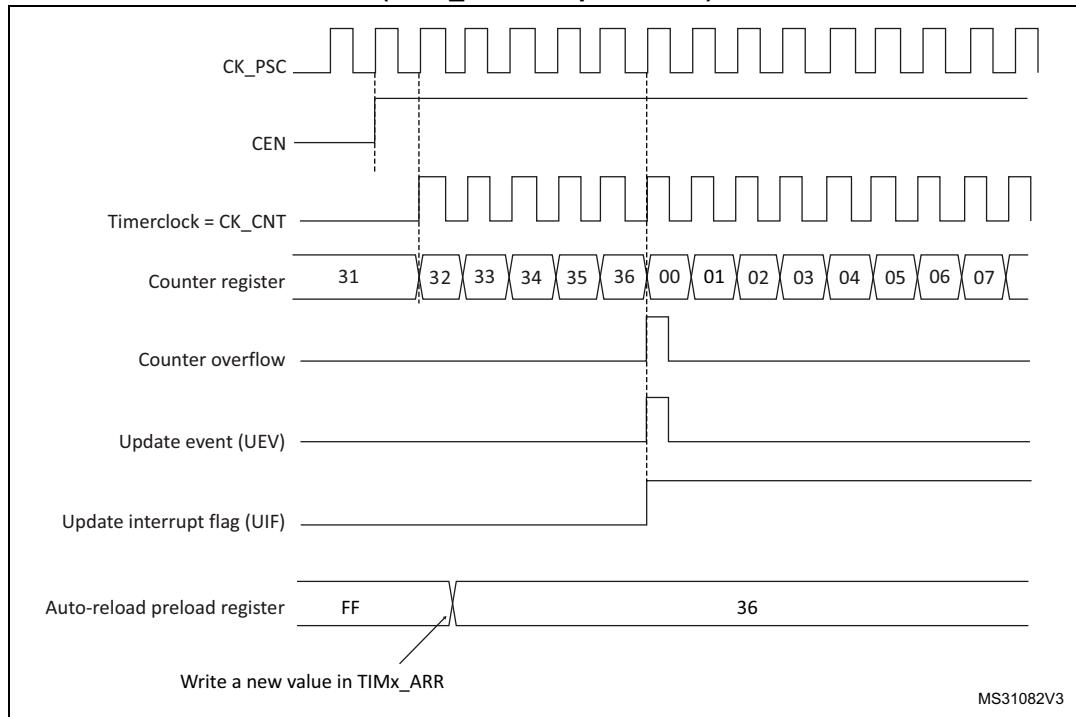
The following figures show some examples of the counter behavior for different clock frequencies when TIMx\_ARR=0x36.

**Figure 112. Counter timing diagram, internal clock divided by 1**

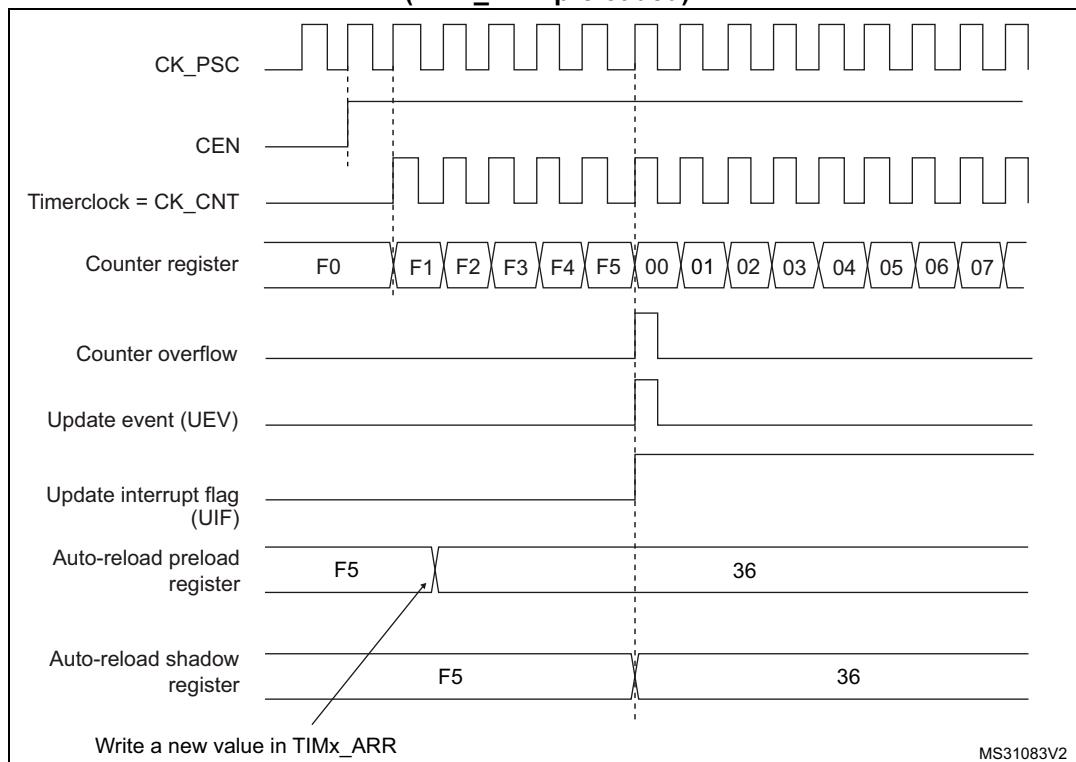


**Figure 113. Counter timing diagram, internal clock divided by 2****Figure 114. Counter timing diagram, internal clock divided by 4****Figure 115. Counter timing diagram, internal clock divided by N**

**Figure 116. Counter timing diagram, update event when ARPE=0  
(TIMx\_ARR not preloaded)**



**Figure 117. 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 plus one (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.

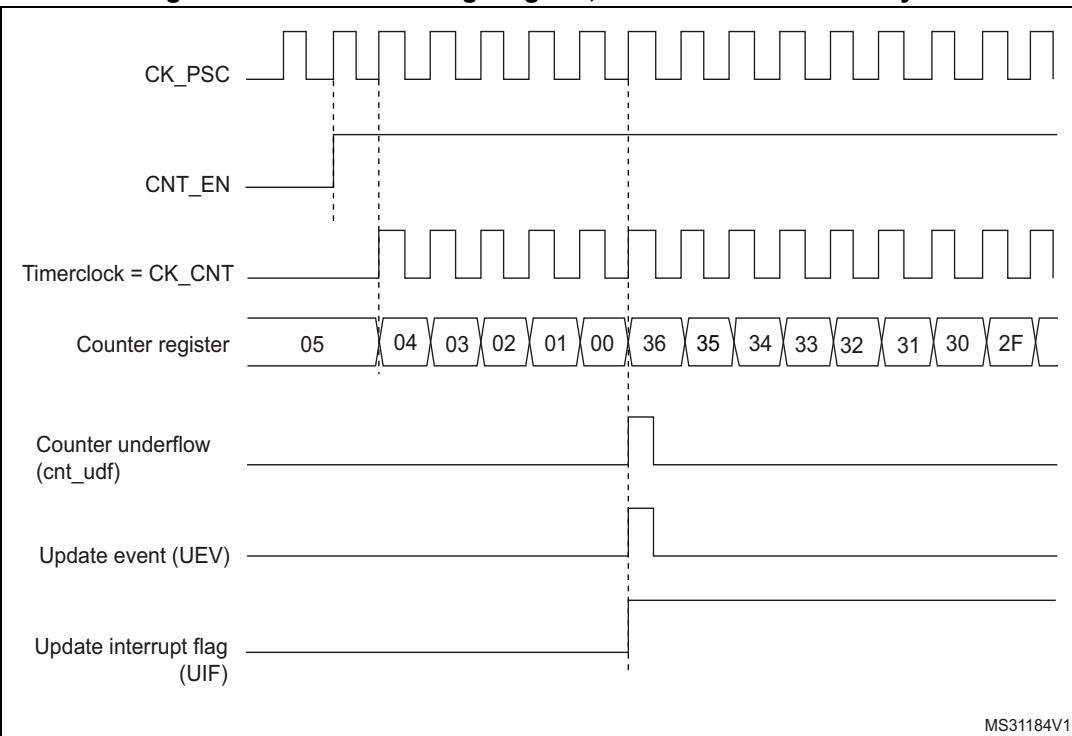
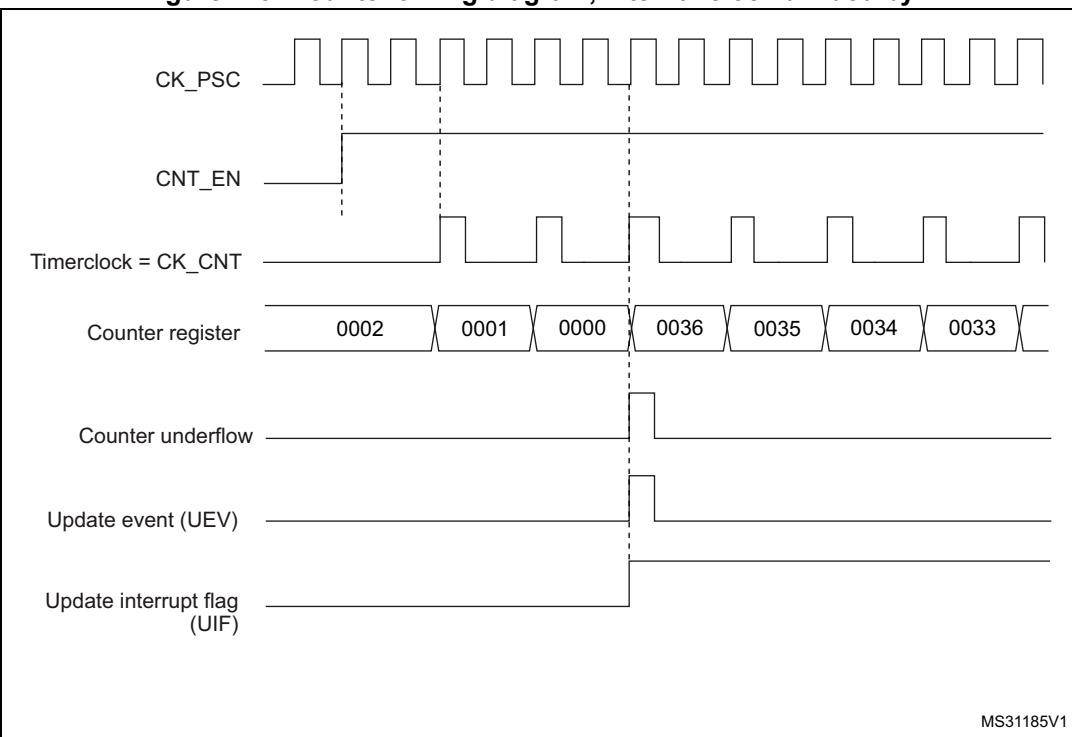
The UEV update event can be disabled by software by setting the UDIS bit in TIMx\_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until UDIS bit has been written to 0. However, the counter restarts from the current auto-reload value, whereas the counter of the prescaler restarts from 0 (but the prescale rate doesn't change).

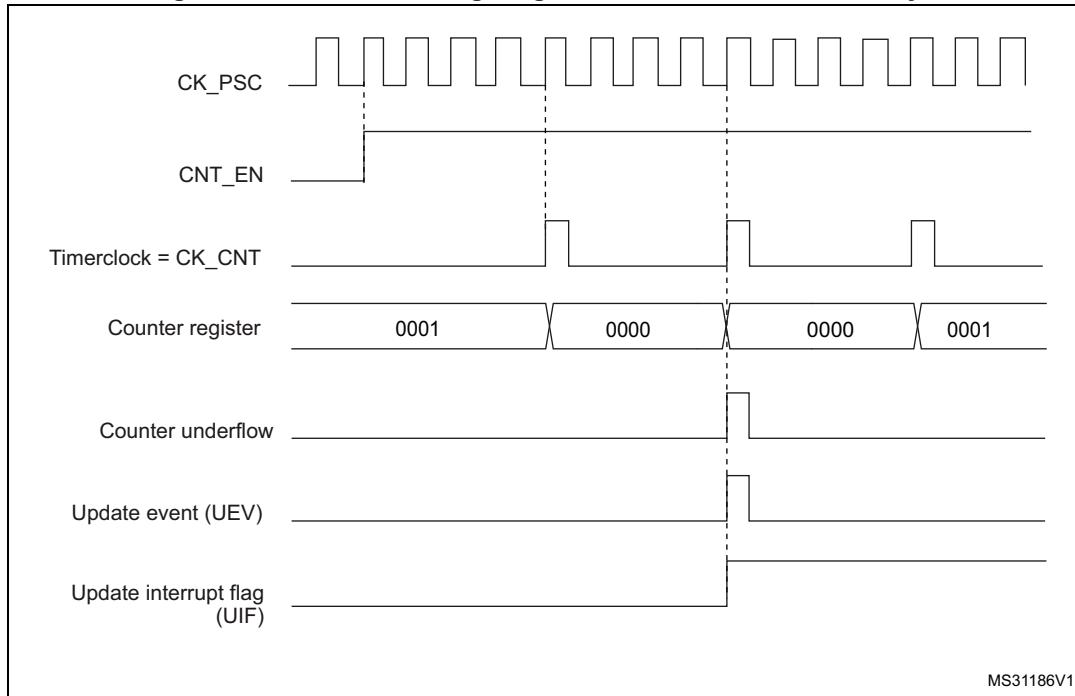
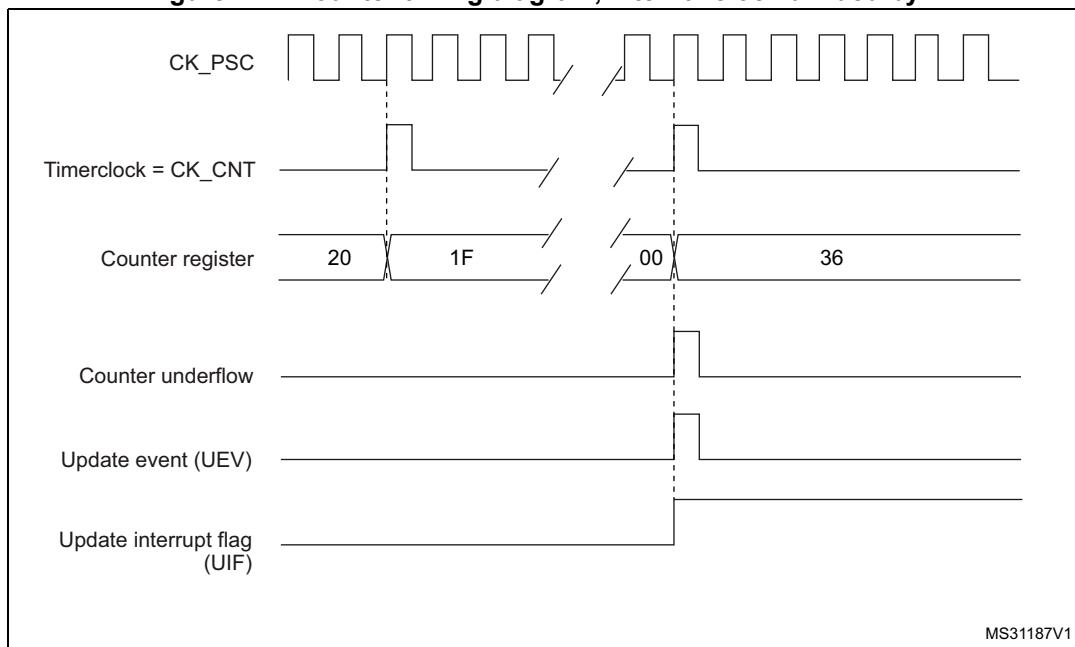
In addition, if the URS bit (update request selection) in TIMx\_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on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

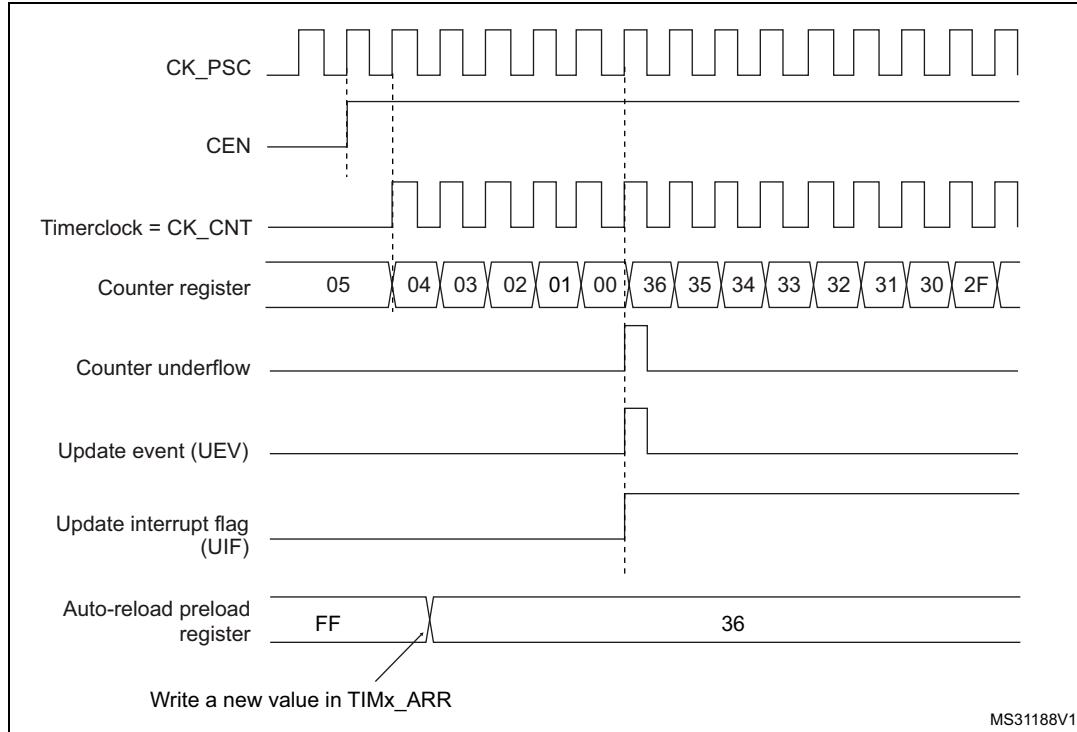
When an update event occurs, all the registers are updated and the update flag (UIF bit in TIMx\_SR register) is set (depending on the URS bit):

- 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 118. Counter timing diagram, internal clock divided by 1****Figure 119. Counter timing diagram, internal clock divided by 2**

**Figure 120. Counter timing diagram, internal clock divided by 4****Figure 121. Counter timing diagram, internal clock divided by N**

**Figure 122. Counter timing diagram, update event when repetition counter is not used**

MS31188V1

### 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") or 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

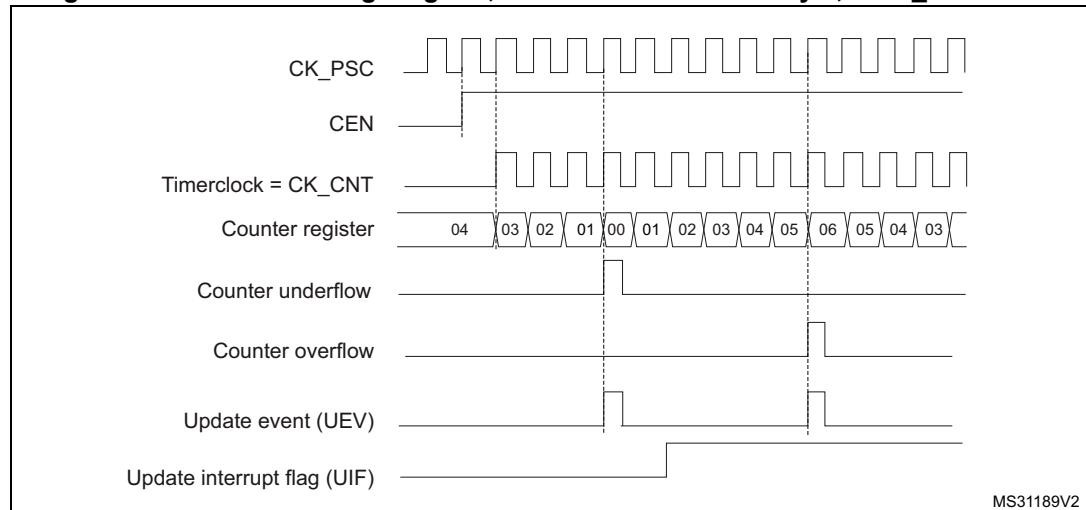
DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on the capture event.

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).

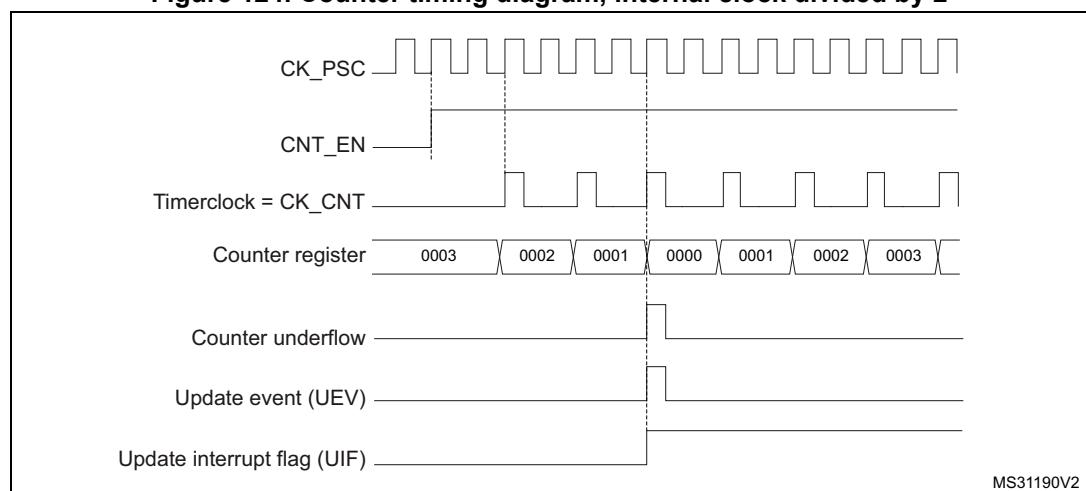
The following figures show some examples of the counter behavior for different clock frequencies.

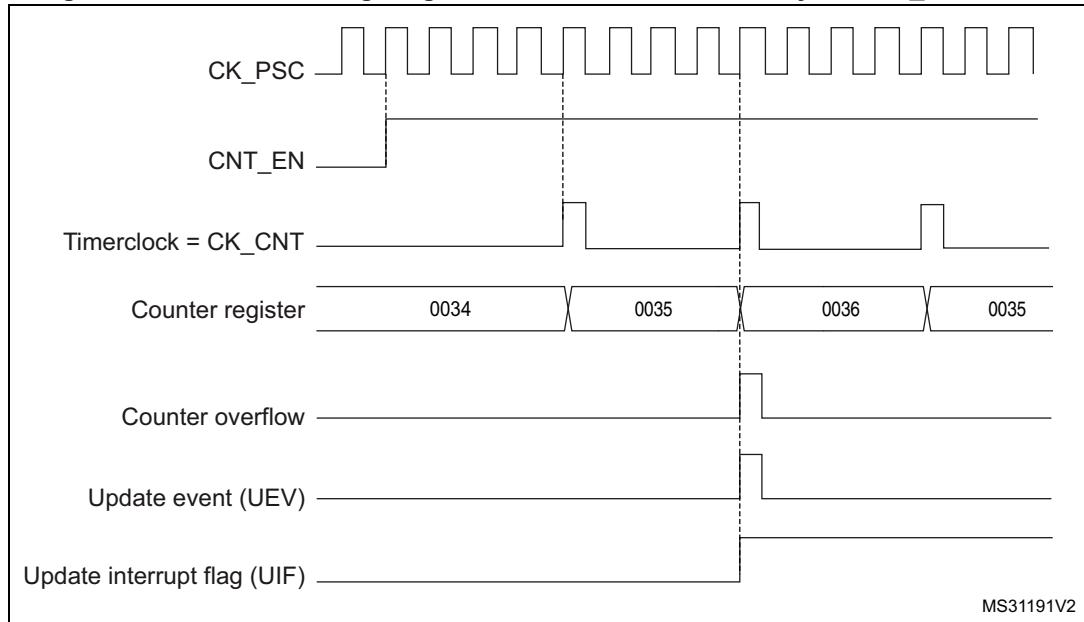
**Figure 123. Counter timing diagram, internal clock divided by 1, TIMx\_ARR = 0x6**



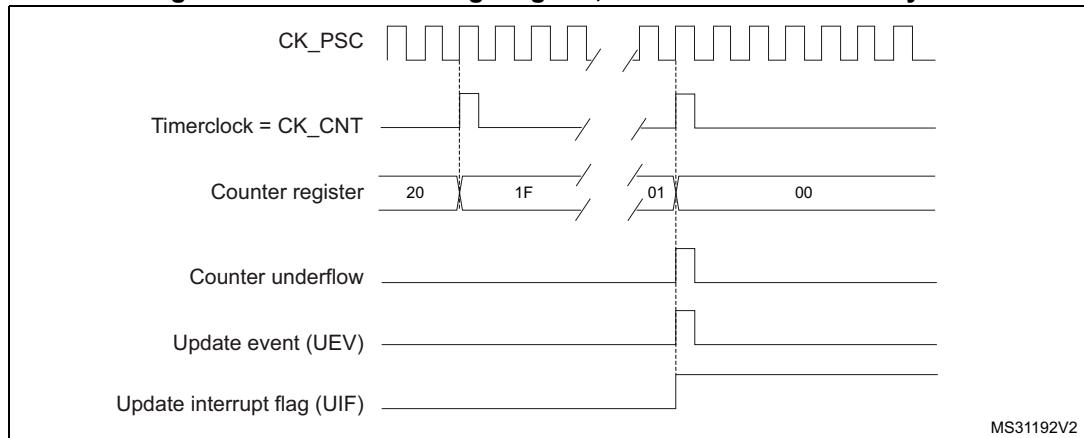
1. Here, center-aligned mode 1 is used (for more details refer to [Section 16.4: TIM1&TIM8 registers](#)).

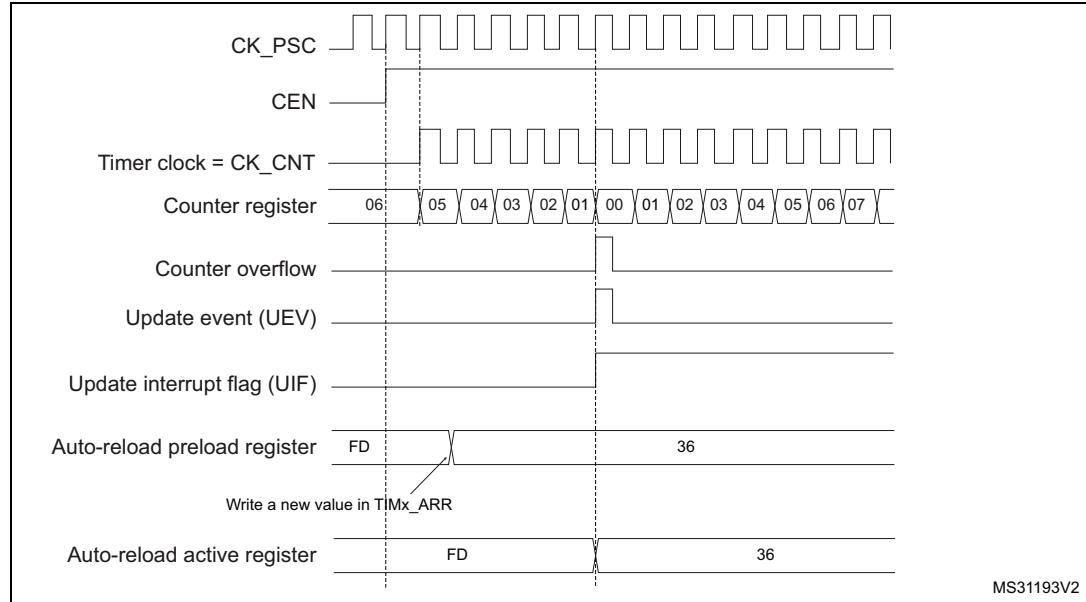
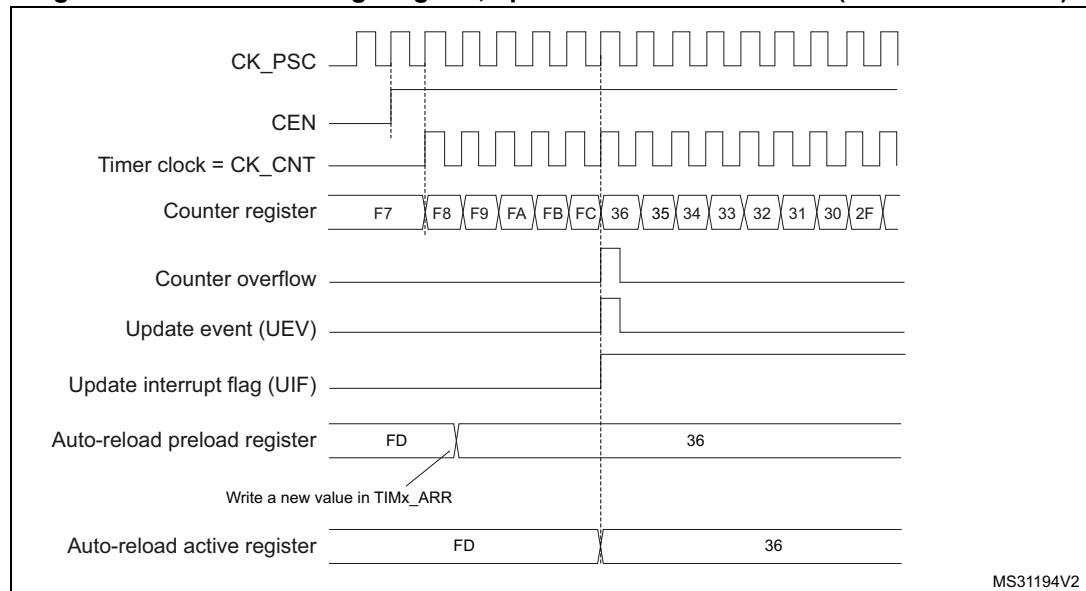
**Figure 124. Counter timing diagram, internal clock divided by 2**



**Figure 125. 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 126. Counter timing diagram, internal clock divided by N**

**Figure 127. Counter timing diagram, update event with ARPE=1 (counter underflow)****Figure 128. Counter timing diagram, update event with ARPE=1 (counter overflow)**

### 16.3.3 Repetition counter

[Section 16.3.1: 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.

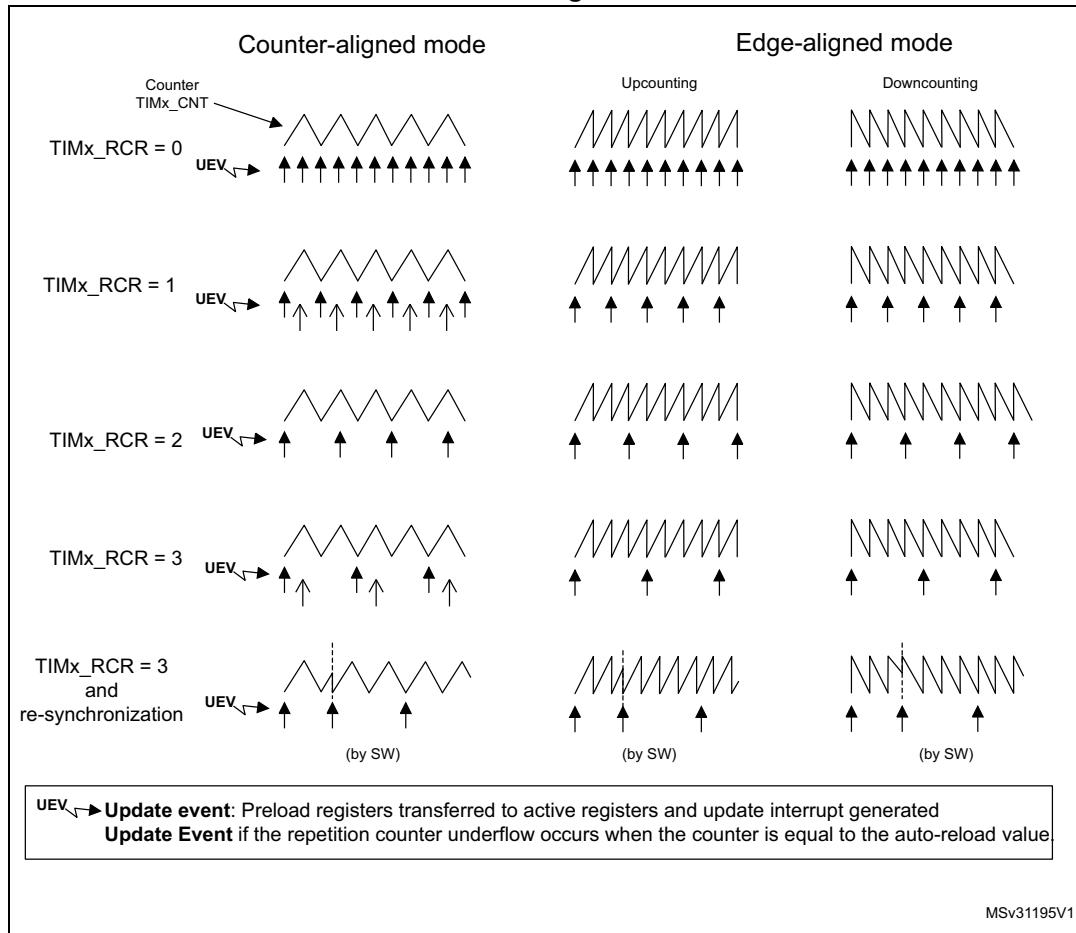
The repetition counter is decremented:

- 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 128 PWM cycles, it makes it possible to update the duty cycle twice per PWM period. When refreshing compare registers only once per PWM period in center-aligned mode, maximum resolution is  $2 \times T_{ck}$ , due to the symmetry of the pattern.

The repetition counter is an auto-reload type; the repetition rate is maintained as defined by the TIMx\_RCR register value (refer to [Figure 129](#)). When the update event is generated by software (by setting the UG bit in TIMx\_EGR register) or by hardware through the slave mode controller, it occurs immediately whatever the value of the repetition counter is and the repetition counter is reloaded with the content of the TIMx\_RCR register.

In center-aligned mode, for odd values of RCR, the update event occurs either on the overflow or on the underflow depending on when the RCR register was written and when the counter was started. If the RCR was written before starting the counter, the UEV occurs on the overflow. If the RCR was written after starting the counter, the UEV occurs on the underflow. For example for RCR = 3, the UEV is generated on each 4th overflow or underflow event depending on when RCR was written.

**Figure 129. Update rate examples depending on mode and TIMx\_RCR register settings**



MSv31195V1

### 16.3.4 Clock selection

The counter clock can be provided by the following clock sources:

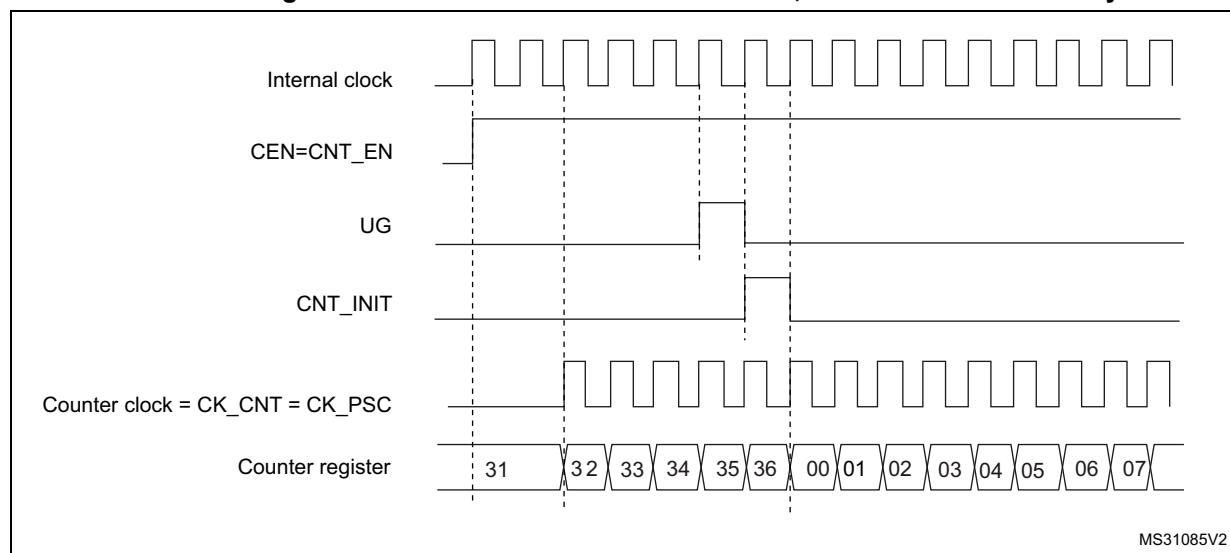
- Internal clock (CK\_INT)
- External clock mode1: external input pin
- External clock mode2: external trigger input ETR
- Internal trigger inputs (ITRx): using one timer as prescaler for another timer, for example, you can configure Timer 1 to act as a prescaler for Timer 2. Refer to [Using one timer as prescaler for another timer](#) for more details.

#### Internal clock source (CK\_INT)

If the slave mode controller is disabled (SMS=000), then the CEN, DIR (in the TIMx\_CR1 register) and UG bits (in the TIMx\_EGR register) are actual control bits and can be changed only by software (except UG which remains cleared automatically). As soon as the CEN bit is written to 1, the prescaler is clocked by the internal clock CK\_INT.

[Figure 130](#) shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

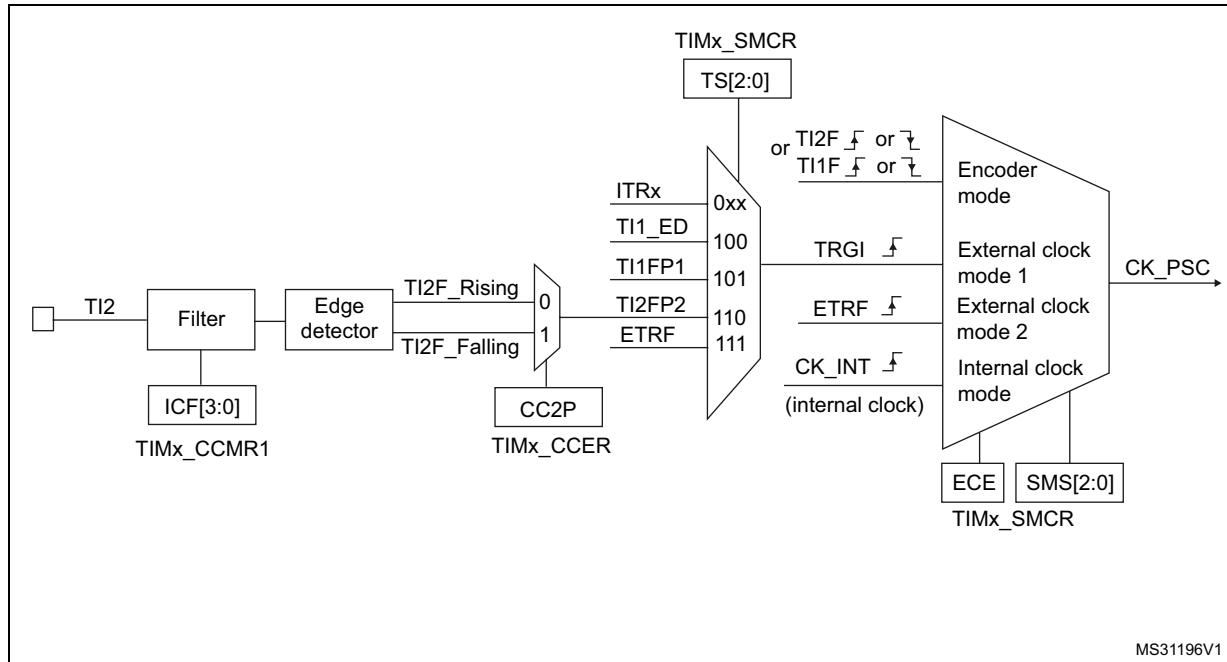
**Figure 130. Control circuit in normal mode, internal clock divided by 1**



#### 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 131. TI2 external clock connection example



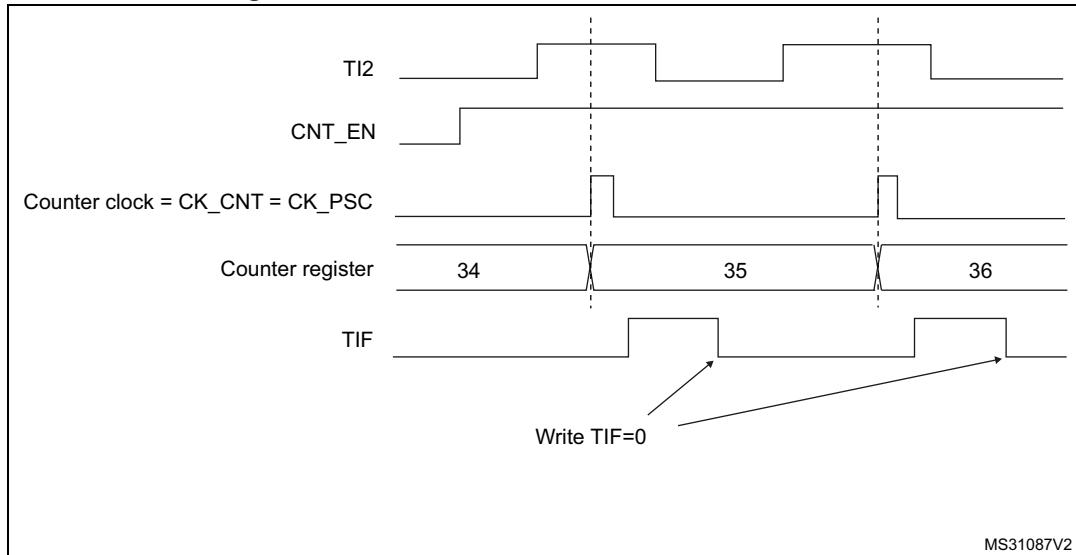
For example, to configure the upcounter to count in response to a rising edge on the TI2 input, use the following procedure:

1. Configure channel 2 to detect rising edges on the TI2 input by writing CC2S = '01' in the TIMx\_CCMR1 register.
2. Configure the input filter duration by writing the IC2F[3:0] bits in the TIMx\_CCMR1 register (if no filter is needed, keep IC2F=0000).
3. Select rising edge polarity by writing CC2P=0 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.

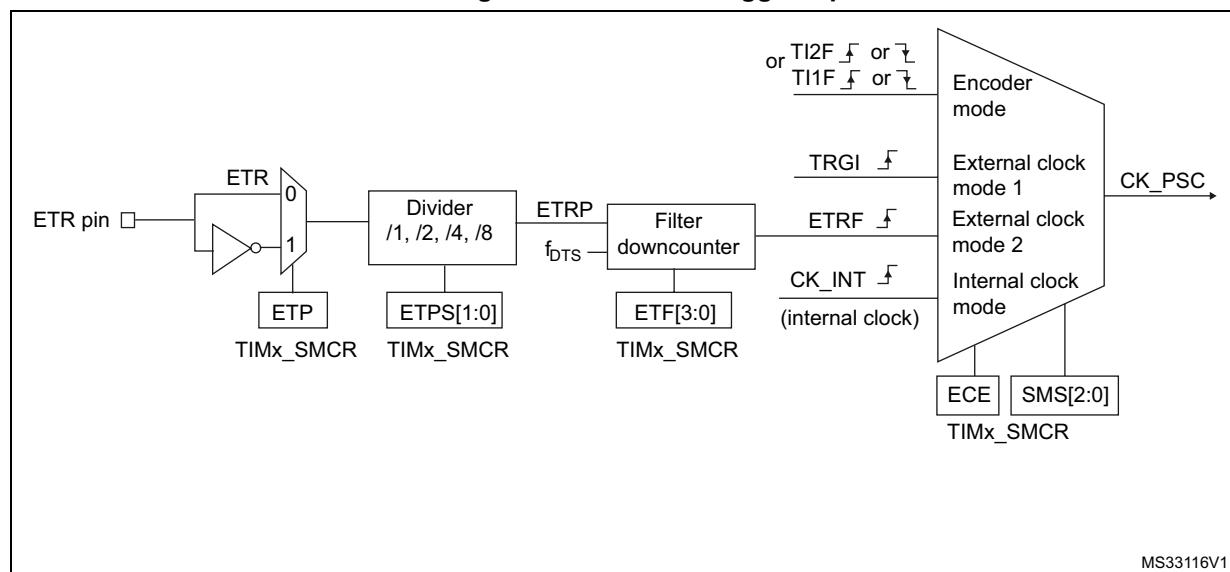
The delay between the rising edge on TI2 and the actual clock of the counter is due to the resynchronization circuit on TI2 input.

**Figure 132. 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 133](#) gives an overview of the external trigger input block.

**Figure 133. External trigger input block**

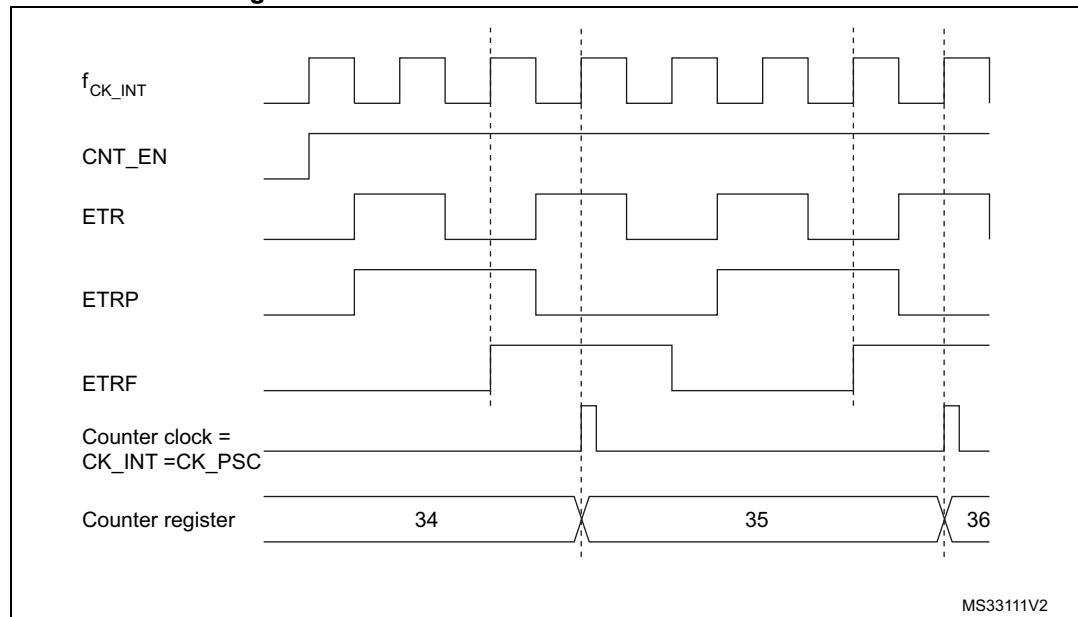
For example, to configure the upcounter to count each 2 rising edges on ETR, use the following procedure:

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 134. Control circuit in external clock mode 2**



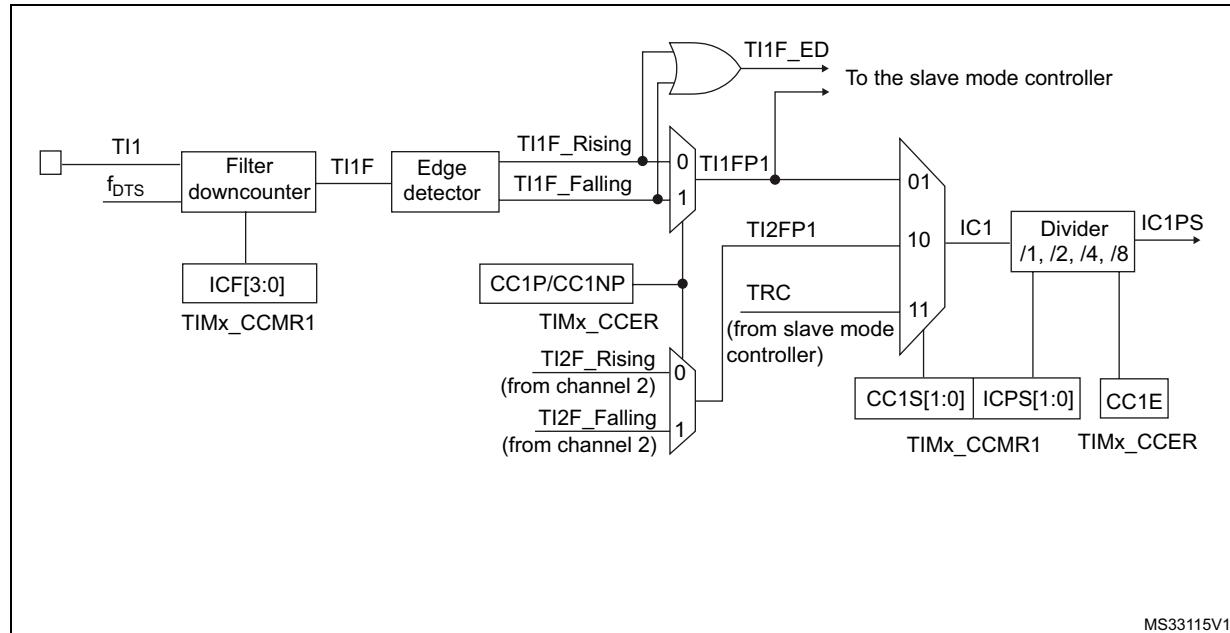
### 16.3.5 Capture/compare channels

Each Capture/Compare channel is built around a capture/compare register (including a shadow register), a input stage for capture (with digital filter, multiplexing and prescaler) and an output stage (with comparator and output control).

*Figure 135 to Figure 138* 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).

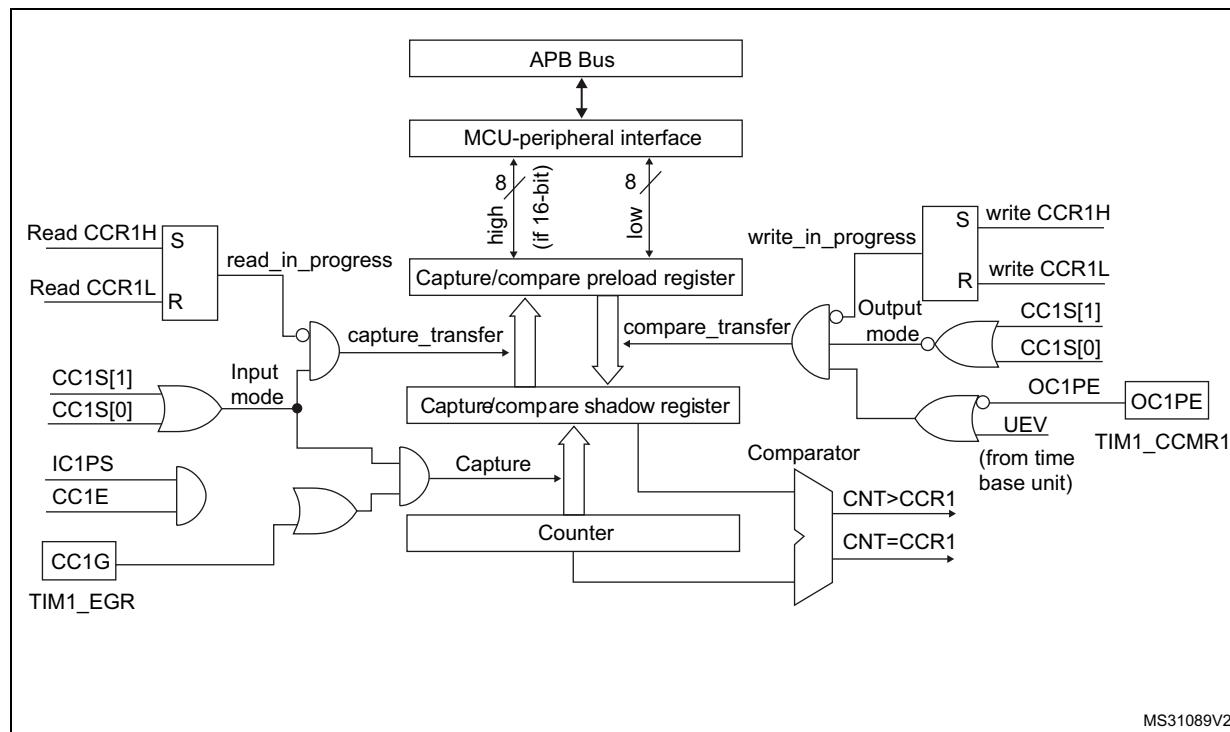
Figure 135. Capture/compare channel (example: channel 1 input stage)



MS33115V1

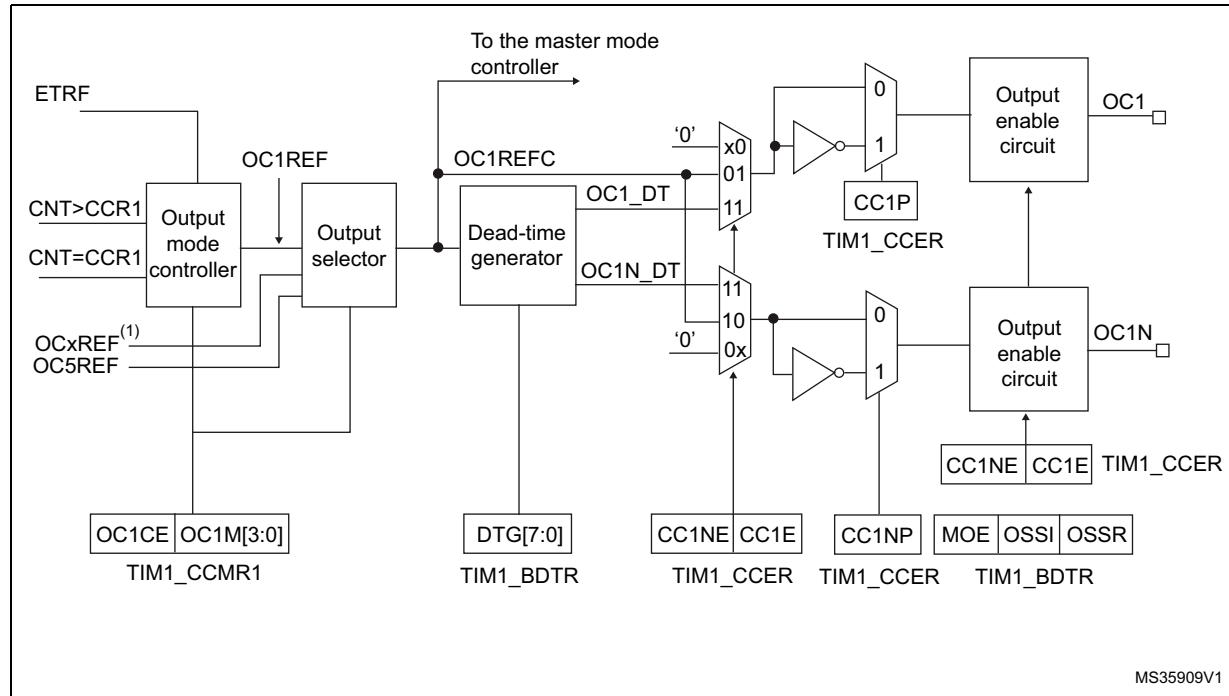
The output stage generates an intermediate waveform which is then used for reference: OCxRef (active high). The polarity acts at the end of the chain.

Figure 136. Capture/compare channel 1 main circuit



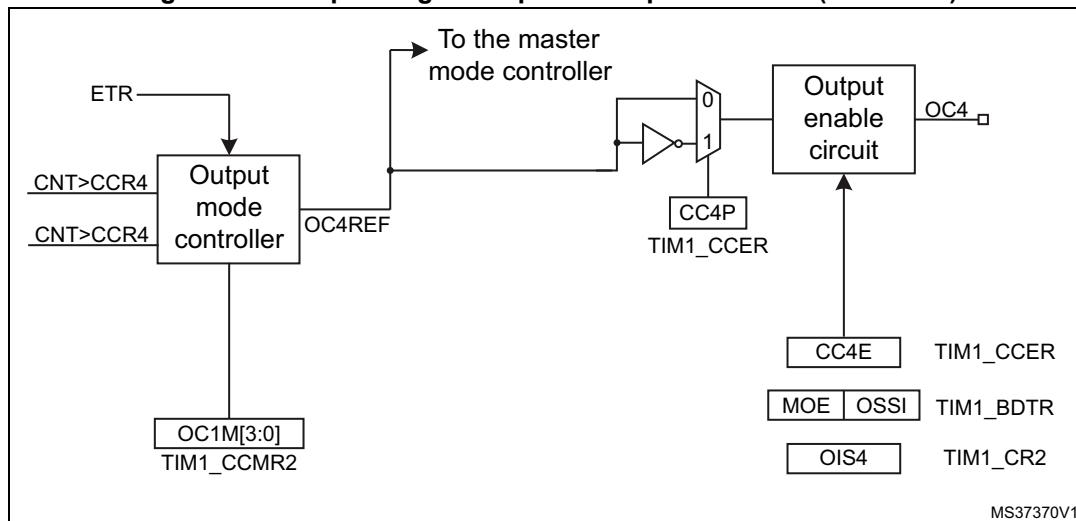
MS31089V2

Figure 137. Output stage of capture/compare channel (channels 1 to 3)



MS35909V1

Figure 138. Output stage of capture/compare channel (channel 4)



MS37370V1

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.

### 16.3.6 Input capture mode

In Input capture mode, the Capture/Compare Registers (TIMx\_CCRx) are used to latch the value of the counter after a transition detected by the corresponding IC<sub>x</sub> signal. When a capture occurs, the corresponding CC<sub>x</sub>IF flag (TIMx\_SR register) is set and an interrupt or a DMA request can be sent if they are enabled. If a capture occurs while the CC<sub>x</sub>IF flag was already high, then the over-capture flag CC<sub>x</sub>OF (TIMx\_SR register) is set. CC<sub>x</sub>IF can be cleared by software by writing it to '0' or by reading the captured data stored in the TIMx\_CCRx register. CC<sub>x</sub>OF is cleared when you write it to '0'.

The following example shows how to capture the counter value in TIMx\_CCR1 when TI1 input rises. To do this, use the following procedure:

- Select the active input: TIMx\_CCR1 must be linked to the TI1 input, so write the CC1S bits to 01 in the TIMx\_CCMR1 register. As soon as CC1S becomes different from 00, the channel is configured in input and the TIMx\_CCR1 register becomes read-only.
- Program the input filter duration you need with respect to the signal you connect to the timer (by programming ICxF bits in the TIMx\_CCMRx register if the input is a TIx input). 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<sub>DTS</sub> frequency). Then write IC1F bits to 0011 in the TIMx\_CCMR1 register.
- Select the edge of the active transition on the TI1 channel by writing CC1P and CC1NP bits to 0 in the TIMx\_CCER register (rising edge in this case).
- Program the input prescaler. In our example, we wish the capture to be performed at each valid transition, so the prescaler is disabled (write IC1PS bits to '00' in the TIMx\_CCMR1 register).
- Enable capture from the counter into the capture register by setting the CC1E bit in the TIMx\_CCER register.
- If needed, enable the related interrupt request by setting the CC1IE bit in the TIMx\_DIER register, and/or the DMA request by setting the CC1DE bit in the TIMx\_DIER register.

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 CC<sub>x</sub>G bit in the TIMx\_EGR register.*

### 16.3.7 PWM input mode

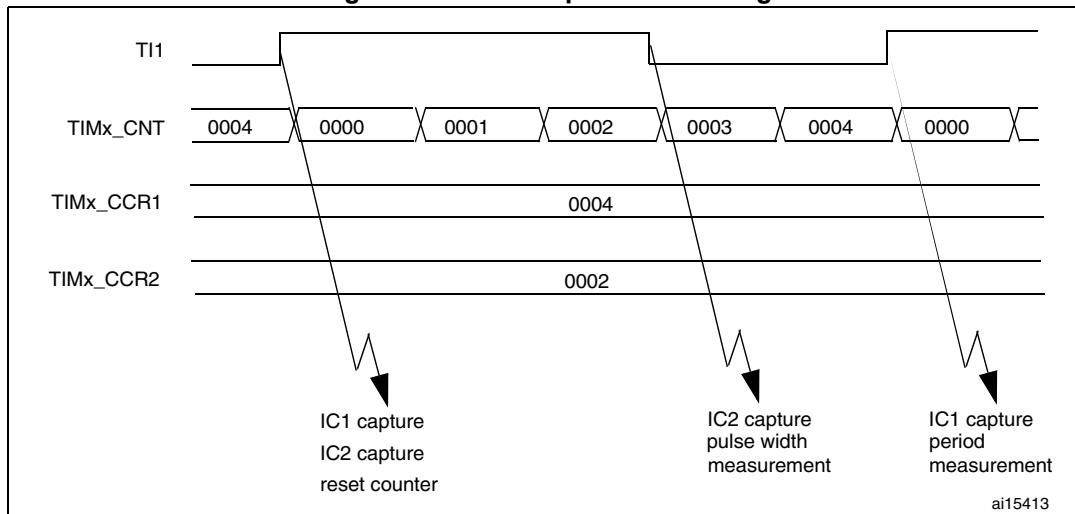
This mode is a particular case of input capture mode. The procedure is the same except:

- Two ICx signals are mapped on the same TIx input.
- These 2 ICx signals are active on edges with opposite polarity.
- One of the two TIxFP signals is selected as trigger input and the slave mode controller is configured in reset mode.

For example, you can measure the period (in TIMx\_CCR1 register) and the duty cycle (in TIMx\_CCR2 register) of the PWM applied on TI1 using the following procedure (depending on CK\_INT frequency and prescaler value):

- Select the active input for TIMx\_CCR1: write the CC1S bits to 01 in the TIMx\_CCMR1 register (TI1 selected).
- Select the active polarity for TI1FP1 (used both for capture in TIMx\_CCR1 and counter clear): write the CC1P and CC1NP bits to '0' (active on rising edge).
- Select the active input for TIMx\_CCR2: write the CC2S bits to 10 in the TIMx\_CCMR1 register (TI1 selected).
- Select the active polarity for TI1FP2 (used for capture in TIMx\_CCR2): write the CC2P and CC2NP bits to '1' (active on falling edge).
- Select the valid trigger input: write the TS bits to 101 in the TIMx\_SMCR register (TI1FP1 selected).
- Configure the slave mode controller in reset mode: write the SMS bits to 100 in the TIMx\_SMCR register.
- Enable the captures: write the CC1E and CC2E bits to '1' in the TIMx\_CCER register.

**Figure 139. PWM input mode timing**



### 16.3.8 Forced output mode

In output mode (CCxS bits = 00 in the TIMx\_CCMRx register), each output compare signal (OCxREF and then OCx/OCxN) can be forced to active or inactive level directly by software, independently of any comparison between the output compare register and the counter.

To force an output compare signal (OCXREF/OCx) to its active level, 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.

For example: CCxP=0 (OCx active high) => OCx is forced to high level.

The OCxREF signal can be forced low by writing the OCxM bits to 100 in the TIMx\_CCMRx register.

Anyway, the comparison between the TIMx\_CCRx shadow register and the counter is still performed and allows the flag to be set. Interrupt and DMA requests can be sent accordingly. This is described in the output compare mode section below.

### 16.3.9 Output compare mode

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

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

- Assigns the corresponding output pin to a programmable value defined by the output compare mode (OCxM bits in the TIMx\_CCMRx register) and the output polarity (CCxP bit in the TIMx\_CCER register). The output pin can keep its level (OCXM=000), be set active (OCxM=001), be set inactive (OCxM=010) or can toggle (OCxM=011) on match.
- Sets a flag in the interrupt status register (CCxIF bit in the TIMx\_SR register).
- Generates an interrupt if the corresponding interrupt mask is set (CCxEIE 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).

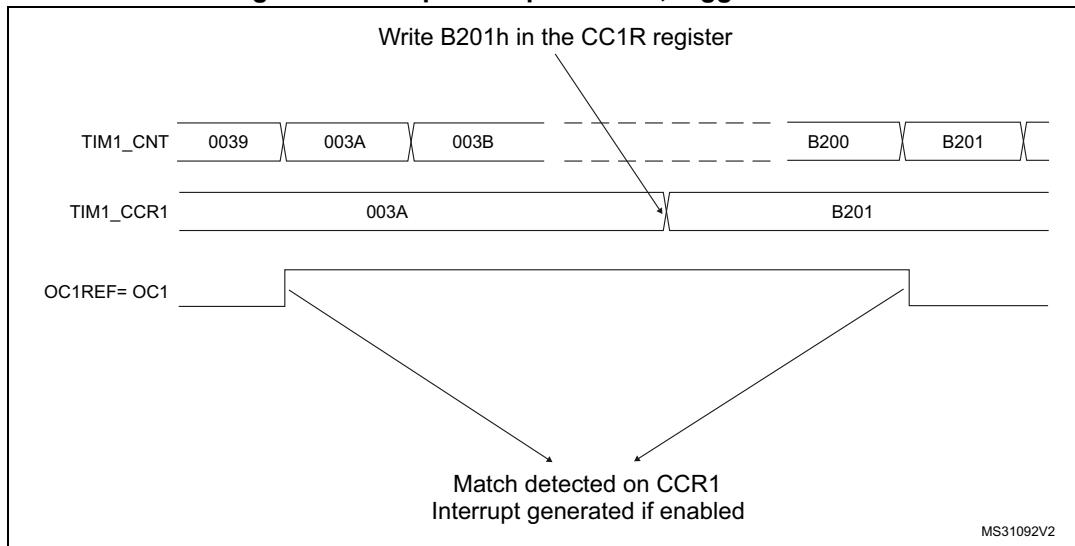
The TIMx\_CCRx registers can be programmed with or without preload registers using the OCxPE bit in the TIMx\_CCMRx register.

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 CCxEIE bit if an interrupt request is to be generated.
4. Select the output mode. For example:
  - Write OCxM = 011 to toggle OCx output pin when CNT matches CCRx
  - Write OCxPE = 0 to disable preload register
  - Write CCxP = 0 to select active high polarity
  - Write CCxE = 1 to enable the output
5. Enable the counter by setting the CEN bit in the TIMx\_CR1 register.

The TIMx\_CCRx register can be updated at any time by software to control the output waveform, provided that the preload register is not enabled (OCxPE='0', else TIMx\_CCRx shadow register is updated only at the next update event UEV). An example is given in [Figure 140](#).

**Figure 140. Output compare mode, toggle on OC1.**

### 16.3.10 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 a combination of the CCxE, CCxNE, MOE, OSS1 and OSSR bits (TIMx\_CCER and TIMx\_BDTR registers). Refer to the TIMx\_CCER register description for more details.

In PWM mode (1 or 2), TIMx\_CNT and TIMx\_CCRx are always compared to determine whether TIMx\_CCRx  $\leq$  TIMx\_CNT or TIMx\_CNT  $\leq$  TIMx\_CCRx (depending on the direction of the counter).

The timer is able to generate PWM in edge-aligned mode or center-aligned mode depending on the CMS bits in the TIMx\_CR1 register.

### PWM edge-aligned mode

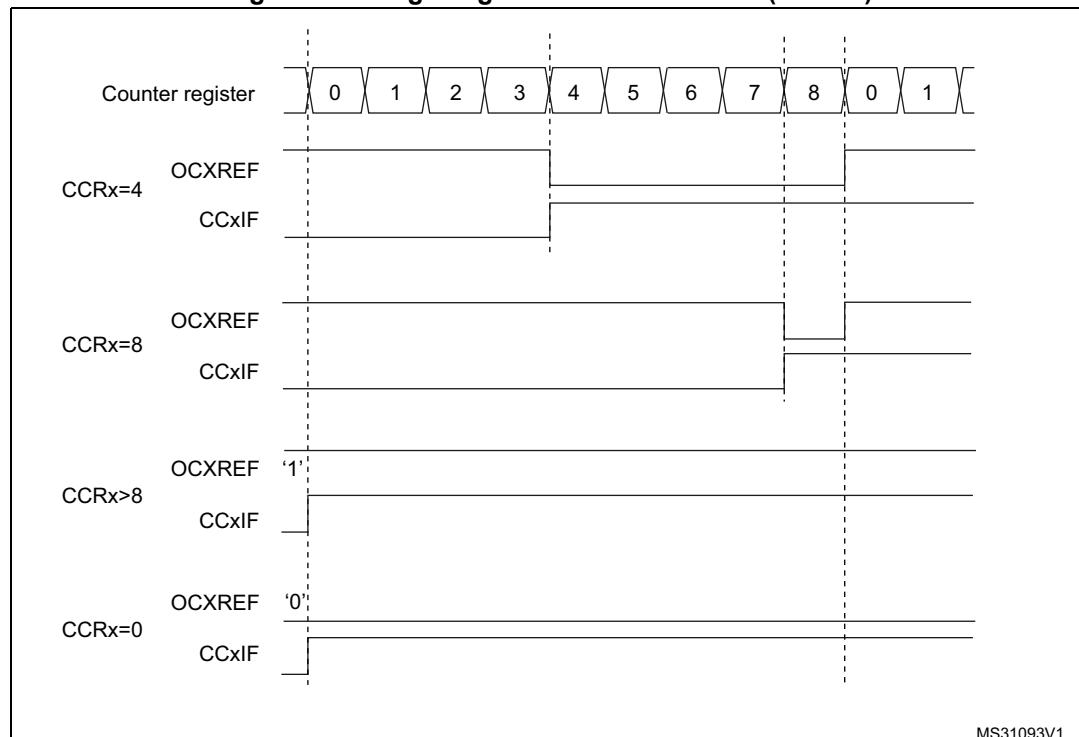
- Upcounting configuration

Upcounting is active when the DIR bit in the TIMx\_CR1 register is low. Refer to [Upcounting mode](#).

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  $\text{TIMx\_CCR}_x$  is greater than the auto-reload value (in  $\text{TIMx\_ARR}$ ) then OCxREF is held at '1'. If the compare value is 0 then OCxRef is held at '0'.

[Figure 141](#) shows some edge-aligned PWM waveforms in an example where  $\text{TIMx\_ARR}=8$ .

**Figure 141. Edge-aligned PWM waveforms (ARR=8)**



MS31093V1

- Downcounting configuration

Downcounting is active when DIR bit in TIMx\_CR1 register is high. Refer to [Downcounting mode](#).

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  $\text{TIMx\_CCR}_x$  is greater than the auto-reload value in  $\text{TIMx\_ARR}$ , then OCxREF is held at '1'. 0% PWM is not possible in this mode.

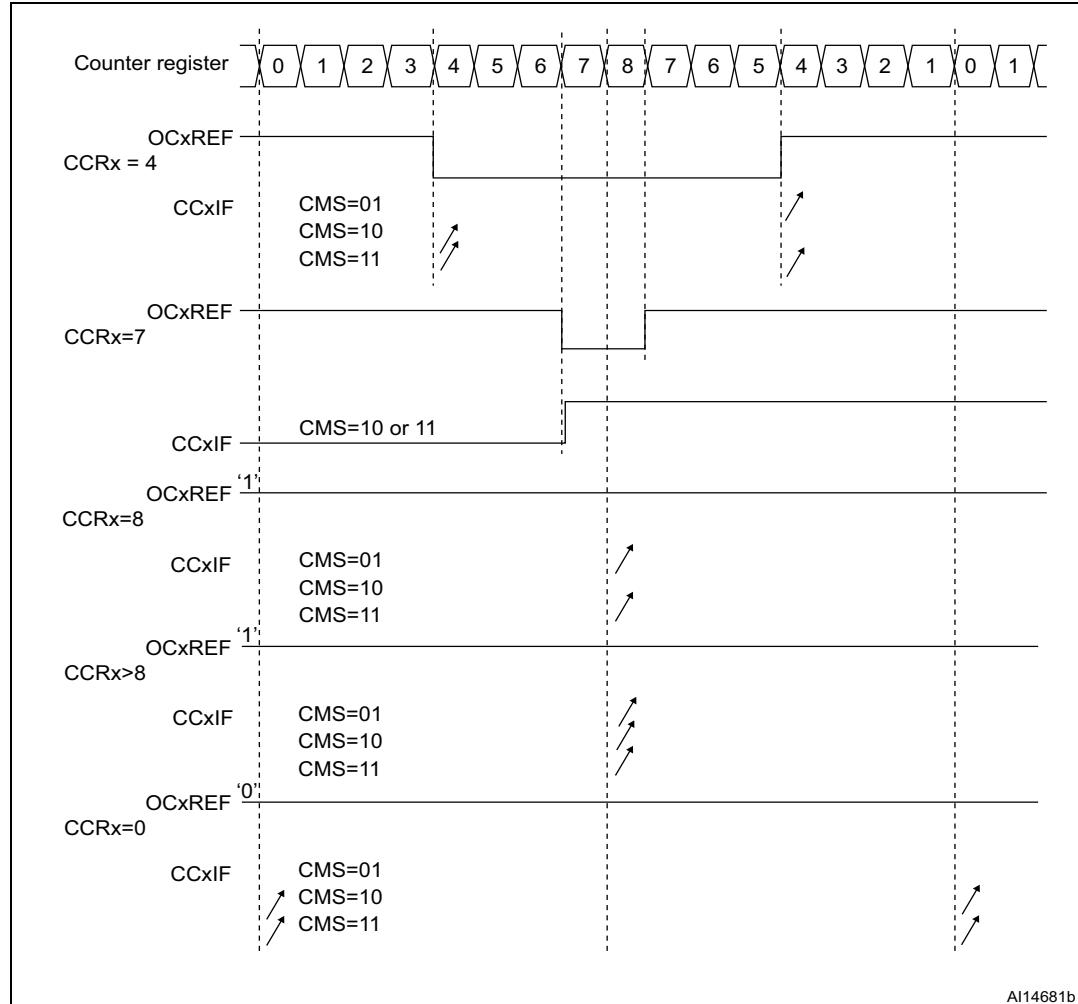
### PWM center-aligned mode

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 [Center-aligned mode \(up/down counting\)](#).

[Figure 142](#) shows some center-aligned PWM waveforms in an example where:

- TIMx\_ARR=8,
- PWM mode is the PWM mode 1,
- The flag is set when the counter counts down corresponding to the center-aligned mode 1 selected for CMS=01 in TIMx\_CR1 register.

**Figure 142. Center-aligned PWM waveforms (ARR=8)**



**Hints on using center-aligned mode:**

- When starting in center-aligned mode, the current up-down configuration is used. It means that the counter counts up or down depending on the value written in the DIR bit in the TIMx\_CR1 register. Moreover, the DIR and CMS bits must not be changed at the same time by the software.
- Writing to the counter while running in center-aligned mode is not recommended as it can lead to unexpected results. In particular:
  - The direction is not updated if 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.

**16.3.11 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<sub>x</sub> or complementary OC<sub>xN</sub>) independently for each output. This is done by writing to the CC<sub>xP</sub> and CC<sub>xNP</sub> bits in the TIM<sub>x</sub>\_CCER register.

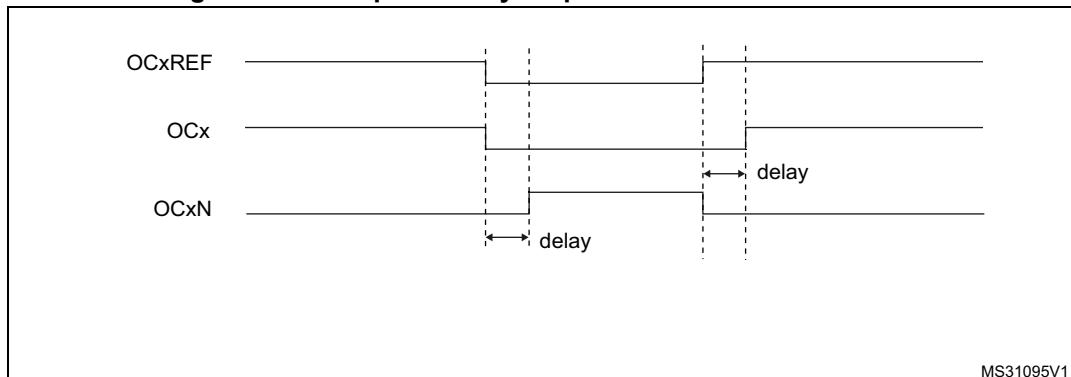
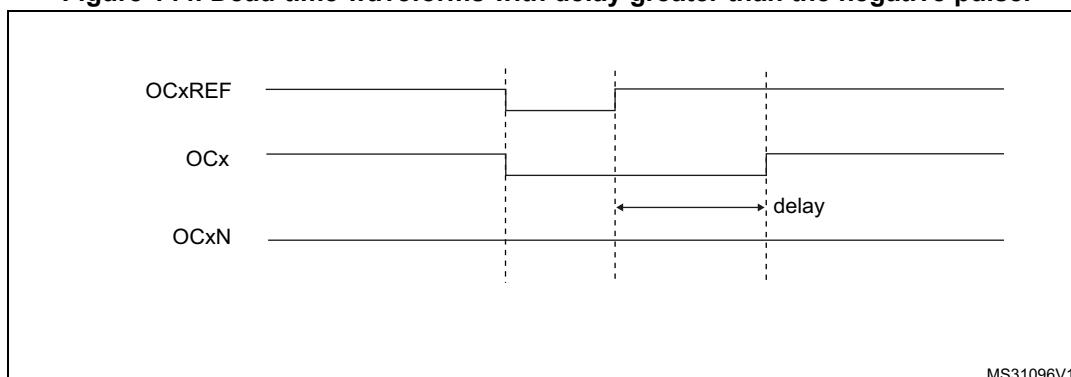
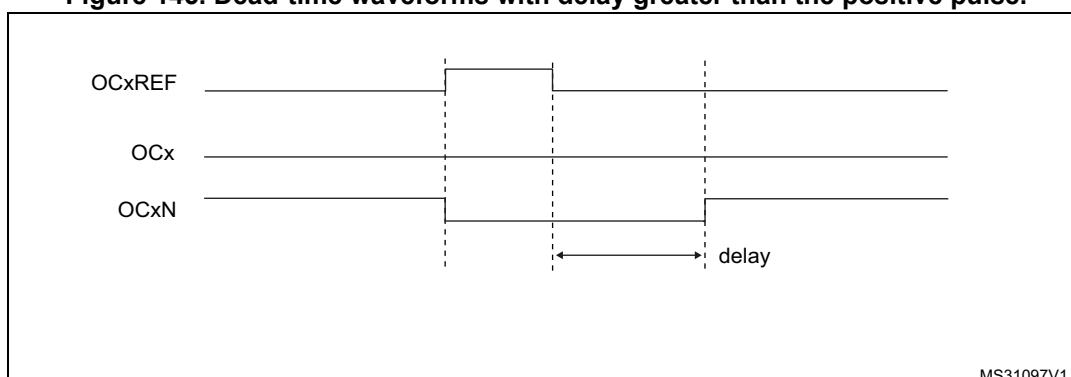
The complementary signals OC<sub>x</sub> and OC<sub>xN</sub> are activated by a combination of several control bits: the CC<sub>xE</sub> and CC<sub>xNE</sub> bits in the TIM<sub>x</sub>\_CCER register and the MOE, OIS<sub>x</sub>, OIS<sub>xN</sub>, OSS<sub>I</sub> and OSS<sub>R</sub> bits in the TIM<sub>x</sub>\_BDTR and TIM<sub>x</sub>\_CR2 registers. Refer to [Table 109](#) for more details. In particular, the dead-time is activated when switching to the IDLE state (MOE falling down to 0).

Dead-time insertion is enabled by setting both CC<sub>xE</sub> and CC<sub>xNE</sub> bits, and the MOE bit if the break circuit is present. DTG[7:0] bits of the TIM<sub>x</sub>\_BDTR register are used to control the dead-time generation for all channels. From a reference waveform OC<sub>xREF</sub>, it generates 2 outputs OC<sub>x</sub> and OC<sub>xN</sub>. If OC<sub>x</sub> and OC<sub>xN</sub> are active high:

- The OC<sub>x</sub> output signal is the same as the reference signal except for the rising edge, which is delayed relative to the reference rising edge.
- The OC<sub>xN</sub> output signal is the opposite of the reference signal except for the rising edge, which is delayed relative to the reference falling edge.

If the delay is greater than the width of the active output (OC<sub>x</sub> or OC<sub>xN</sub>) then the corresponding pulse is not generated.

The following figures show the relationships between the output signals of the dead-time generator and the reference signal OC<sub>xREF</sub>. (we suppose CC<sub>xP</sub>=0, CC<sub>xNP</sub>=0, MOE=1, CC<sub>xE</sub>=1 and CC<sub>xNE</sub>=1 in these examples).

**Figure 143. Complementary output with dead-time insertion.****Figure 144. Dead-time waveforms with delay greater than the negative pulse.****Figure 145. 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 16.4.18: TIM1&TIM8 break and dead-time register \(TIMx\\_BDTR\)](#) 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.

### 16.3.12 Using the break function

When using the break function, the output enable signals and inactive levels are modified according to additional control bits (MOE, OSS1 and OSSR bits in the TIMx\_BDTR register, OISx and OISxN bits in the TIMx\_CR2 register). In any case, the OCx and OCxN outputs cannot be set both to active level at a given time. Refer to [Table 109](#) for more details.

The break source can be either the break input pin or a clock failure event, generated by the Clock Security System (CSS), from the Reset Clock Controller. For further information on the Clock Security System, refer to [Section 6.2.7](#).

When exiting from reset, the break circuit is disabled and the MOE bit is low. You can enable the break function by setting the BKE bit in the TIMx\_BDTR register. The break input polarity can be selected by configuring the BKP bit in the same register. BKE and BKP can be modified at the same time. When the BKE and BKP bits are written, a delay of 1 APB clock cycle is applied before the writing is effective. Consequently, it is necessary to wait 1 APB clock period to correctly read back the bit after the write operation.

Because MOE falling edge can be asynchronous, a resynchronization circuit has been inserted between the actual signal (acting on the outputs) and the synchronous control bit (accessed in the TIMx\_BDTR register). It results in some delays between the asynchronous and the synchronous signals. In particular, if you write MOE to 1 whereas it was low, you must insert a delay (dummy instruction) before reading it correctly. This is because you write the asynchronous signal and read the synchronous signal.

When a break occurs (selected level on the break input):

- The MOE bit is cleared asynchronously, putting the outputs in inactive state, idle state or in reset state (selected by the OSS1 bit). This feature functions even if the MCU oscillator is off.
- Each output channel is driven with the level programmed in the OISx bit in the TIMx\_CR2 register as soon as MOE=0. If OSS1=0 then the timer releases the enable output else the enable output remains high.
- When complementary outputs are used:
  - The outputs are first put in reset state inactive state (depending on the polarity). This is done asynchronously so that it works even if no clock is provided to the timer.
  - If the timer clock is still present, then the dead-time generator is reactivated in order to drive the outputs with the level programmed in the OISx and OISxN bits after a dead-time. Even in this case, OCx and OCxN cannot be driven to their

active level together. Note that because of the resynchronization on MOE, the dead-time duration is a bit longer than usual (around 2 ck\_tim clock cycles).

- If OSS1=0 then the timer releases the enable outputs else the enable outputs remain or become high as soon as one of the CCxE or CCxNE bits is high.
- The break status flag (BIF bit in the TIMx\_SR register) is set. An interrupt can be generated if the BIE bit in the TIMx\_DIER register is set. 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. This can be used to perform a regulation, for instance. Else, MOE remains low until you write it to '1' again. In this case, it can be used for security and you can connect the break input to an alarm from power drivers, thermal sensors or any security components.

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

The break can be generated by the BRK input which has a programmable polarity and an enable bit BKE in the TIMx\_BDTR Register.

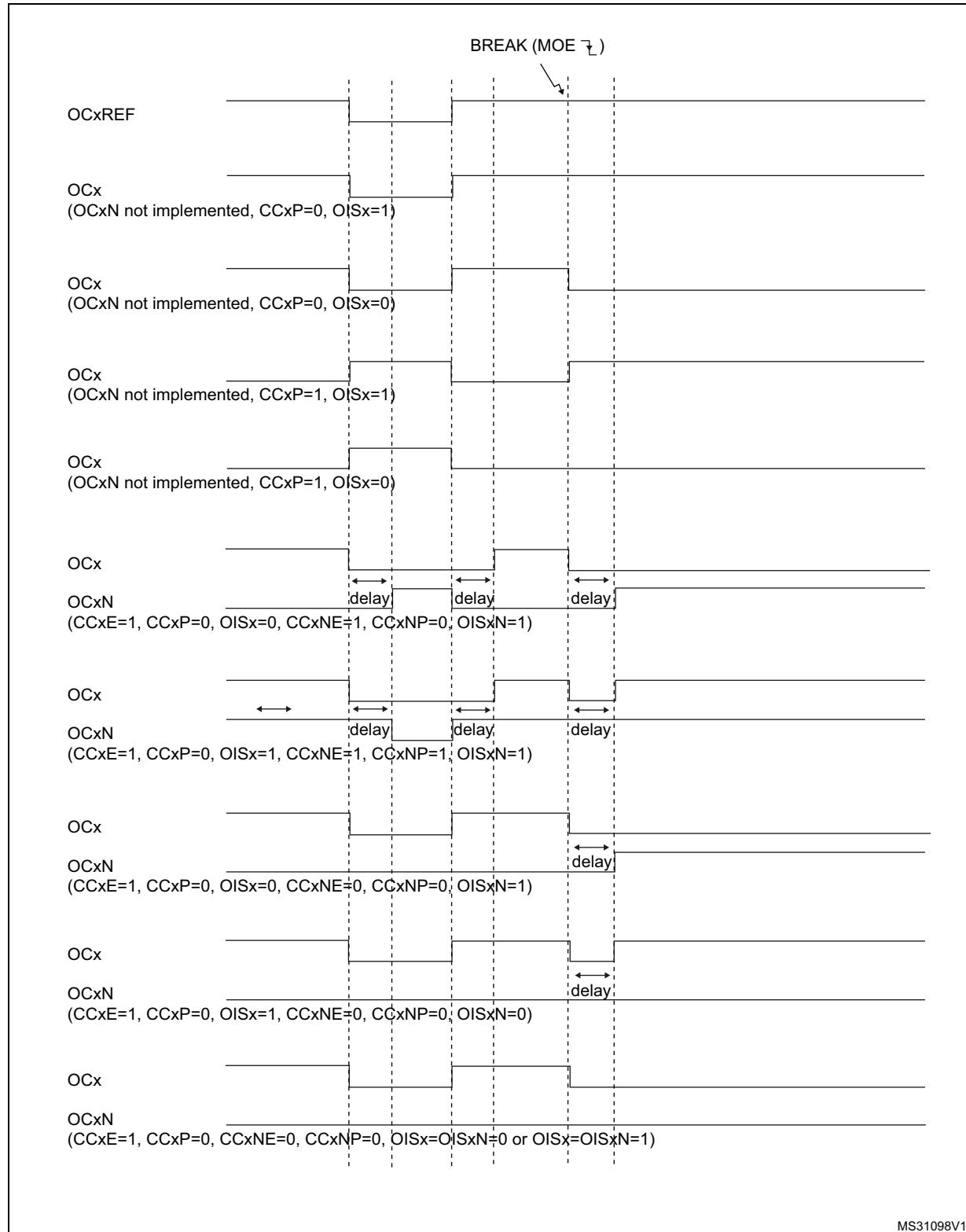
There are two solutions to generate a break:

- By using the BRK input which has a programmable polarity and an enable bit BKE in the TIMx\_BDTR register
- By software through the BG bit of the TIMx\_EGR register.

In addition to the break input and the output management, a write protection has been implemented inside the break circuit to safeguard the application. It allows you to freeze the configuration of several parameters (dead-time duration, OCx/OCxN polarities and state when disabled, OCxM configurations, break enable and polarity). You can choose from 3 levels of protection selected by the LOCK bits in the TIMx\_BDTR register. Refer to [Section 16.4.18: TIM1&TIM8 break and dead-time register \(TIMx\\_BDTR\)](#). The LOCK bits can be written only once after an MCU reset.

*Figure 146* shows an example of behavior of the outputs in response to a break.

Figure 146. Output behavior in response to a break.



MS31098V1

### 16.3.13 Clearing the OCxREF signal on an external event

The OC<sub>x</sub>REF signal for a given channel can be driven Low by applying a High level to the ETRF input (OC<sub>x</sub>CE enable bit of the corresponding TIM<sub>x</sub>\_CCMR<sub>x</sub> register set to '1'). The OC<sub>x</sub>REF signal remains Low until the next update event, UEV, occurs.

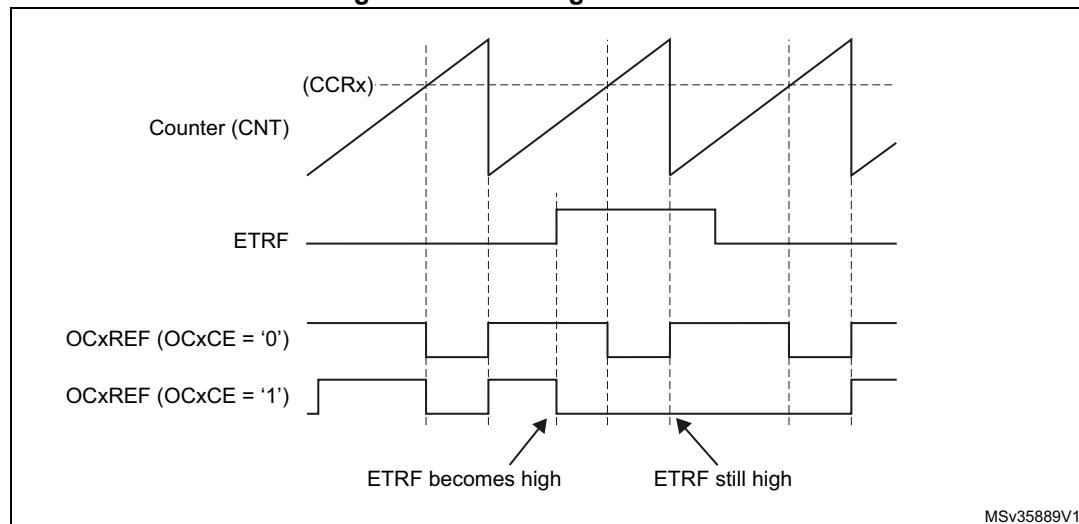
This function can only be used in output compare and PWM modes, and does not work in forced mode.

For example, the ETR signal can be connected to the output of a comparator to be used for current handling. In this case, the ETR must be configured as follow:

1. The External Trigger Prescaler should be kept off: bits ETPS[1:0] of the TIM<sub>x</sub>\_SMCR register set to '00'.
2. The external clock mode 2 must be disabled: bit ECE of the TIM<sub>x</sub>\_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 147* shows the behavior of the OC<sub>x</sub>REF signal when the ETRF Input becomes High, for both values of the enable bit OC<sub>x</sub>CE. In this example, the timer TIM<sub>x</sub> is programmed in PWM mode.

**Figure 147. Clearing TIM<sub>x</sub> OC<sub>x</sub>REF**



Note:

*In case of a PWM with a 100% duty cycle (if CCR<sub>x</sub>>ARR), then OC<sub>x</sub>REF is enabled again at the next counter overflow.*

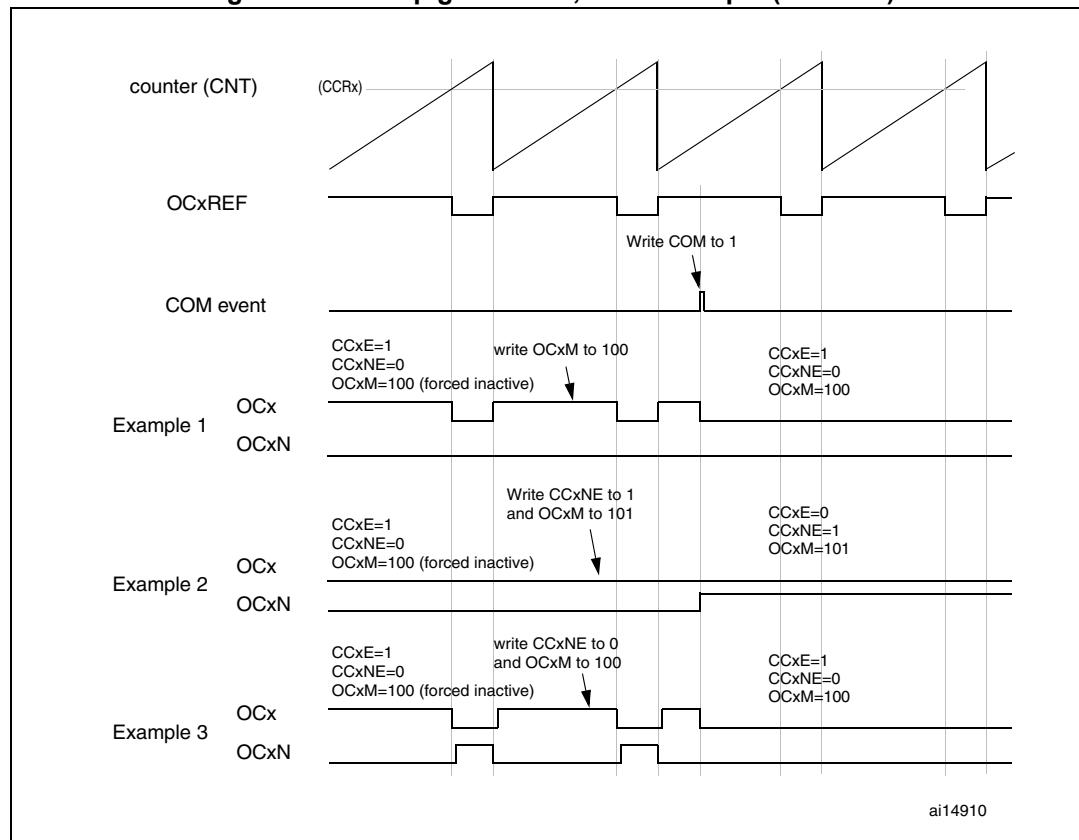
### 16.3.14 6-step PWM generation

When complementary outputs are used on a channel, preload bits are available on the OCxM, CCxE and CCxNE bits. The preload bits are transferred to the shadow bits at the COM commutation event. Thus 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).

*Figure 148* describes the behavior of the OCx and OCxN outputs when a COM event occurs, in 3 different examples of programmed configurations.

**Figure 148. 6-step generation, COM example (OSSR=1)**



### 16.3.15 One-pulse mode

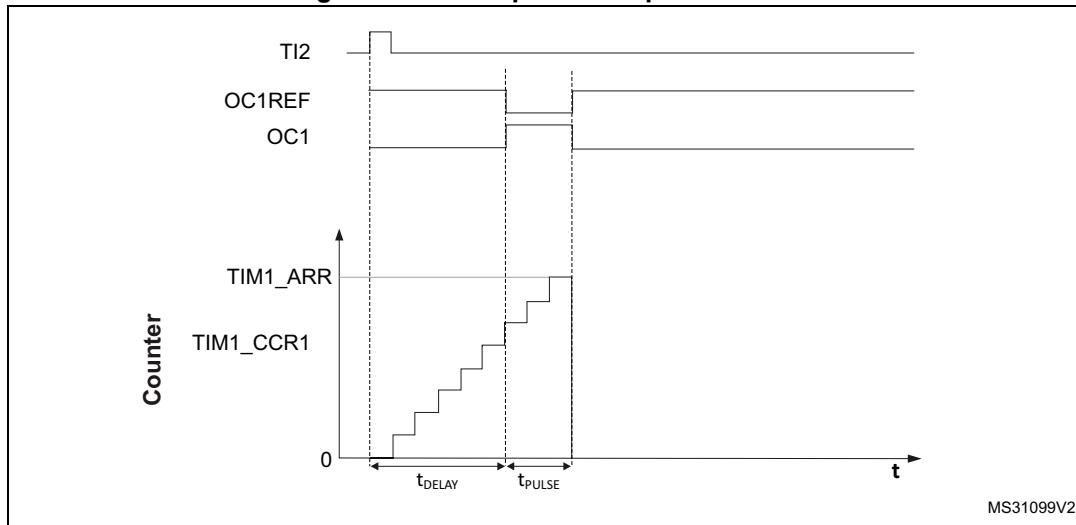
One-pulse mode (OPM) is a particular case of the previous modes. It allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length after a programmable delay.

Starting the counter can be controlled through the slave mode controller. Generating the waveform can be done in output compare mode or PWM mode. 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 149. Example of one pulse mode.**



For example you may want to generate a positive pulse on OC1 with a length of  $t_{PULSE}$  and after a delay of  $t_{DELAY}$  as soon as a positive edge is detected on the TI2 input pin.

Let's use TI2FP2 as trigger 1:

- Map TI2FP2 to TI2 by writing CC2S='01' in the TIMx\_CCMR1 register.
- TI2FP2 must detect a rising edge, write CC2P='0' and CC2NP='0' in the TIMx\_CCER register.
- Configure TI2FP2 as trigger for the slave mode controller (TRGI) by writing TS='110' in the TIMx\_SMCR register.
- TI2FP2 is used to start the counter by writing SMS to '110' in the TIMx\_SMCR register (trigger mode).

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

- The  $t_{DELAY}$  is defined by the value written in the TIMx\_CCR1 register.
- The  $t_{PULSE}$  is defined by the difference between the auto-reload value and the compare value (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.

### 16.3.16 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 incremental encoder. Refer to [Table 107](#). 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, repetition counter, trigger output features continue to work as normal. Encoder mode and External clock mode 2 are not compatible and must not be selected together.

In this mode, the counter is modified automatically following the speed and the direction of the incremental encoder and its content, therefore, always represents the encoder's position. The count direction correspond to the rotation direction of the connected sensor. *Table 107* summarizes the possible combinations, assuming TI1 and TI2 don't switch at the same time.

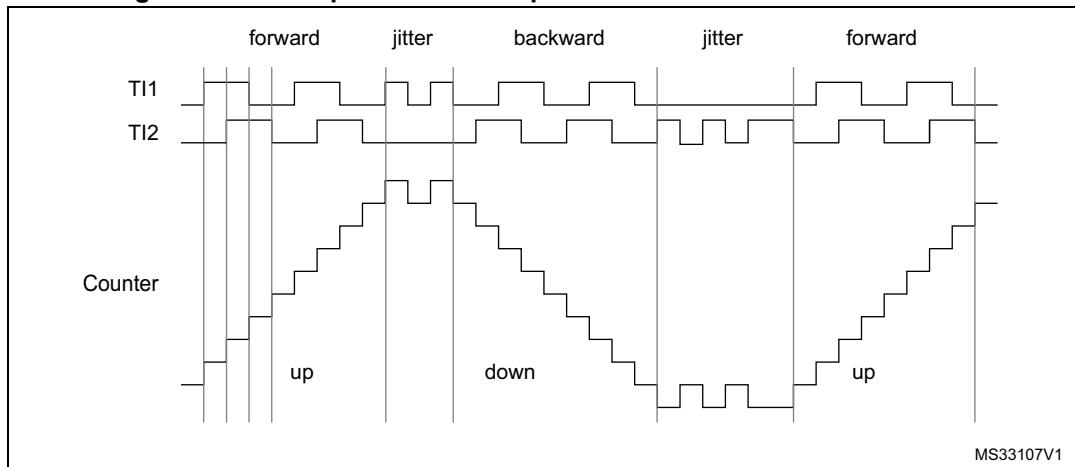
**Table 107. Counting direction versus encoder signals**

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

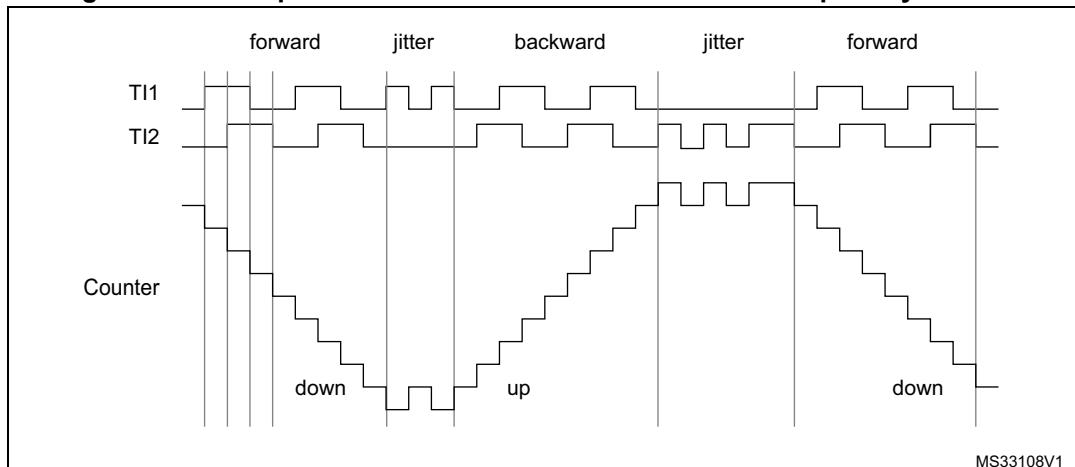
An external incremental encoder can be connected directly to the MCU without external interface logic. However, comparators are normally be used to convert the encoder's differential outputs to digital signals. This greatly increases noise immunity. The third encoder output which indicate the mechanical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

*Figure 150* 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', CC1NP='0', and IC1F = '0000' (TIMx\_CCER register, TI1FP1 non-inverted, TI1FP1=TI1).
- CC2P='0', CC2NP='0', and IC2F = '0000' (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 150. Example of counter operation in encoder interface mode.**

*Figure 151* gives an example of counter behavior when TI1FP1 polarity is inverted (same configuration as above except CC1P='1').

**Figure 151. 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.

### 16.3.17 Timer input XOR function

The TI1S bit in the TIMx\_CR2 register allows the input filter of channel 1 to be connected to the output of a XOR gate, combining the three input pins TIMx\_CH1, TIMx\_CH2 and TIMx\_CH3.

The XOR output can be used with all the timer input functions such as trigger or input capture. An example of this feature used to interface Hall sensors is given in [Section 16.3.18](#) below.

### 16.3.18 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 or TIM5) referred to as “interfacing timer” in [Figure 152](#). The “interfacing timer” captures the 3 timer input pins (TIMx\_CH1, TIMx\_CH2, and TIMx\_CH3) 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 135](#)). 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.

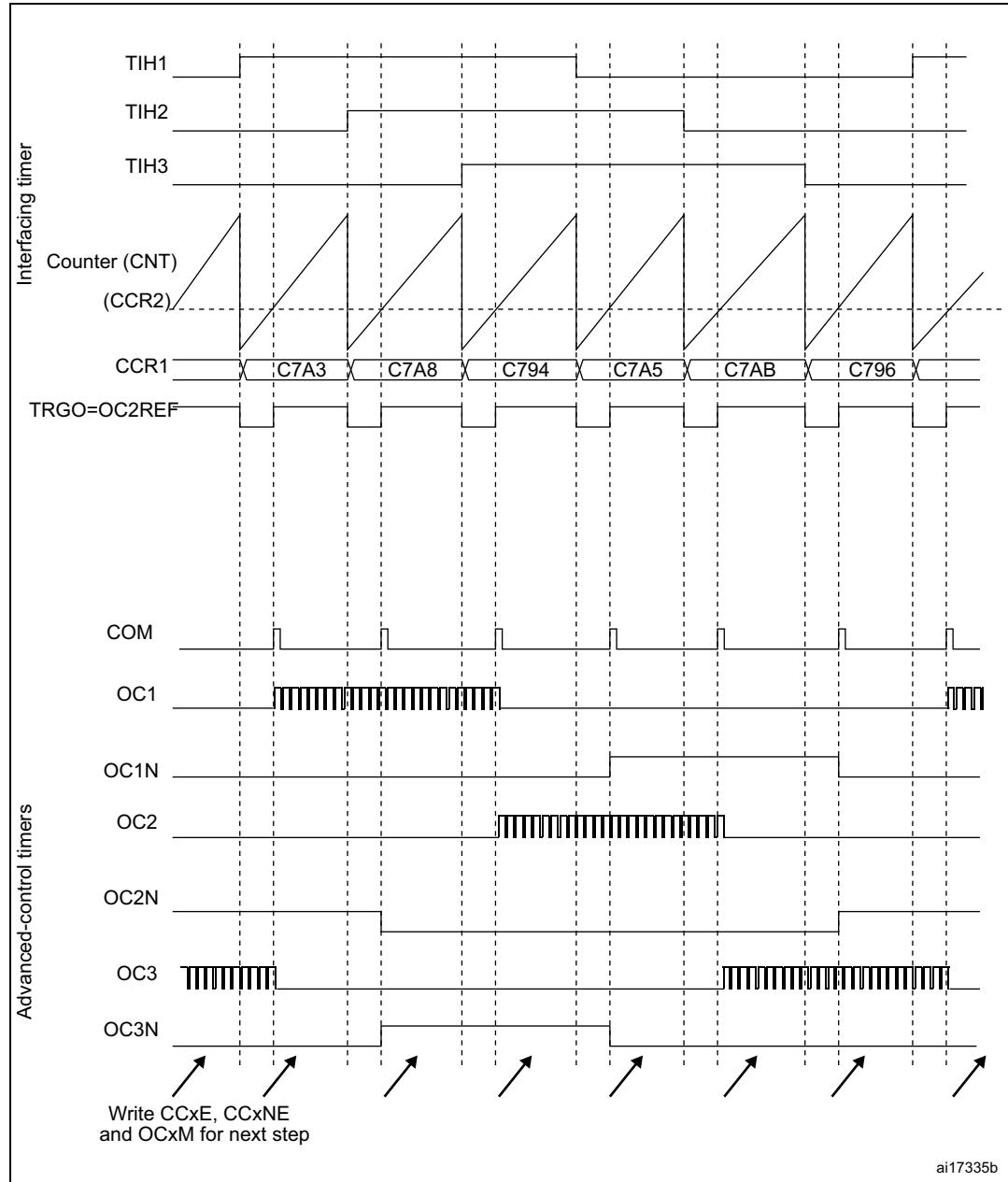
Example: you want to change the PWM configuration of your advanced-control timer TIM1 after a programmed delay each time a change occurs on the Hall inputs connected to one of the TIMx timers.

- 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 channel 1 in capture mode (TRC selected): write the CC1S bits in the TIMx\_CCMR1 register to ‘11’. You can also program the digital filter if needed,
- Program channel 2 in PWM 2 mode with the desired delay: write the OC2M bits to ‘111’ and the CC2S bits to ‘00’ in the TIMx\_CCMR1 register,
- Select OC2REF as trigger output on TRGO: write the MMS bits in the TIMx\_CR2 register to ‘101’,

In the advanced-control timer TIM1, the right ITR input must be selected as trigger input, the timer is programmed to generate PWM signals, the capture/compare control signals are preloaded (CCPC=1 in the TIMx\_CR2 register) and the COM event is controlled by the trigger input (CCUS=1 in the TIMx\_CR2 register). The PWM control bits (CCxE, OCxM) are written after a COM event for the next step (this can be done in an interrupt subroutine generated by the rising edge of OC2REF).

[Figure 152](#) describes this example.

**Figure 152. Example of Hall sensor interface**



### 16.3.19 TIMx and external trigger synchronization

The TIMx timer can be synchronized with an external trigger in several modes: Reset mode, Gated mode and Trigger mode.

#### Slave mode: Reset mode

The counter and its prescaler can be reinitialized in response to an event on a trigger input. Moreover, if the URS bit from the TIMx\_CR1 register is low, an update event UEV is generated. Then all the preloaded registers (TIMx\_ARR, TIMx\_CCRx) are updated.

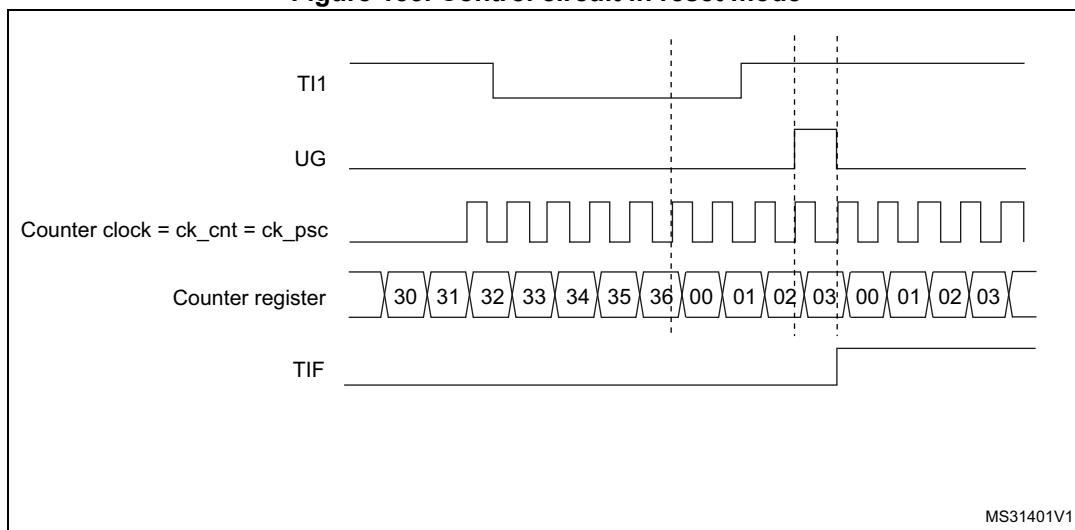
In the following example, the upcounter is cleared in response to a rising edge on TI1 input:

- Configure the channel 1 to detect rising edges on TI1. Configure the input filter duration (in this example, we 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 153. Control circuit in reset mode**



### Slave mode: Gated mode

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

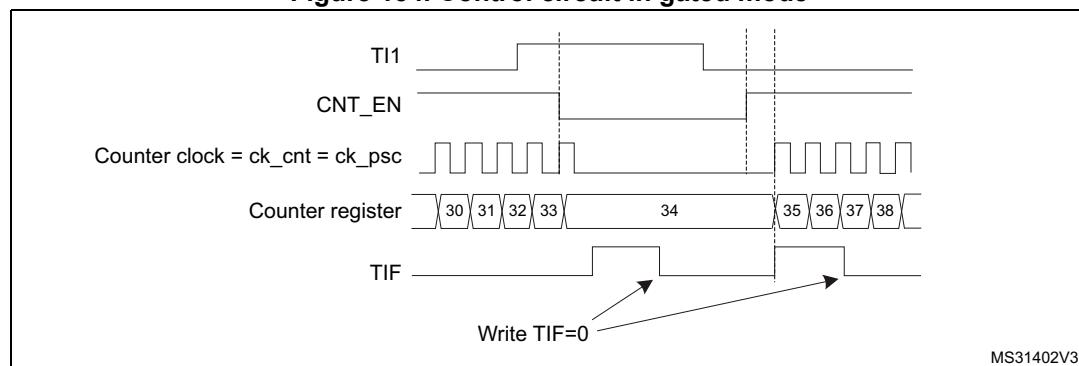
In the following example, the upcounter counts only when TI1 input is low:

- 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 154. Control circuit in gated mode**



### Slave mode: Trigger mode

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

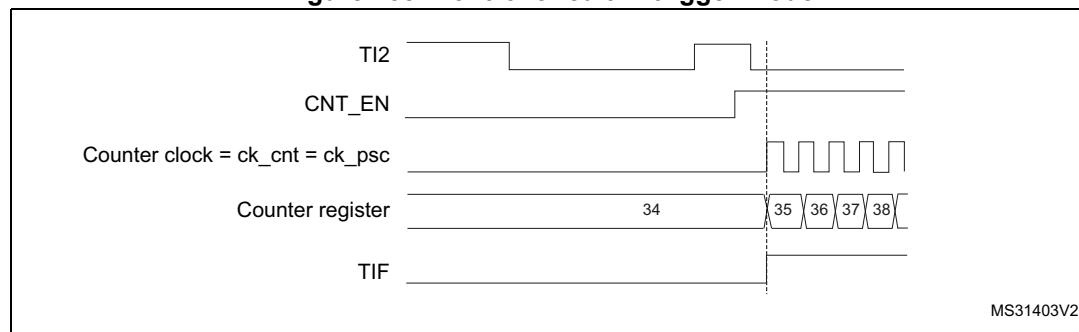
In the following example, the upcounter starts in response to a rising edge on TI2 input:

- 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 155. Control circuit in trigger mode**



### Slave mode: external clock mode 2 + trigger mode

The external clock mode 2 can be used in addition to another slave mode (except external clock mode 1 and encoder mode). In this case, the ETR signal is used as external clock input, and another input can be selected as trigger input (in reset mode, gated mode or trigger mode). It is recommended not to select ETR as TRGI through the TS bits of TIMx\_SMCR register.

In the following example, the upcounter is incremented at each rising edge of the ETR signal as soon as a rising edge of TI1 occurs:

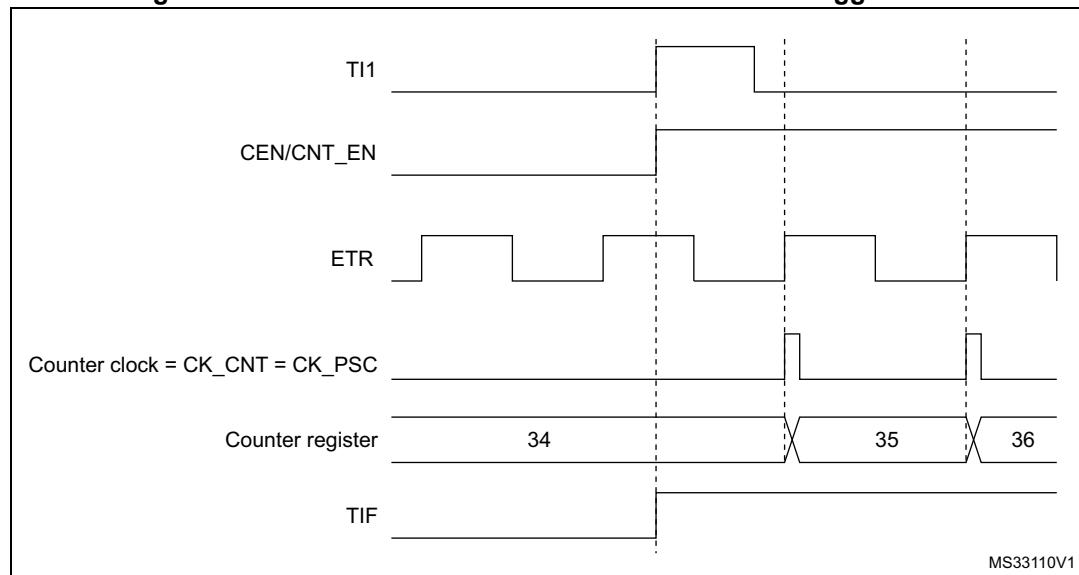
- Configure the external trigger input circuit by programming the TIMx\_SMCR register as follows:
  - ETF = 0000: no filter
  - ETPS = 00: prescaler disabled
  - ETP = 0: detection of rising edges on ETR and ECE=1 to enable the external clock mode 2.

2. Configure the channel 1 as follows, to detect rising edges on TI:
  - IC1F=0000: no filter.
  - The capture prescaler is not used for triggering and does not need to be configured.
  - CC1S=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 156. Control circuit in external clock mode 2 + trigger mode**



### 16.3.20 Timer synchronization

The TIM timers are linked together internally for timer synchronization or chaining. Refer to [Section 17.3.15: Timer synchronization on page 552](#) for details.

### 16.3.21 Debug mode

When the microcontroller enters debug mode (Cortex®-M4 with FPU 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 34.16.2: Debug support for timers, watchdog, bxCAN and I2C](#).

## 16.4 TIM1&TIM8 registers

Refer to [Section 1.1: List of abbreviations for registers](#) for a list of abbreviations used in register descriptions.

The peripheral registers must be written by half-words (16 bits) or words (32 bits). Read accesses can be done by bytes (8 bits), half-word (16 bits) or words (32 bits).

### 16.4.1 TIM1&TIM8 control register 1 (TIMx\_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	CKD[1:0]		ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN

Bits 15:10 Reserved, must be kept at reset value.

Bits 9:8 **CKD[1:0]**: Clock division

This bit-field indicates the division ratio between the timer clock (CK\_INT) frequency and the dead-time and sampling clock ( $t_{DTS}$ ) used by the dead-time generators and the digital filters (ETR, TIx),

00:  $t_{DTS}=t_{CK\_INT}$

01:  $t_{DTS}=2*t_{CK\_INT}$

10:  $t_{DTS}=4*t_{CK\_INT}$

11: Reserved, do not program this value

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

- 0: TIMx\_ARR register is not buffered
- 1: TIMx\_ARR register is buffered

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

00: Edge-aligned mode. The counter counts up or down depending on the direction bit (DIR).

01: Center-aligned mode 1. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx\_CCMRx register) are set only when the counter is counting down.

10: Center-aligned mode 2. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx\_CCMRx register) are set only when the counter is counting up.

11: Center-aligned mode 3. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx\_CCMRx register) are set both when the counter is counting up or down.

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

Bit 4 **DIR**: Direction

- 0: Counter used as upcounter
- 1: Counter used as downcounter

*Note:* This bit is read only when the timer is configured in Center-aligned mode or Encoder mode.

**Bit 3 OPM:** One pulse mode

- 0: Counter is not stopped at update event  
1: Counter stops counting at the next update event (clearing the bit CEN)

**Bit 2 URS:** Update request source

This bit is set and cleared by software to select the UEV event sources.

0: Any of the following events generate an update interrupt or DMA request if enabled.

These events can be:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

1: Only counter overflow/underflow generates an update interrupt or DMA request if enabled.

**Bit 1 UDIS:** Update disable

This bit is set and cleared by software to enable/disable UEV event generation.

0: UEV enabled. The Update (UEV) event is generated by one of the following events:

- Counter overflow/underflow
- Setting the UG bit
- Update generation through the slave mode controller

Buffered registers are then loaded with their preload values.

1: UEV disabled. The Update event is not generated, shadow registers keep their value (ARR, PSC, CCRx). However the counter and the prescaler are reinitialized if the UG bit is set or if a hardware reset is received from the slave mode controller.

**Bit 0 CEN:** Counter enable

0: Counter disabled

1: Counter enabled

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

## 16.4.2 TIM1&TIM8 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.	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

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)

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.*

### 16.4.3 TIM1&TIM8 slave mode control register (TIMx\_SMCR)

Address offset: 0x08

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ETP	ECE	ETPS[1:0]		ETF[3:0]				MSM	TS[2:0]				Res.	SMS[2:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw			rw	rw

Bit 15 **ETP**: External trigger polarity

This bit selects whether ETR or  $\overline{ETR}$  is used for trigger operations

0: ETR is non-inverted, active at high level or rising edge.

1: ETR is inverted, active at low level or falling edge.

Bit 14 **ECE**: External clock enable

This bit enables External clock mode 2.

0: External clock mode 2 disabled

1: External clock mode 2 enabled. The counter is clocked by any active edge on the ETRF signal.

*Note: 1: Setting the ECE bit has the same effect as selecting external clock mode 1 with TRGI connected to ETRF (SMS=111 and TS=111).*

*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: TI1 Edge Detector (TI1F\_ED)
- 101: Filtered Timer Input 1 (TI1FP1)
- 110: Filtered Timer Input 2 (TI2FP2)
- 111: External Trigger input (ETRF)

See [Table 108: TIMx Internal trigger connection](#) for more details on ITRx meaning for each Timer.

*Note: These bits must be changed only when they are not used (e.g. when SMS=000) to avoid wrong edge detections at the transition.*

Bit 3 Reserved, must be kept at reset value.

Bits 2:0 **SMS:** Slave mode selection

When external signals are selected the active edge of the trigger signal (TRGI) is linked to the polarity selected on the external input (see Input Control register and Control Register description).

000: Slave mode disabled - if CEN = '1' then the prescaler is clocked directly by the internal clock.

001: Encoder mode 1 - Counter counts up/down on TI2FP2 edge depending on TI1FP1 level.

010: Encoder mode 2 - Counter counts up/down on TI1FP1 edge depending on TI2FP2 level.

011: Encoder mode 3 - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input.

100: Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter and generates an update of the registers.

101: Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high. The counter stops (but is not reset) as soon as the trigger becomes low. Both start and stop of the counter are controlled.

110: Trigger Mode - The counter starts at a rising edge of the trigger TRGI (but it is not reset). Only the start of the counter is controlled.

111: External Clock Mode 1 - Rising edges of the selected trigger (TRGI) clock the counter.

*Note: The gated mode must not be used if T11F\_ED is selected as the trigger input (TS='100'). Indeed, T11F\_ED outputs 1 pulse for each transition on T11F, whereas the gated mode checks the level of the trigger signal.*

**Table 108. 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

**16.4.4 TIM1&TIM8 DMA/interrupt enable register (TIMx\_DIER)**

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	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

### 16.4.5 TIM1&TIM8 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.	Res.	Res.	CC4OF	CC3OF	CC2OF	CC1OF	Res.	BIF	TIF	COMIF	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

Bit 8 Reserved, must be kept at reset value.

Bit 7 **BIF**: Break interrupt flag

This flag is set by hardware as soon as the break input goes active. It can be cleared by software if the break input is not active.  
0: No break event occurred.  
1: An active level has been detected on the break input.

Bit 6 **TIF**: Trigger interrupt flag

This flag is set by hardware on trigger event (active edge detected on TRGI input when the slave mode controller is enabled in all modes but gated mode. It is 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 16.4.3: TIM1&TIM8 slave mode control register \(TIMx\\_SMCR\)](#)), if URS=0 and UDIS=0 in the TIMx\_CR1 register.

## 16.4.6 TIM1&TIM8 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.	BG	TG	COMG	CC4G	CC3G	CC2G	CC1G	UG							

Bits 15:8 Reserved, must be kept at reset value.

Bit 7 **BG**: Break generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: A break event is generated. MOE bit is cleared and BIF flag is set. Related interrupt or DMA transfer can occur if enabled.

Bit 6 **TG**: Trigger generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: The TIF flag is set in TIMx\_SR register. Related interrupt or DMA transfer can occur if enabled.

Bit 5 **COMG**: Capture/Compare control update generation

This bit can be set by software, it is automatically cleared by hardware

0: No action

1: When CCPC bit is set, it allows to update CCxE, CCxNE and OCxM bits

*Note: This bit acts only on channels having a complementary output.*

Bit 4 **CC4G**: Capture/Compare 4 generation

refer to CC1G description

Bit 3 **CC3G**: Capture/Compare 3 generation

refer to CC1G description

Bit 2 **CC2G**: Capture/Compare 2 generation

refer to CC1G description

Bit 1 **CC1G**: Capture/Compare 1 generation

This bit is set by software in order to generate an event, it is automatically cleared by hardware.

0: No action

1: A capture/compare event is generated on channel 1:

**If channel CC1 is configured as output:**

CC1IF flag is set, Corresponding interrupt or DMA request is sent if enabled.

**If channel CC1 is configured as input:**

The current value of the counter is captured in TIMx\_CCR1 register. The CC1IF flag is set, the corresponding interrupt or DMA request is sent if enabled. The CC1OF flag is set if the CC1IF flag was already high.

Bit 0 **UG**: Update generation

This bit can be set by software, it is automatically cleared by hardware.

0: No action

1: Reinitialize the counter and generates an update of the registers. Note that the prescaler counter is cleared too (anyway the prescaler ratio is not affected). The counter is cleared if the center-aligned mode is selected or if DIR=0 (upcounting), else it takes the auto-reload value (TIMx\_ARR) if DIR=1 (downcounting).

### 16.4.7 TIM1&TIM8 capture/compare mode register 1 (TIMx\_CCMR1)

Address offset: 0x18

Reset value: 0x0000

The channels can be used in input (capture mode) or in output (compare mode). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OCxx describes its function when the channel is configured in output, ICxx describes its function when the channel is configured in input. So you must take care that the same bit can have a different meaning for the input stage and for the output stage.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
OC2 CE	OC2M[2:0]			OC2 PE	OC2 FE	CC2S[1:0]	OC1 CE	OC1M[2:0]			OC1 PE	OC1 FE	CC1S[1:0]			
IC2F[3:0]			IC2PSC[1:0]				IC1F[3:0]			IC1PSC[1:0]						
RW	RW	RW	RW	RW	RW		RW	RW	RW	RW	RW	RW	RW	RW	RW	

#### Output compare mode:

Bit 15 **OC2CE**: Output Compare 2 clear enable

Bits 14:12 **OC2M[2:0]**: Output Compare 2 mode

Bit 11 **OC2PE**: Output Compare 2 preload enable

Bit 10 **OC2FE**: Output Compare 2 fast enable

Bits 9:8 **CC2S[1:0]**: Capture/Compare 2 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output

01: CC2 channel is configured as input, IC2 is mapped on TI2

10: CC2 channel is configured as input, IC2 is mapped on TI1

11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode is working only if an internal trigger input is selected through the TS bit (TIMx\_SMCR register)

Note: CC2S bits are writable only when the channel is OFF (CC2E = '0' in TIMx\_CCER).

Bit 7 **OC1CE**: Output Compare 1 clear enable

OC1CE: Output Compare 1 Clear Enable

0: OC1Ref is not affected by the ETRF Input

1: OC1Ref is cleared as soon as a High level is detected on ETRF input

#### Bits 6:4 **OC1M**: Output Compare 1 mode

These bits define the behavior of the output reference signal OC1REF from which OC1 and OC1N are derived. OC1REF is active high whereas OC1 and OC1N active level depends on CC1P and CC1NP bits.

000: Frozen - The comparison between the output compare register TIMx\_CCR1 and the counter TIMx\_CNT has no effect on the outputs.(this mode is used to generate a timing base).

001: Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

011: Toggle - OC1REF toggles when TIMx\_CNT=TIMx\_CCR1.

100: Force inactive level - OC1REF is forced low.

101: Force active level - OC1REF is forced high.

110: PWM mode 1 - In upcounting, channel 1 is active as long as TIMx\_CNT<TIMx\_CCR1 else inactive. In downcounting, channel 1 is inactive (OC1REF='0') as long as TIMx\_CNT>TIMx\_CCR1 else active (OC1REF='1').

111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIMx\_CNT<TIMx\_CCR1 else active. In downcounting, channel 1 is active as long as TIMx\_CNT>TIMx\_CCR1 else inactive.

*Note: 1: These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx\_BDTR register) and CC1S='00' (the channel is configured in output).*

*2: In PWM mode 1 or 2, the OCREF level changes only when the result of the comparison changes or when the output compare mode switches from “frozen” mode to “PWM” mode.*

*3: On channels having a complementary output, this bit field is preloaded. If the CCPC bit is set in the TIMx\_CR2 register then the OC1M active bits take the new value from the preloaded bits only when a COM event is generated.*

#### Bit 3 **OC1PE**: Output Compare 1 preload enable

0: Preload register on TIMx\_CCR1 disabled. TIMx\_CCR1 can be written at anytime, the new value is taken in account immediately.

1: Preload register on TIMx\_CCR1 enabled. Read/Write operations access the preload register. TIMx\_CCR1 preload value is loaded in the active register at each update event.

*Note: 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. OC1FE acts only if the channel is configured in PWM1 or PWM2 mode.

Bits 1:0 **CC1S**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output

01: CC1 channel is configured as input, IC1 is mapped on TI1

10: CC1 channel is configured as input, IC1 is mapped on TI2

11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC1S bits are writable only when the channel is OFF (CC1E = '0' in TIMx\_CCER).*

### Input capture mode

Bits 15:12 **IC2F**: Input capture 2 filter

Bits 11:10 **IC2PSC[1:0]**: Input capture 2 prescaler

Bits 9:8 **CC2S**: Capture/Compare 2 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output

01: CC2 channel is configured as input, IC2 is mapped on TI2

10: CC2 channel is configured as input, IC2 is mapped on TI1

11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC2S bits are writable only when the channel is OFF (CC2E = '0' in TIMx\_CCER).*

Bits 7:4 **IC1F[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).*

**16.4.8 TIM1&TIM8 capture/compare mode register 2 (TIMx\_CCMR2)**

Address offset: 0x1C

Reset value: 0x0000

Refer to the above CCMR1 register description.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC4CE	OC4M[2:0]			OC4PE	OC4FE	CC4S[1:0]	OC3CE.	OC3M[2:0]			OC3PE	OC3FE	CC3S[1:0]		
IC4F[3:0]			IC4PSC[1:0]				IC3F[3:0]			IC3PSC[1:0]					
rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw

**Output compare mode**

Bit 15 **OC4CE:** Output compare 4 clear enable

Bits 14:12 **OC4M:** Output compare 4 mode

Bit 11 **OC4PE:** Output compare 4 preload enable

Bit 10 **OC4FE:** Output compare 4 fast enable

Bits 9:8 **CC4S:** Capture/Compare 4 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC4 channel is configured as output

01: CC4 channel is configured as input, IC4 is mapped on TI4

10: CC4 channel is configured as input, IC4 is mapped on TI3

11: CC4 channel is configured as input, IC4 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC4S bits are writable only when the channel is OFF (CC4E = '0' in TIMx\_CCER).*

Bit 7 **OC3CE:** Output compare 3 clear enable

Bits 6:4 **OC3M:** Output compare 3 mode

Bit 3 **OC3PE**: Output compare 3 preload enable

Bit 2 **OC3FE**: Output compare 3 fast enable

Bits 1:0 **CC3S**: Capture/Compare 3 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC3 channel is configured as output

01: CC3 channel is configured as input, IC3 is mapped on TI3

10: CC3 channel is configured as input, IC3 is mapped on TI4

11: CC3 channel is configured as input, IC3 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC3S bits are writable only when the channel is OFF (CC3E = '0' in TIMx\_CCER).*

### Input capture mode

Bits 15:12 **IC4F**: Input capture 4 filter

Bits 11:10 **IC4PSC**: Input capture 4 prescaler

Bits 9:8 **CC4S**: Capture/Compare 4 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC4 channel is configured as output

01: CC4 channel is configured as input, IC4 is mapped on TI4

10: CC4 channel is configured as input, IC4 is mapped on TI3

11: CC4 channel is configured as input, IC4 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC4S bits are writable only when the channel is OFF (CC4E = '0' in TIMx\_CCER).*

Bits 7:4 **IC3F**: Input capture 3 filter

Bits 3:2 **IC3PSC**: Input capture 3 prescaler

Bits 1:0 **CC3S**: Capture/compare 3 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC3 channel is configured as output

01: CC3 channel is configured as input, IC3 is mapped on TI3

10: CC3 channel is configured as input, IC3 is mapped on TI4

11: CC3 channel is configured as input, IC3 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC3S bits are writable only when the channel is OFF (CC3E = '0' in TIMx\_CCER).*

## 16.4.9 TIM1&TIM8 capture/compare enable register (TIMx\_CCER)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E

Bits 15:14 Reserved, must be kept at reset value.

Bit 13 **CC4P**: Capture/Compare 4 output polarity

refer to CC1P description

- Bit 12 **CC4E**: Capture/Compare 4 output enable  
refer to CC1E description
- Bit 11 **CC3NP**: Capture/Compare 3 complementary output polarity  
refer to CC1NP description
- Bit 10 **CC3NE**: Capture/Compare 3 complementary output enable  
refer to CC1NE description
- Bit 9 **CC3P**: Capture/Compare 3 output polarity  
refer to CC1P description
- Bit 8 **CC3E**: Capture/Compare 3 output enable  
refer to CC1E description
- Bit 7 **CC2NP**: Capture/Compare 2 complementary output polarity  
refer to CC1NP description
- Bit 6 **CC2NE**: Capture/Compare 2 complementary output enable  
refer to CC1NE description
- Bit 5 **CC2P**: Capture/Compare 2 output polarity  
refer to CC1P description
- Bit 4 **CC2E**: Capture/Compare 2 output enable  
refer to CC1E description
- Bit 3 **CC1NP**: Capture/Compare 1 complementary output polarity  
**CC1 channel 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: 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.*  
*Note: This bit is not writable as soon as LOCK level 2 or 3 has been programmed (LOCK bits in TIMx\_BDTR register) and CC1S="00" (the channel is configured in output).*
- Bit 2 **CC1NE**: Capture/Compare 1 complementary output enable  
0: Off - OC1N is not active. OC1N level is then function of MOE, OSS1, OSSR, OIS1, OIS1N and CC1E bits.  
1: On - OC1N signal is output on the corresponding output pin depending on MOE, OSS1, OSSR, OIS1, OIS1N and CC1E bits.  
*Note: On channels having a complementary output, this bit is preloaded. If the CCPC bit is set in the TIMx\_CR2 register then the CC1NE active bit takes the new value from the preloaded 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: 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.*

*Note: This bit is not writable as soon as LOCK level 2 or 3 has been programmed (LOCK bits in TIMx\_BDTR register).*

Bit 0 **CC1E**: Capture/Compare 1 output enable

**CC1 channel configured as output:**

0: Off - OC1 is not active. OC1 level is then function of MOE, OSS1, OSSR, OIS1, OIS1N and CC1NE bits.

1: On - OC1 signal is output on the corresponding output pin depending on MOE, OSS1, OSSR, OIS1, OIS1N and CC1NE bits.

**CC1 channel configured as input:**

This bit determines if a capture of the counter value can actually be done into the input capture/compare register 1 (TIMx\_CCR1) or not.

0: Capture disabled.

1: Capture enabled.

*Note: On channels having a complementary output, this bit is preloaded. If the CCPC bit is set in the TIMx\_CR2 register then the CC1E active bit takes the new value from the preloaded bit only when a Commutation event is generated.*

**Table 109. Output control bits for complementary OCx and OCxN channels with break feature**

Control bits					Output states <sup>(1)</sup>	
MOE bit	OSSI bit	OSSR bit	CCxE bit	CCxNE bit	OCx output state	OCxN output state
1	X	0	0	0	Output Disabled (not driven by the timer) OCx=0, OCx_EN=0	Output Disabled (not driven by the timer) OCxN=0, OCxN_EN=0
				1	Output Disabled (not driven by the timer) OCx=0, OCx_EN=0	OCxREF + Polarity OCxN=OCxREF xor CCxNP, OCxN_EN=1
			1	0	OCxREF + Polarity OCx=OCxREF xor CCxP, OCx_EN=1	Output Disabled (not driven by the timer) OCxN=0, OCxN_EN=0
				1	OCREF + Polarity + dead-time OCx_EN=1	Complementary to OCREF (not OCREF) + Polarity + dead-time OCxN_EN=1
		1	0	0	Output Disabled (not driven by the timer) OCx=CCxP, OCx_EN=0	Output Disabled (not driven by the timer) OCxN=CCxNP, OCxN_EN=0
				1	Off-State (output enabled with inactive state) OCx=CCxP, OCx_EN=1	OCxREF + Polarity OCxN=OCxREF xor CCxNP, OCxN_EN=1
			1	0	OCxREF + Polarity OCx=OCxREF xor CCxP, OCx_EN=1	Off-State (output enabled with inactive state) OCxN=CCxNP, OCxN_EN=1
				1	OCREF + Polarity + dead-time OCx_EN=1	Complementary to OCREF (not OCREF) + Polarity + dead-time OCxN_EN=1
0	X	0	0	0	Output Disabled (not driven by the timer) OCx=CCxP, OCx_EN=0	Output Disabled (not driven by the timer) OCxN=CCxNP, OCxN_EN=0
				1	Output Disabled (not driven by the timer)	
			1	0	Asynchronously: OCx=CCxP, OCx_EN=0, OCxN=CCxNP, OCxN_EN=0	
				1	Then if the clock is present: OCx=OISx and OCxN=OISxN after a dead-time, assuming that OISx and OISxN do not correspond to OCX and OCxN both in active state.	
		1	0	0	Output Disabled (not driven by the timer) OCx=CCxP, OCx_EN=0	Output Disabled (not driven by the timer) OCxN=CCxNP, OCxN_EN=0
				1	Off-State (output enabled with inactive state)	
			1	0	Asynchronously: OCx=CCxP, OCx_EN=1, OCxN=CCxNP, OCxN_EN=1	
				1	Then if the clock is present: OCx=OISx and OCxN=OISxN after a dead-time, assuming that OISx and OISxN do not correspond to OCX and OCxN both in active state	

- When both outputs of a channel are not used (CCxE = CCxNE = 0), the OISx, OISxN, CCxP and CCxNP bits must be kept cleared.

**Note:** The state of the external I/O pins connected to the complementary OC<sub>x</sub> and OC<sub>xN</sub> channels depends on the OC<sub>x</sub> and OC<sub>xN</sub> channel state and the GPIO registers.

#### 16.4.10 TIM1&TIM8 counter (TIMx\_CNT)

Address offset: 0x24

Reset value: 0x0000

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

Bits 15:0 **CNT[15:0]**: Counter value

#### 16.4.11 TIM1&TIM8 prescaler (TIMx\_PSC)

Address offset: 0x28

Reset value: 0x0000

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

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency (CK\_CNT) is equal to  $f_{CK\_PSC} / (PSC[15:0] + 1)$ .

PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx\_EGR register or through trigger controller when configured in “reset mode”).

#### 16.4.12 TIM1&TIM8 auto-reload register (TIMx\_ARR)

Address offset: 0x2C

Reset value: 0x0000

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

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 16.3.1: Time-base unit](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

### 16.4.13 TIM1&TIM8 repetition counter register (TIMx\_RCR)

Address offset: 0x30

Reset value: 0x0000

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

Bits 15:8 Reserved, must be kept at reset value.

Bits 7:0 **REP[7:0]**: Repetition counter value

These bits allow the user to set-up the update rate of the compare registers (i.e. periodic transfers from preload to active registers) when preload registers are enable, as well as the update interrupt generation rate, if this interrupt is enable.

Each time the REP\_CNT related downcounter reaches zero, an update event is generated and it restarts counting from REP value. As REP\_CNT is reloaded with REP value only at the repetition update event U\_RC, any write to the TIMx\_RCR register is not taken in account until the next repetition update event.

It means in PWM mode (REP+1) corresponds to:

- the number of PWM periods in edge-aligned mode
- the number of half PWM period in center-aligned mode.

### 16.4.14 TIM1&TIM8 capture/compare register 1 (TIMx\_CCR1)

Address offset: 0x34

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR1[15:0]**: Capture/Compare 1 value

**If channel CC1 is configured as output:**

CCR1 is the value to be loaded in the actual capture/compare 1 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR1 register (bit OC1PE). Else the preload value is copied in the active capture/compare 1 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signaled on OC1 output.

**If channel CC1 is configured as input:**

CCR1 is the counter value transferred by the last input capture 1 event (IC1).

### 16.4.15 TIM1&TIM8 capture/compare register 2 (TIMx\_CCR2)

Address offset: 0x38

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR2[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR2[15:0]**: Capture/Compare 2 value

**If channel CC2 is configured as output:**

CCR2 is the value to be loaded in the actual capture/compare 2 register (preload value). It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR2 register (bit OC2PE). Else the preload value is copied in the active capture/compare 2 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signalled on OC2 output.

**If channel CC2 is configured as input:**

CCR2 is the counter value transferred by the last input capture 2 event (IC2).

### 16.4.16 TIM1&TIM8 capture/compare register 3 (TIMx\_CCR3)

Address offset: 0x3C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR3[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR3[15:0]**: Capture/Compare value

**If channel CC3 is configured as output:**

CCR3 is the value to be loaded in the actual capture/compare 3 register (preload value). It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR3 register (bit OC3PE). Else the preload value is copied in the active capture/compare 3 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signalled on OC3 output.

**If channel CC3 is configured as input:**

CCR3 is the counter value transferred by the last input capture 3 event (IC3).

### 16.4.17 TIM1&TIM8 capture/compare register 4 (TIMx\_CCR4)

Address offset: 0x40

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR4[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR4[15:0]**: Capture/Compare value

**If channel CC4 is configured as output:**

CCR4 is the value to be loaded in the actual capture/compare 4 register (preload value). It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR4 register (bit OC4PE). Else the preload value is copied in the active capture/compare 4 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signalled on OC4 output.

**If channel CC4 is configured as input:**

CCR4 is the counter value transferred by the last input capture 4 event (IC4).

### 16.4.18 TIM1&TIM8 break and dead-time register (TIMx\_BDTR)

Address offset: 0x44

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK[1:0]		DTG[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

**Note:** As the bits AOE, BKP, BKE, OSSR and DTG[7:0] can be write-locked depending on the LOCK configuration, it can be necessary to configure all of them during the first write access to the TIMx\_BDTR register.

**Bit 15 MOE: Main output enable**

This bit is cleared asynchronously by hardware as soon as the break input is active. It is set by software or automatically depending on the AOE bit. It is acting only on the channels which are configured in output.

0: OC and OCN outputs are disabled or forced to idle state.

1: OC and OCN outputs are enabled if their respective enable bits are set (CCxE, CCxNE in TIMx\_CCER register).

See OC/OCN enable description for more details ([Section 16.4.9: TIM1&TIM8 capture/compare enable register \(TIMx\\_CCER\) on page 507](#)).

**Bit 14 AOE: Automatic output enable**

0: MOE can be set only by software

1: MOE can be set by software or automatically at the next update event (if the break input is not active)

**Note:** This bit can not be modified as long as LOCK level 1 has been programmed (LOCK bits in TIMx\_BDTR register).

Bit 13 **BKP**: Break polarity

- 0: Break input BRK is active low
- 1: Break input BRK is active high

*Note: This bit can not be modified as long as LOCK level 1 has been programmed (LOCK bits in TIMx\_BDTR register).*

*Note: Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.*

Bit 12 **BKE**: Break enable

- 0: Break inputs (BRK and CSS clock failure event) disabled
- 1: Break inputs (BRK and CSS clock failure event) enabled

*Note: This bit cannot be modified when LOCK level 1 has been programmed (LOCK bits in TIMx\_BDTR register).*

*Note: Any write operation to this bit takes a delay of 1 APB clock cycle to become effective.*

Bit 11 **OSSR**: Off-state selection for Run mode

This bit is used when MOE=1 on channels having a complementary output which are configured as outputs. OSSR is not implemented if no complementary output is implemented in the timer.

See OC/OCN enable description for more details ([Section 16.4.9: TIM1&TIM8 capture/compare enable register \(TIMx\\_CCER\) on page 507](#)).

- 0: When inactive, OC/OCN outputs are disabled (OC/OCN enable output signal=0).
- 1: When inactive, OC/OCN outputs are enabled with their inactive level as soon as CCxE=1 or CCxNE=1. Then, OC/OCN enable output signal=1

*Note: This bit can not be modified as soon as the LOCK level 2 has been programmed (LOCK bits in TIMx\_BDTR register).*

Bit 10 **OSSI**: Off-state selection for Idle mode

This bit is used when MOE=0 on channels configured as outputs.

See OC/OCN enable description for more details ([Section 16.4.9: TIM1&TIM8 capture/compare enable register \(TIMx\\_CCER\) on page 507](#)).

- 0: When inactive, OC/OCN outputs are disabled (OC/OCN enable output signal=0).
- 1: When inactive, OC/OCN outputs are forced first with their idle level as soon as CCxE=1 or CCxNE=1. OC/OCN enable output signal=1

*Note: This bit can not be modified as soon as the LOCK level 2 has been programmed (LOCK bits in TIMx\_BDTR register).*

Bits 9:8 **LOCK[1:0]**: Lock configuration

These bits offer a write protection against software errors.

00: LOCK OFF - No bit is write protected.

01: LOCK Level 1 = DTG bits in TIMx\_BDTR register, OISx and OISxN bits in TIMx\_CR2 register and BKE/BKP/AOE bits in TIMx\_BDTR register can no longer be written.

10: LOCK Level 2 = LOCK Level 1 + CC Polarity bits (CCxP/CCxNP bits in TIMx\_CCER register, as long as the related channel is configured in output through the CCxS bits) as well as OSSR and OSSI bits can no longer be written.

11: LOCK Level 3 = LOCK Level 2 + CC Control bits (OCxM and OCxPE bits in TIMx\_CCMRx registers, as long as the related channel is configured in output through the CCxS bits) can no longer be written.

*Note: The LOCK bits can be written only once after the reset. Once the TIMx\_BDTR register has been written, their content is frozen until the next reset.*

Bits 7:0 **DTG[7:0]**: Dead-time generator setup

This bit-field defines the duration of the dead-time inserted between the complementary outputs. DT correspond to this duration.

$DTG[7:5]=0xx \Rightarrow DT=DTG[7:0]x t_{dtg}$  with  $t_{dtg}=t_{DTS}$ .

$DTG[7:5]=10x \Rightarrow DT=(64+DTG[5:0])x t_{dtg}$  with  $T_{dtg}=2xt_{DTS}$ .

$DTG[7:5]=110 \Rightarrow DT=(32+DTG[4:0])x t_{dtg}$  with  $T_{dtg}=8xt_{DTS}$ .

$DTG[7:5]=111 \Rightarrow DT=(32+DTG[4:0])x t_{dtg}$  with  $T_{dtg}=16xt_{DTS}$ .

Example if  $T_{DTS}=125\text{ns}$  (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).*

#### 16.4.19 TIM1&TIM8 DMA control register (TIMx\_DCR)

Address offset: 0x48

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.			DBL[4:0]			Res.	Res.	Res.		DBA[4:0]			

Bits 15:13 Reserved, must be kept at reset value.

Bits 12:8 **DBL[4:0]**: DMA burst length

This 5-bit vector defines the number of DMA transfers (the timer detects a burst transfer when a read or a write access to the TIMx\_DMAR register address is performed).

the TIMx\_DMAR address)

00000: 1 transfer

00001: 2 transfers

00010: 3 transfers

...

10001: 18 transfers

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

Bits 4:0 **DBA[4:0]**: DMA base address

This 5-bits vector defines the base-address for DMA transfers (when read/write access are done through the TIMx\_DMAR address). DBA is defined as an offset starting from the address of the TIMx\_CR1 register.

Example:

00000: TIMx\_CR1,

00001: TIMx\_CR2,

00010: TIMx\_SMCR,

...

**Example:** Let us consider the following transfer: DBL = 7 transfers and DBA = TIMx\_CR1. In this case the transfer is done to/from 7 registers starting from the TIMx\_CR1 address.

### 16.4.20 TIM1&TIM8 DMA address for full transfer (TIMx\_DMAR)

Address offset: 0x4C

Reset value: 0x0000

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

Bits 15:0 **DMAB[15:0]**: DMA register for burst accesses

A read or write operation to the DMAR register accesses the register located at the address  
 $(\text{TIMx\_CR1 address}) + (\text{DBA} + \text{DMA index}) \times 4$

where TIMx\_CR1 address is the address of the control register 1, DBA is the DMA base address configured in TIMx\_DCR register, DMA index is automatically controlled by the DMA transfer, and ranges from 0 to DBL (DBL configured in TIMx\_DCR).

#### Example of how to use the DMA burst feature

In this example the timer DMA burst feature is used to update the contents of the CCRx registers ( $x = 2, 3, 4$ ) with the DMA transferring half words into the CCRx registers.

This is done in the following steps:

1. Configure the corresponding DMA channel as follows:
  - DMA channel peripheral address is the DMAR register address
  - DMA channel memory address is the address of the buffer in the RAM containing the data to be transferred by DMA into CCRx registers.
  - Number of data to transfer = 3 (See note below).
  - Circular mode disabled.
2. Configure the DCR register by configuring the DBA and DBL bit fields as follows:  
 $\text{DBL} = 3$  transfers,  $\text{DBA} = 0xE$ .
3. Enable the TIMx update DMA request (set the UDE bit in the DIER register).
4. Enable TIMx
5. Enable the DMA channel

Note:

*This example is for the case where every CCRx register to be updated once. If every CCRx register is to be updated twice for example, the number of data to transfer should be 6. Let's take the example of a buffer in the RAM containing data1, data2, data3, data4, data5 and data6. The data is transferred to the CCRx registers as follows: on the first update DMA request, data1 is transferred to CCR2, data2 is transferred to CCR3, data3 is transferred to CCR4 and on the second update DMA request, data4 is transferred to CCR2, data5 is transferred to CCR3 and data6 is transferred to CCR4.*

### 16.4.21 TIM1&TIM8 register map

TIM1&TIM8 registers are mapped as 16-bit addressable registers as described in the table below:

**Table 110. TIM1&TIM8 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
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																																
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.	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.		
0x1C	TIMx_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.		
	Reset value																																
	TIMx_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.		
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	Res.	Res.	Res.	Res.	Res.	Res.	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																																
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																																
0x30	TIMx_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.	Res.		
	Reset value																																

Table 110. TIM1&amp;TIM8 register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x34	<b>TIMx_CCR1</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																															
0x38	<b>TIMx_CCR2</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																															
0x3C	<b>TIMx_CCR3</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																															
0x40	<b>TIMx_CCR4</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																															
0x44	<b>TIMx_BDTR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																															
0x48	<b>TIMx_DCR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																															
0x4C	<b>TIMx_DMAR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																															

Refer to [Section 2.2.2 on page 56](#) for the register boundary addresses.

## 17 General-purpose timers (TIM2 to TIM5)

### 17.1 TIM2 to 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.

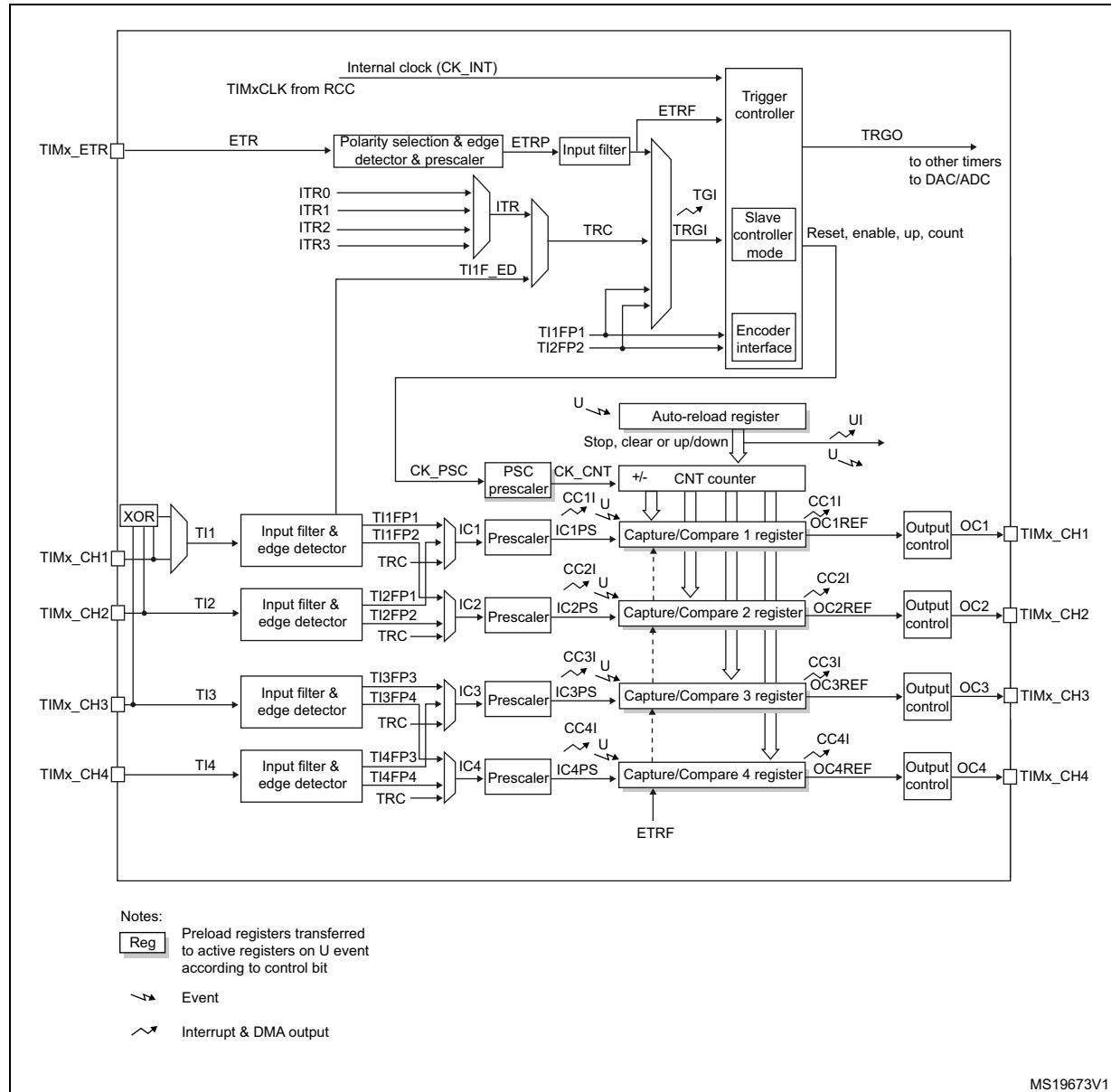
The timers are completely independent, and do not share any resources. They can be synchronized together as described in [Section 17.3.15](#).

### 17.2 TIM2 to TIM5 main features

General-purpose TIMx timer features include:

- 16-bit (TIM3 and 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 65536.
- 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 157. General-purpose timer block diagram



## 17.3 TIM2 to TIM5 functional description

### 17.3.1 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. 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).

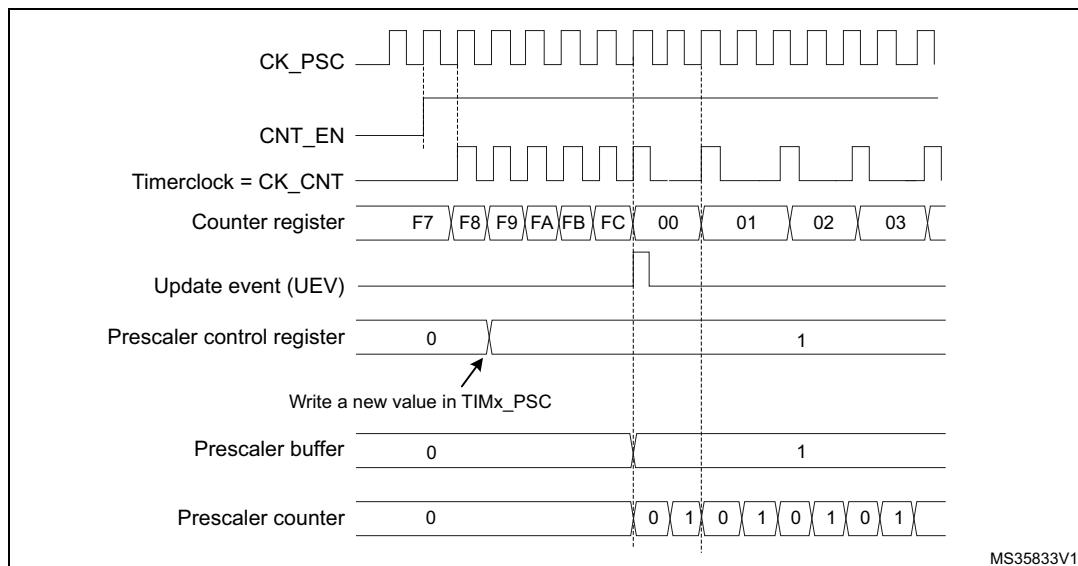
Note that the actual counter enable signal CNT\_EN is set 1 clock cycle after CEN.

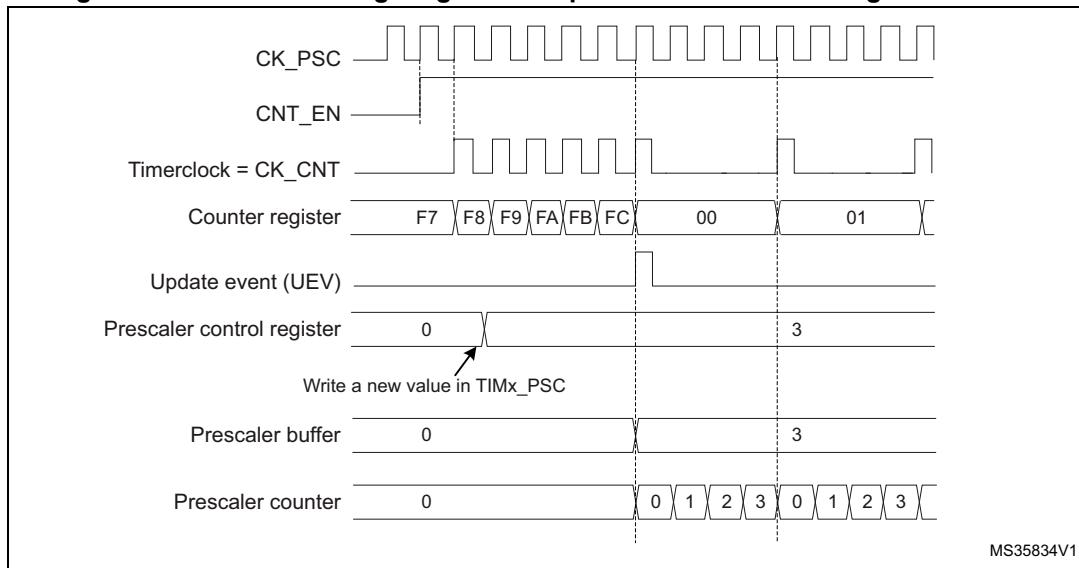
## Prescaler description

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 158](#) and [Figure 159](#) give some examples of the counter behavior when the prescaler ratio is changed on the fly:

**Figure 158.** Counter timing diagram with prescaler division change from 1 to 2



**Figure 159. Counter timing diagram with prescaler division change from 1 to 4**

### 17.3.2 Counter modes

#### Upcounting mode

In upcounting mode, the counter counts from 0 to the auto-reload value (content of the TIMx\_ARR register), then restarts from 0 and generates a counter overflow event.

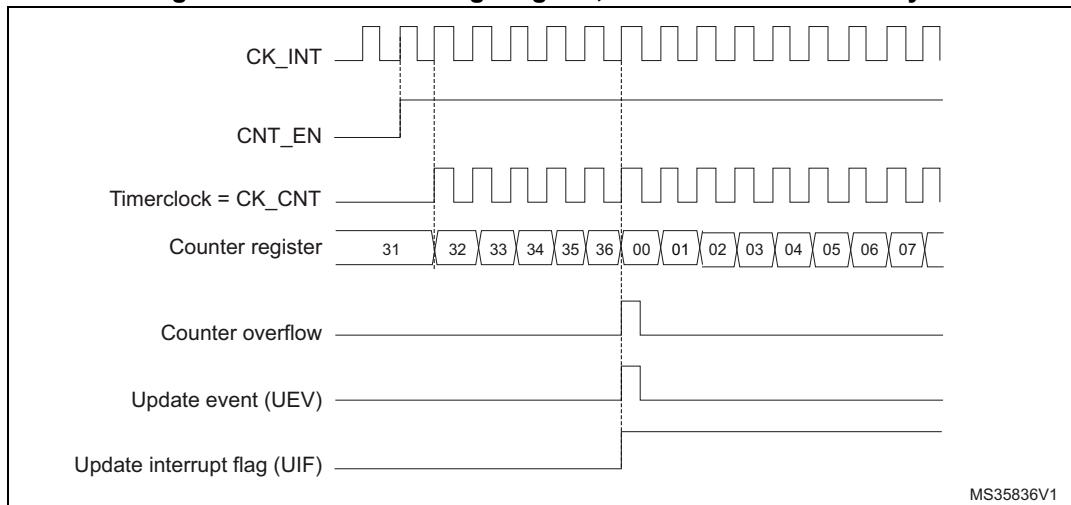
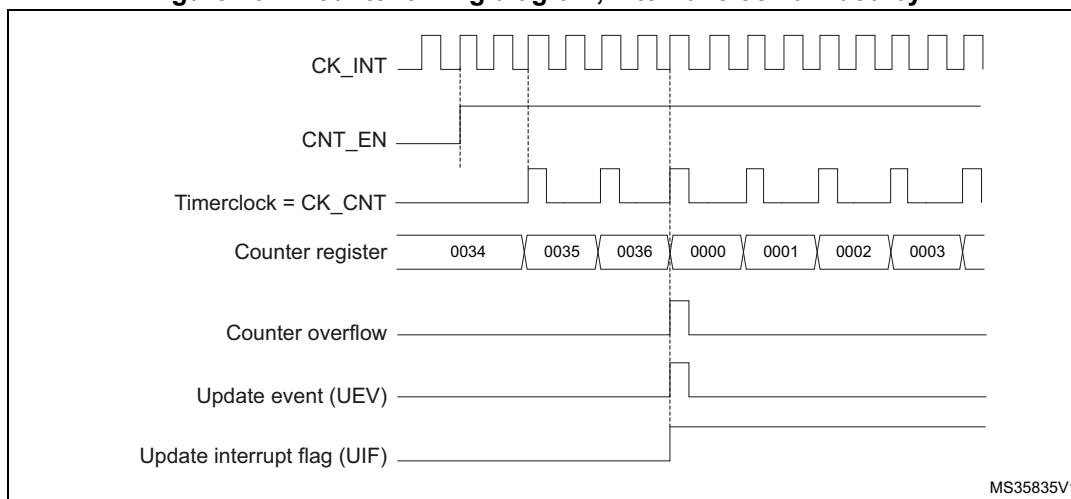
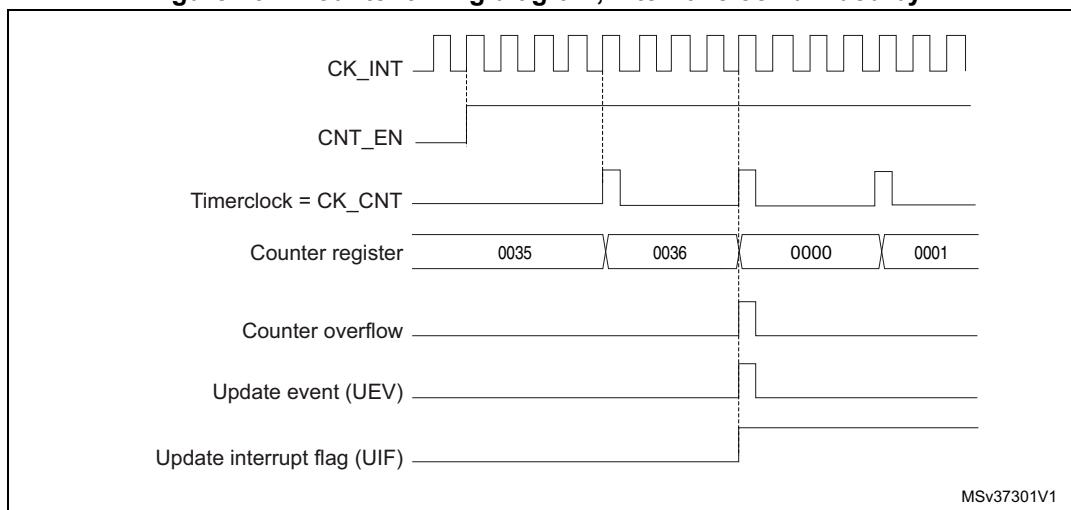
An Update event can be generated at each counter overflow or by setting the UG bit in the TIMx\_EGR register (by software or by using the slave mode controller).

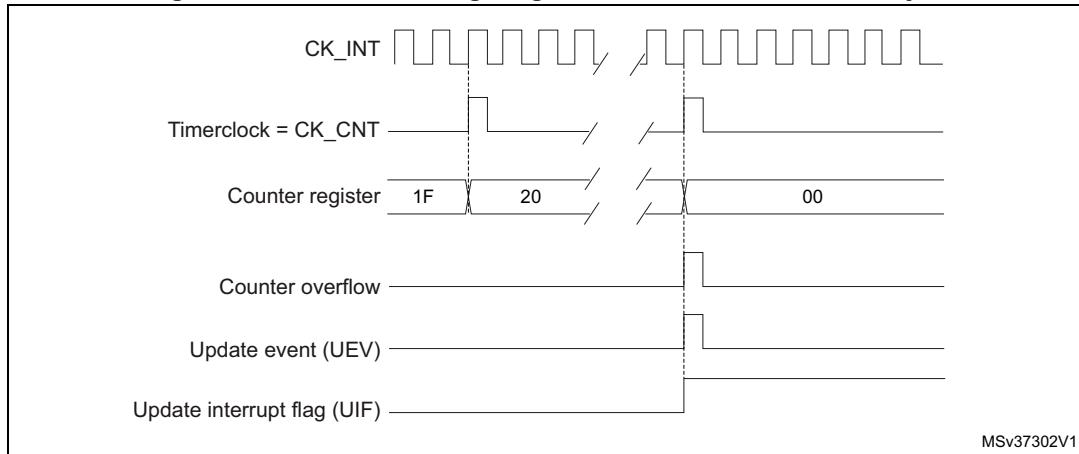
The UEV event can be disabled by software by setting the UDIS bit in TIMx\_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until the UDIS bit has been written to 0. However, the counter restarts from 0, as well 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):

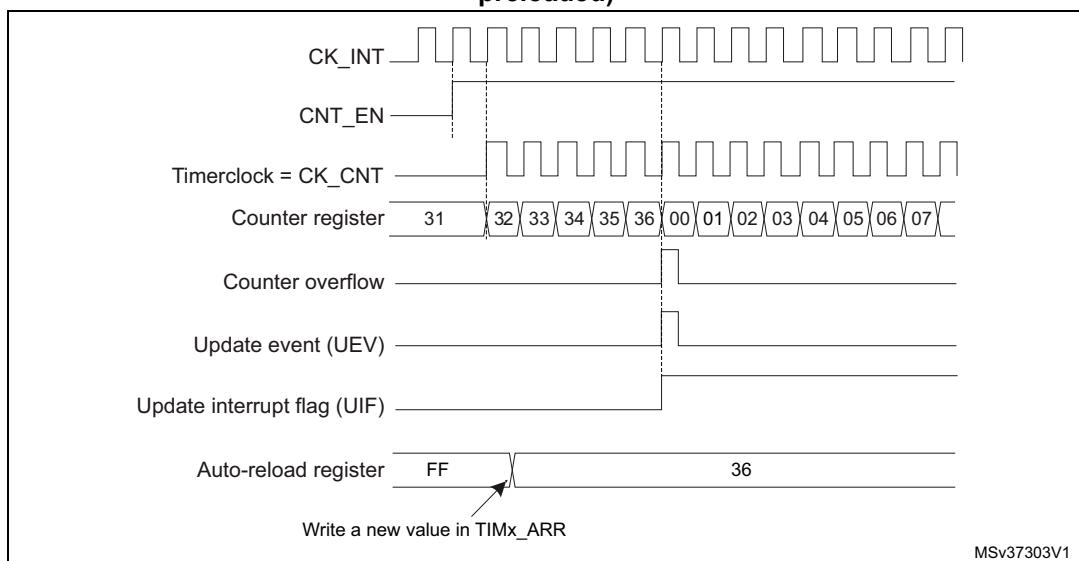
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx\_PSC register)
- The auto-reload shadow register is updated with the preload value (TIMx\_ARR)

The following figures show some examples of the counter behavior for different clock frequencies when TIMx\_ARR=0x36.

**Figure 160. Counter timing diagram, internal clock divided by 1****Figure 161. Counter timing diagram, internal clock divided by 2****Figure 162. Counter timing diagram, internal clock divided by 4**

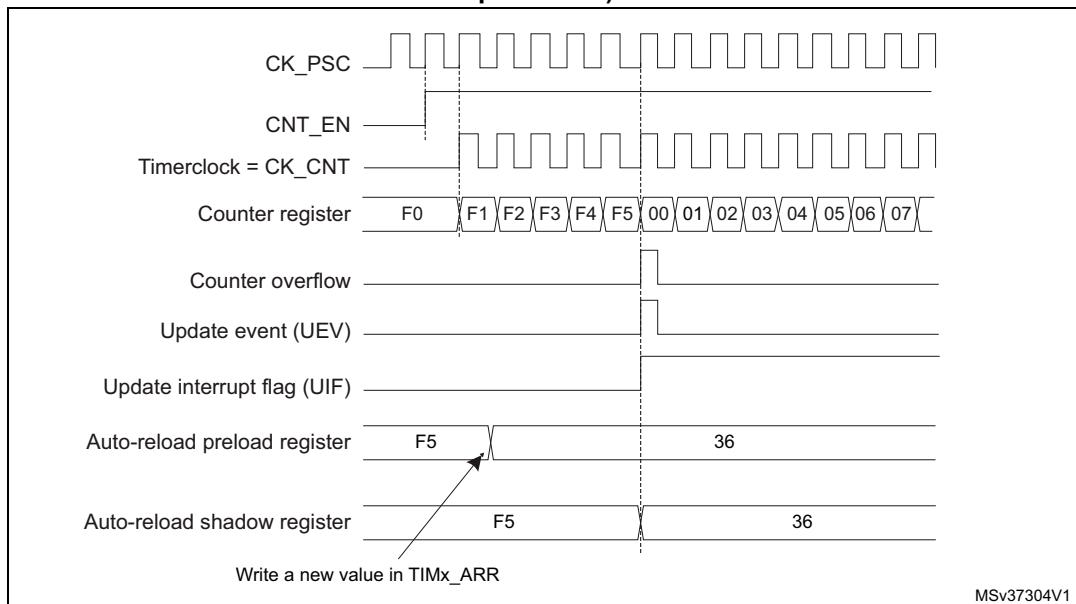
**Figure 163. Counter timing diagram, internal clock divided by N**

MSv37302V1

**Figure 164. Counter timing diagram, Update event when ARPE=0 (TIMx\_ARR not preloaded)**

MSv37303V1

**Figure 165. 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).

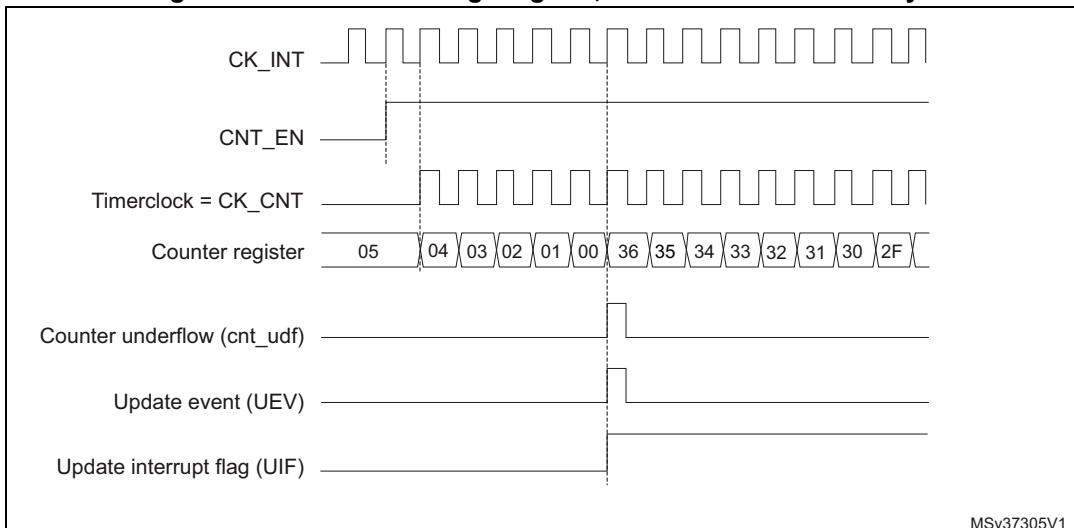
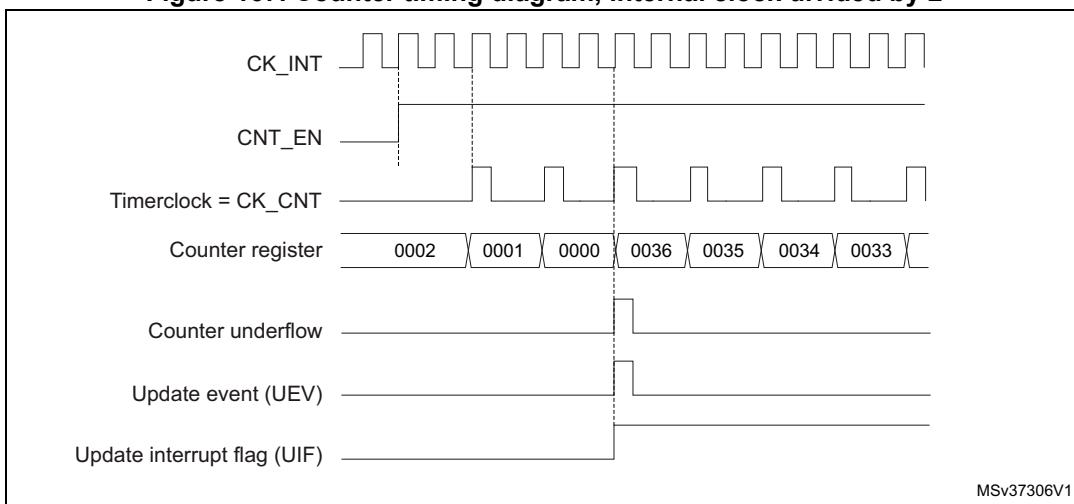
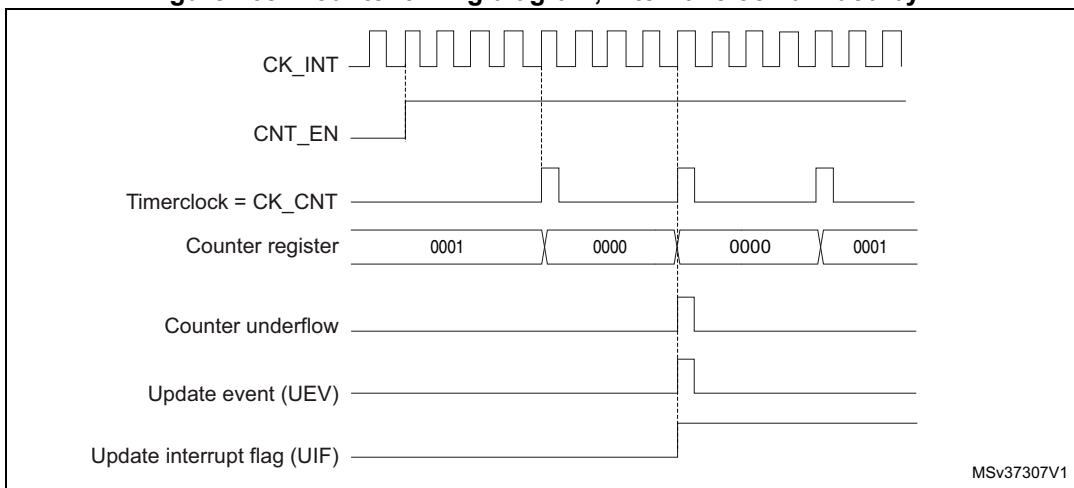
The UEV update event can be disabled by software by setting the UDIS bit in TIMx\_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until UDIS bit has been written to 0. However, the counter restarts from the current auto-reload value, whereas the counter of the prescaler restarts from 0 (but the prescale rate doesn't change).

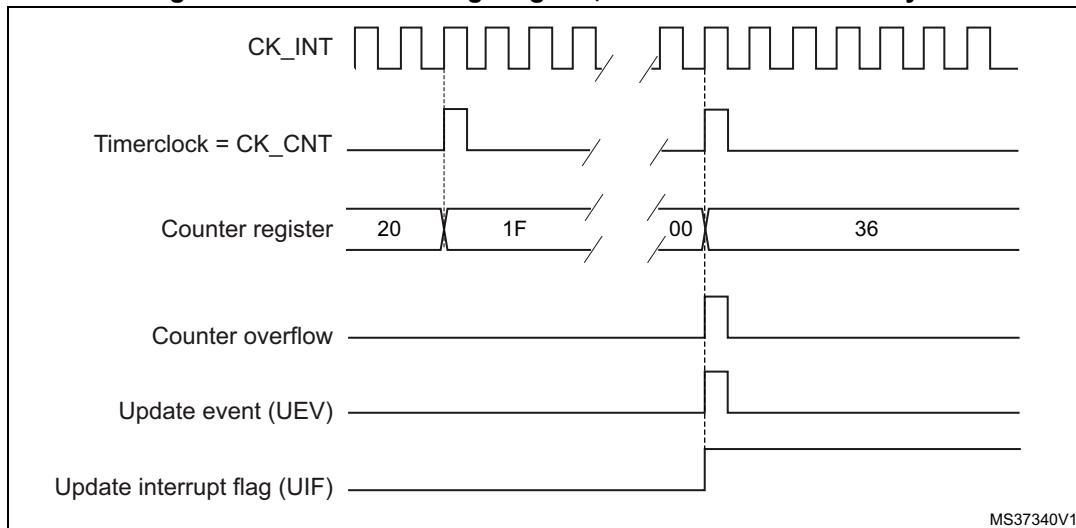
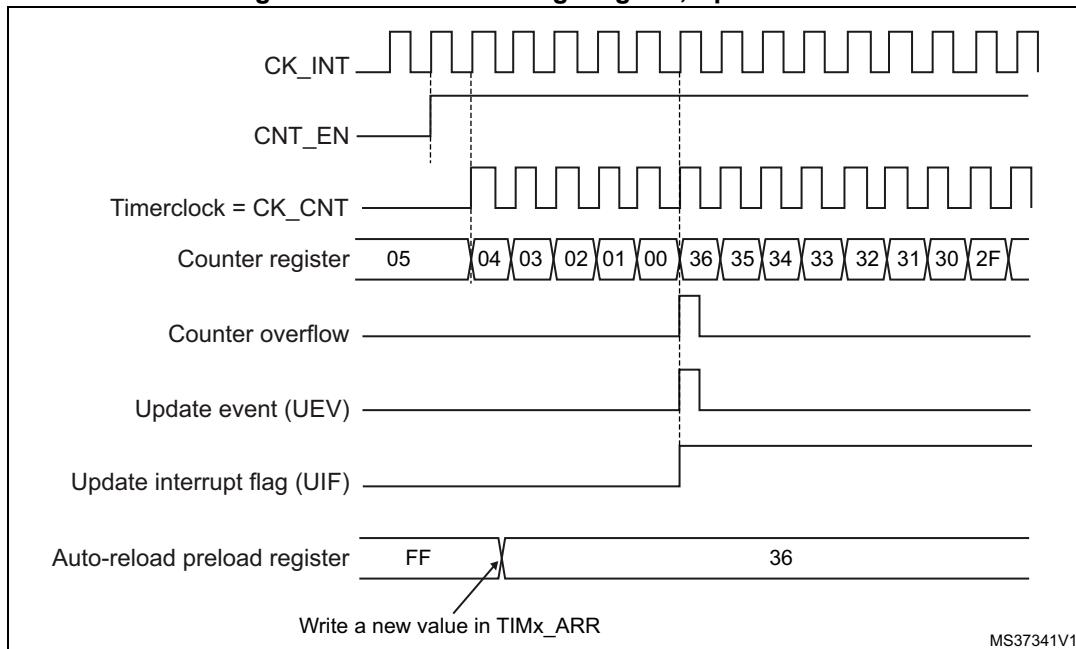
In addition, if the URS bit (update request selection) in TIMx\_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupts when clearing the counter on 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 166. Counter timing diagram, internal clock divided by 1****Figure 167. Counter timing diagram, internal clock divided by 2****Figure 168. Counter timing diagram, internal clock divided by 4**

**Figure 169. Counter timing diagram, internal clock divided by N****Figure 170. Counter timing diagram, Update event**

### 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.

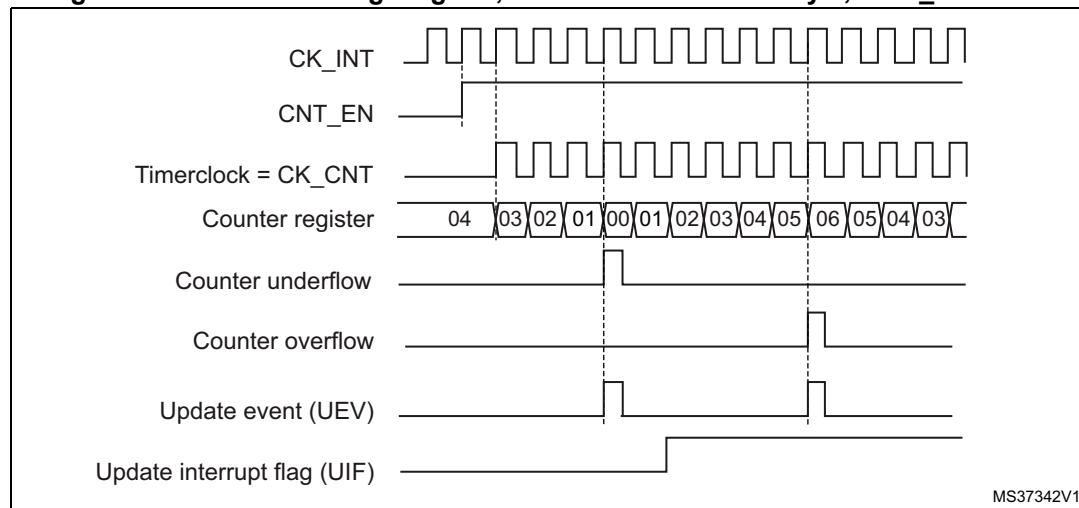
In addition, if the URS bit (update request selection) in TIMx\_CR1 register is set, setting the UG bit generates an update event UEV but without setting the UIF flag (thus no interrupt or DMA request is sent). This is to avoid generating both update and capture interrupt when clearing the counter on 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):

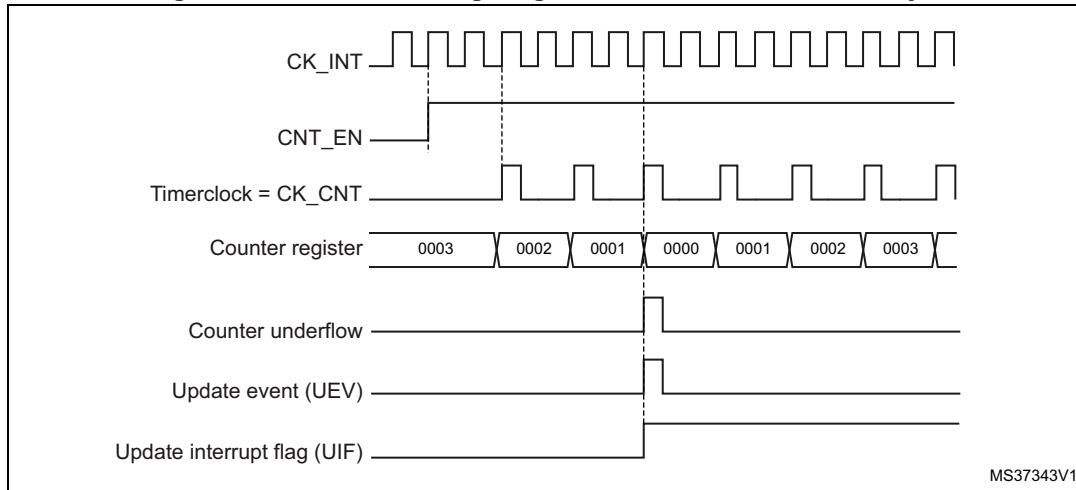
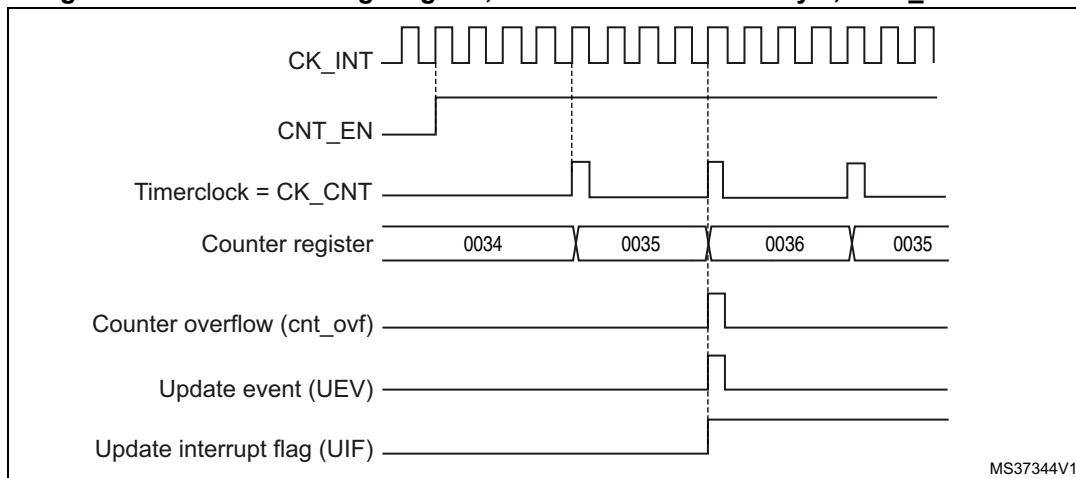
- The buffer of the prescaler is reloaded with the preload value (content of the TIMx\_PSC register).
- The auto-reload active register is updated with the preload value (content of the TIMx\_ARR register). Note that if the update source is a counter overflow, the auto-reload is updated before the counter 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.

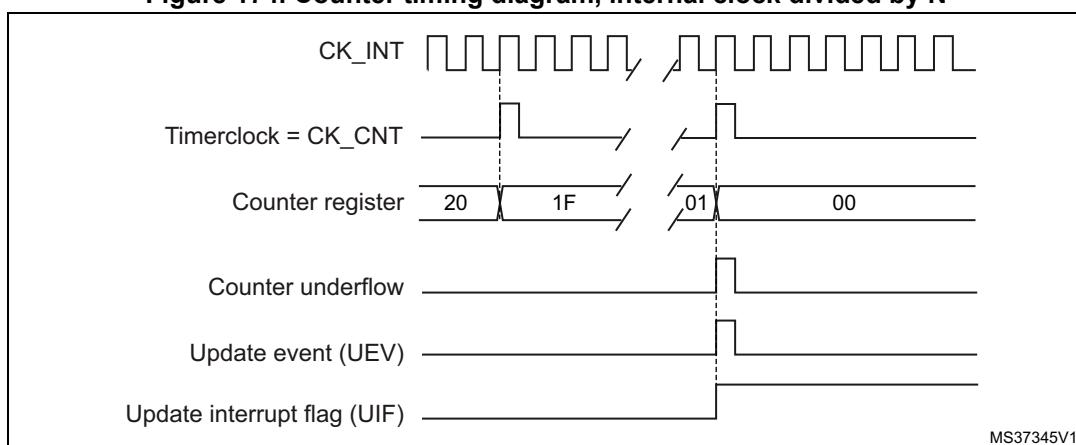
**Figure 171. Counter timing diagram, internal clock divided by 1, TIMx\_ARR=0x6**

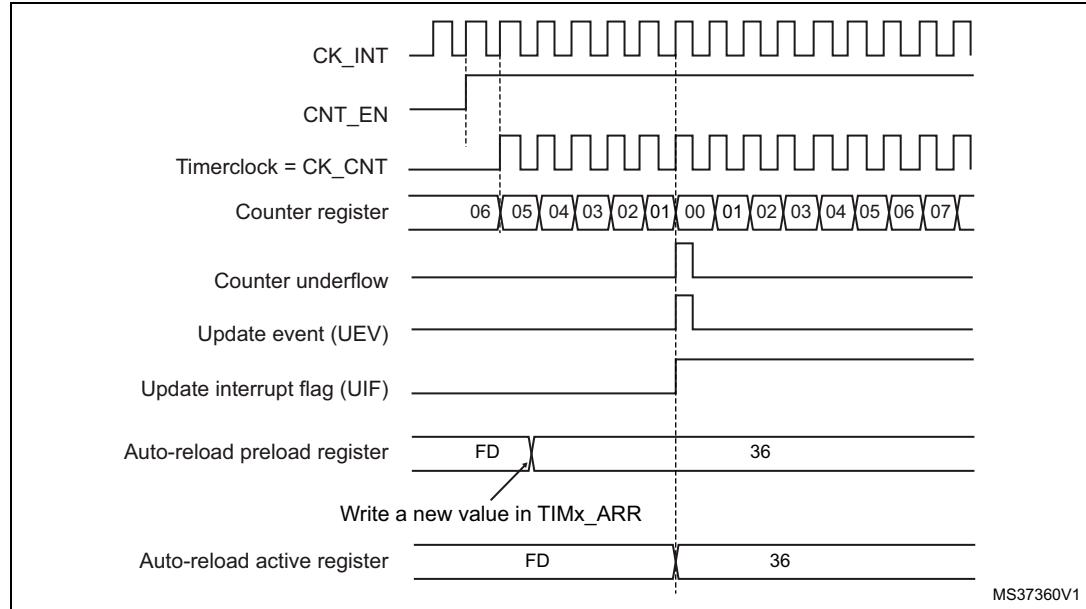
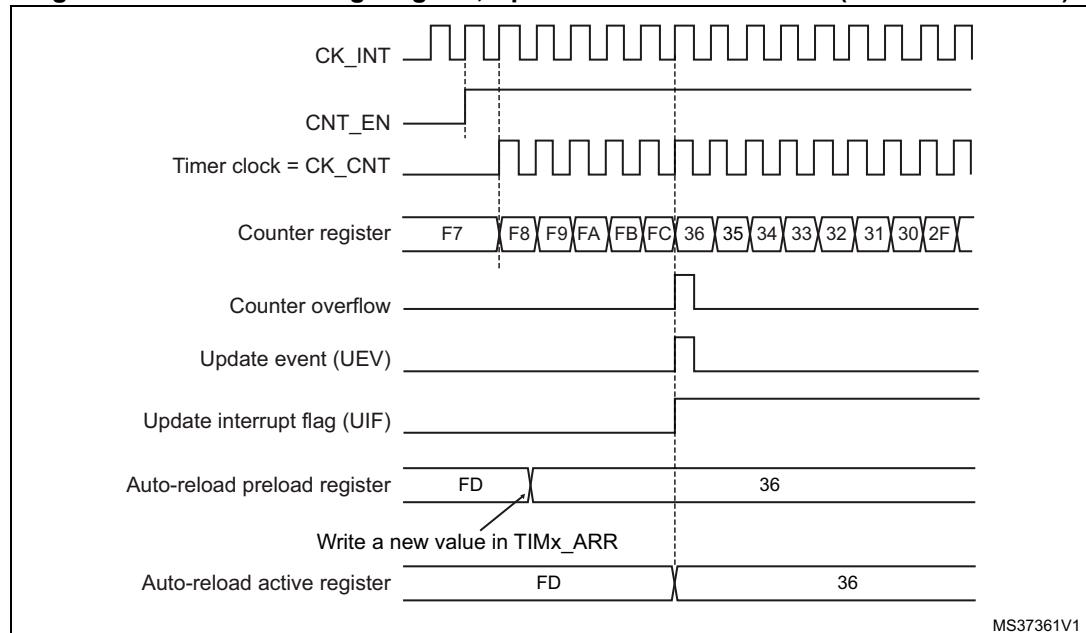


1. Here, center-aligned mode 1 is used (for more details refer to [Section 17.4.1: TIMx control register 1 \(TIMx\\_CR1\) on page 558](#)).

**Figure 172. Counter timing diagram, internal clock divided by 2****Figure 173. 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 174. Counter timing diagram, internal clock divided by N**

**Figure 175. Counter timing diagram, Update event with ARPE=1 (counter underflow)****Figure 176. Counter timing diagram, Update event with ARPE=1 (counter overflow)**

### 17.3.3 Clock selection

The counter clock can be provided by the following clock sources:

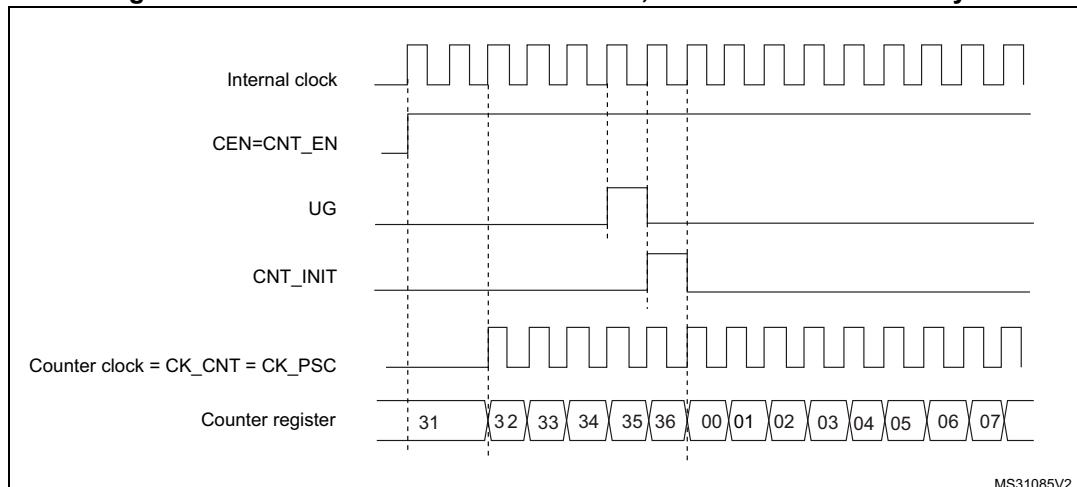
- Internal clock (CK\_INT)
- External clock mode1: external input pin (TIx)
- External clock mode2: external trigger input (ETR) available on TIM2, TIM3 and TIM4 only.
- Internal trigger inputs (ITRx): using one timer as prescaler for another timer, for example, you can configure Timer to act as a prescaler for Timer 2. Refer to [Using one timer as prescaler for another](#) 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 177* shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

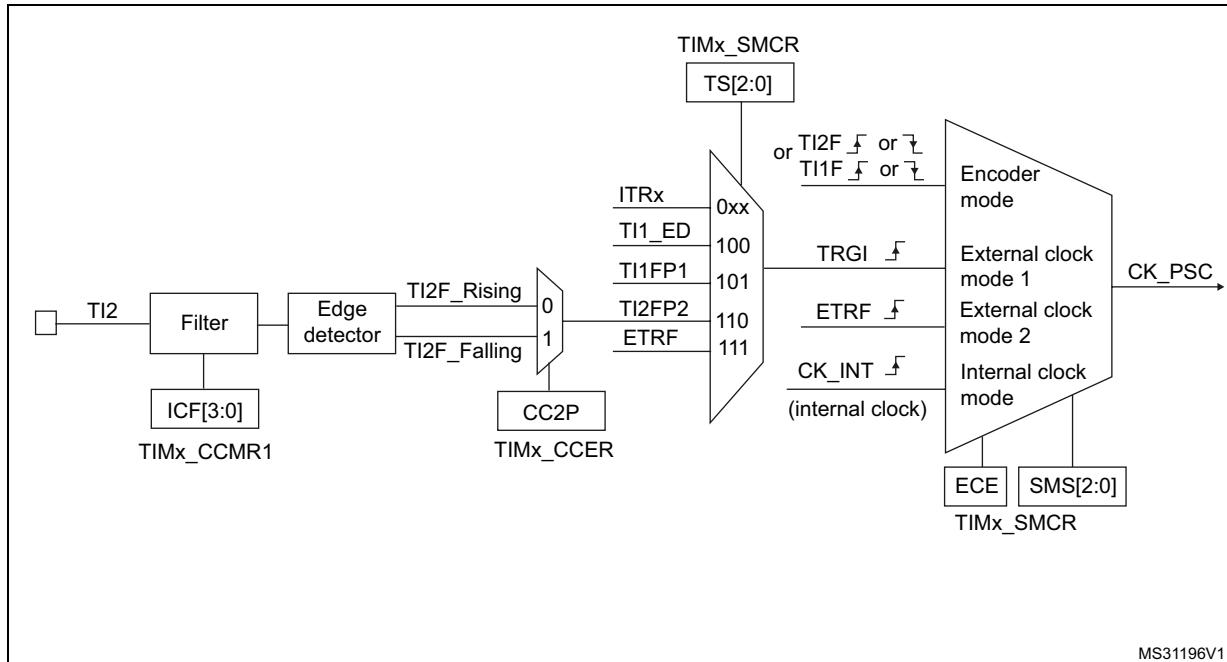
**Figure 177. Control circuit in normal mode, internal clock divided by 1**



#### 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 178. 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).

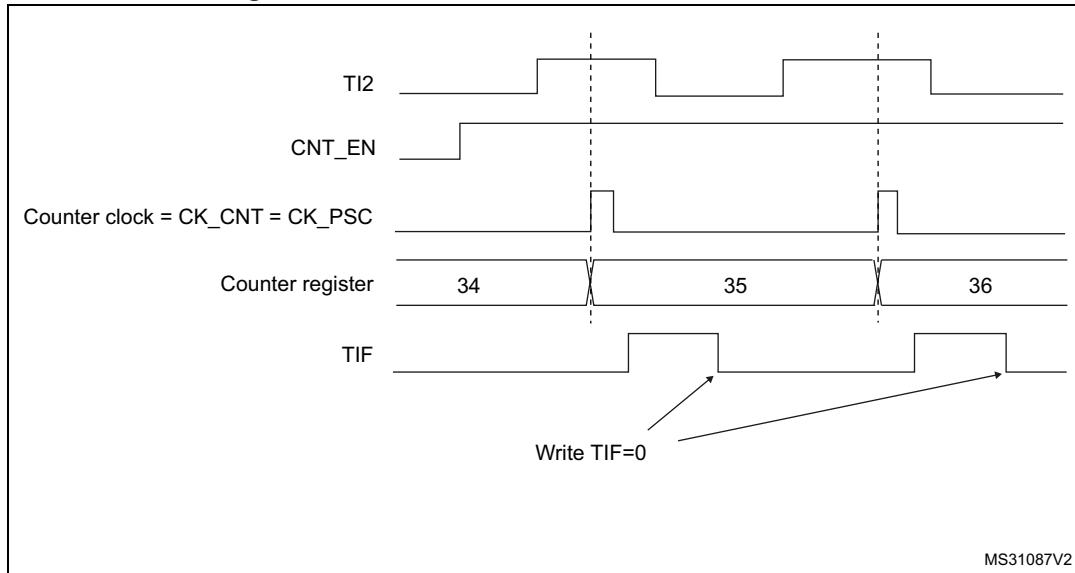
*Note:*

- 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 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 179. Control circuit in external clock mode 1



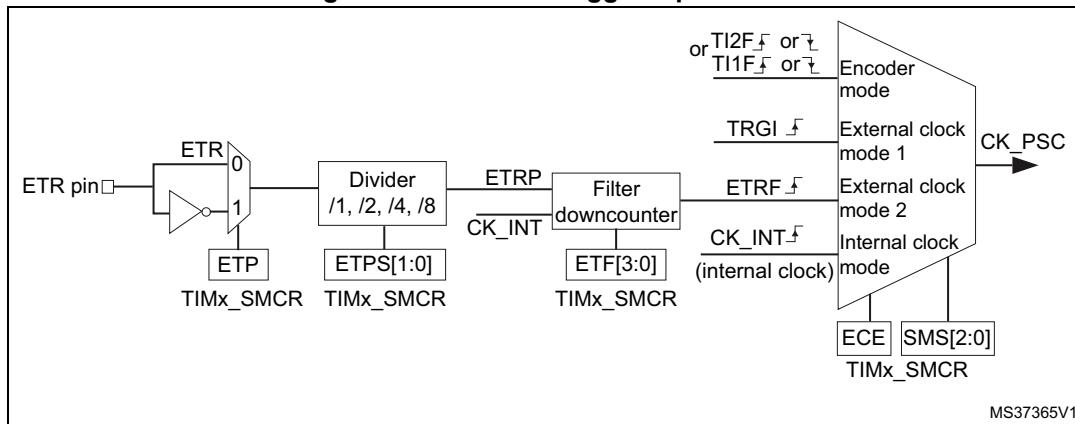
### External clock source mode 2

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 180](#) gives an overview of the external trigger input block.

Figure 180. External trigger input block



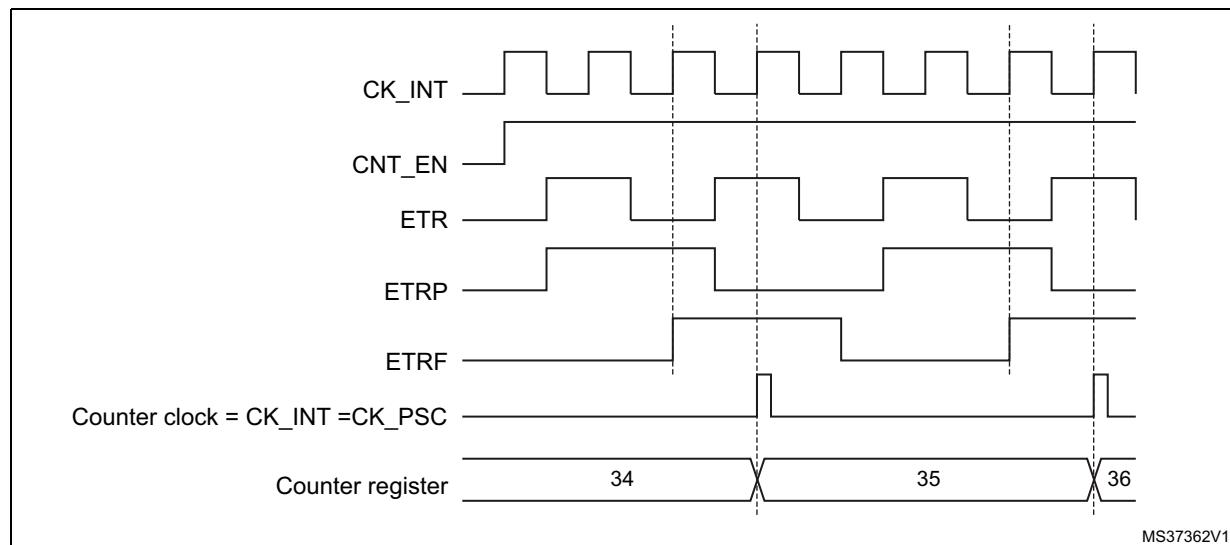
For example, to configure the upcounter to count each 2 rising edges on ETR, use the following procedure:

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 181. Control circuit in external clock mode 2**



MS37362V1

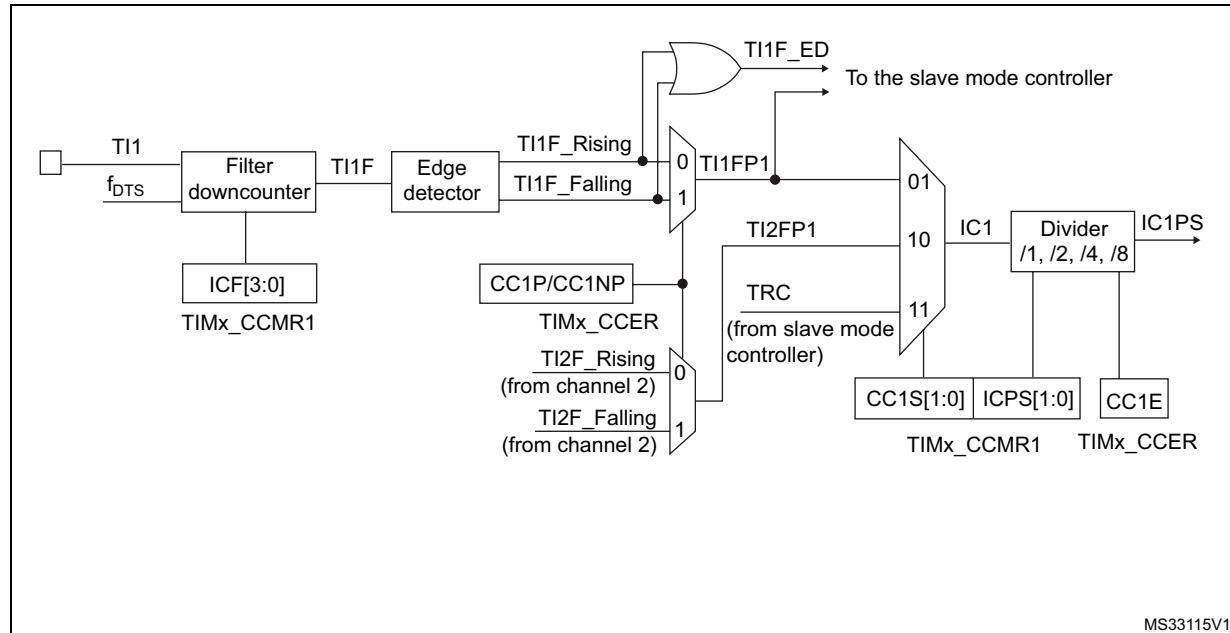
#### 17.3.4 Capture/compare channels

Each Capture/Compare channel is built around a capture/compare register (including a shadow register), a input stage for capture (with digital filter, multiplexing and prescaler) and an output stage (with comparator and output control).

The following figure gives an overview of one Capture/Compare channel.

The input stage samples the corresponding TIx input to generate a filtered signal TIxF. Then, an edge detector with polarity selection generates a signal (TIxFPx) which can be used as trigger input by the slave mode controller or as the capture command. It is prescaled before the capture register (ICxPS).

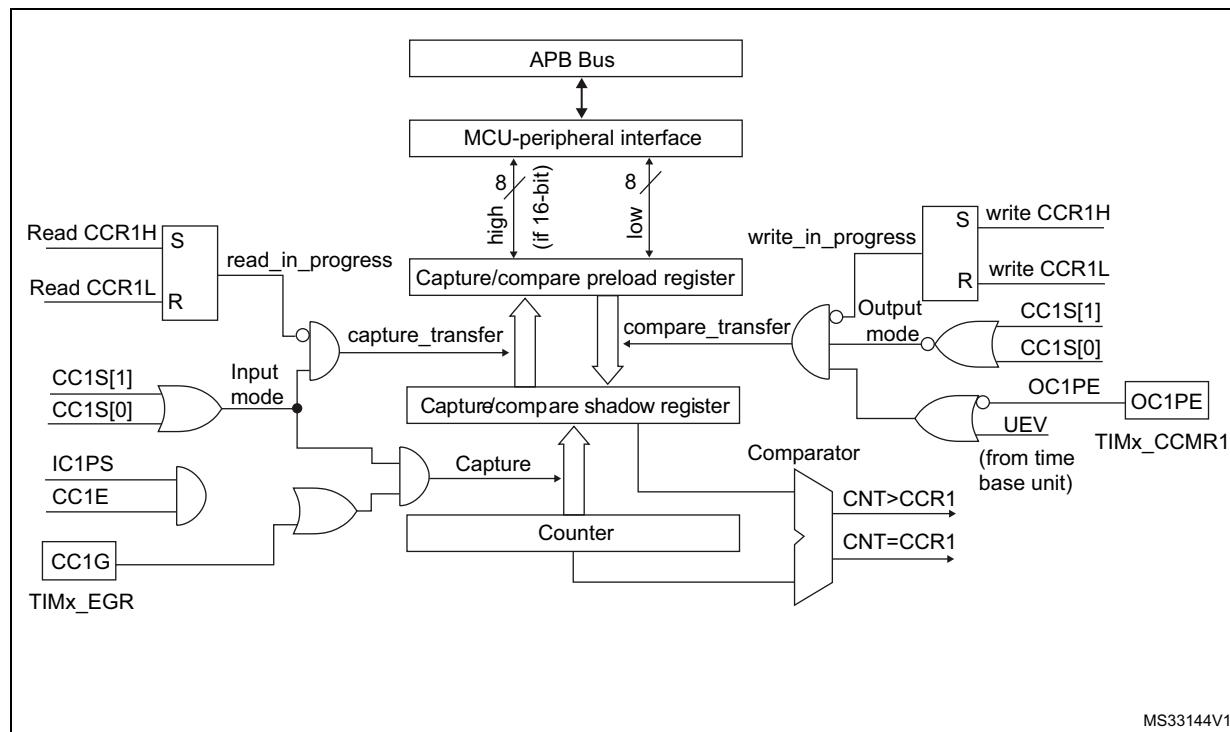
Figure 182. Capture/compare channel (example: channel 1 input stage)



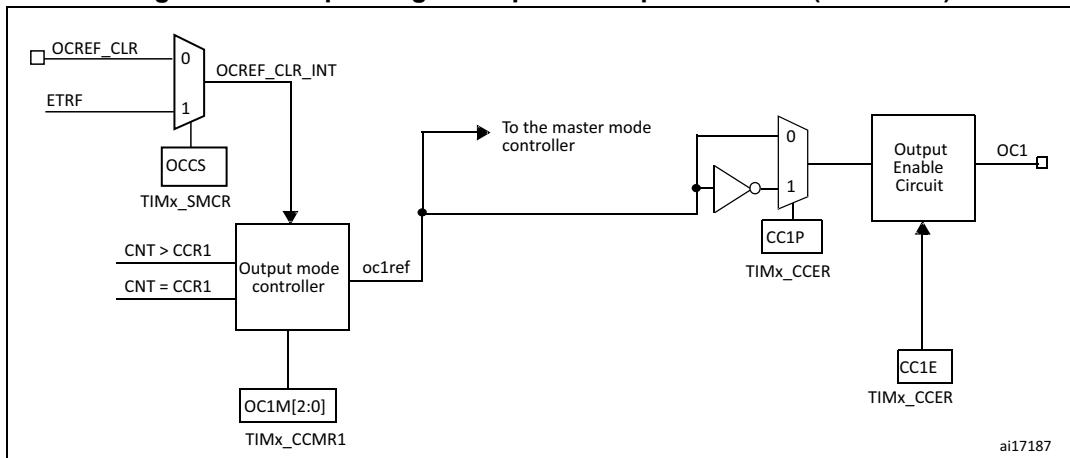
MS33115V1

The output stage generates an intermediate waveform which is then used for reference: OCxRef (active high). The polarity acts at the end of the chain.

Figure 183. Capture/compare channel 1 main circuit



MS33144V1

**Figure 184. 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.

### 17.3.5 Input capture mode

In Input capture mode, the Capture/Compare Registers (TIMx\_CCRx) are used to latch the value of the counter after a transition detected by the corresponding IC<sub>x</sub> signal. When a capture occurs, the corresponding CC<sub>x</sub>IF flag (TIMx\_SR register) is set and an interrupt or a DMA request can be sent if they are enabled. If a capture occurs while the CC<sub>x</sub>IF flag was already high, then the over-capture flag CC<sub>x</sub>OF (TIMx\_SR register) is set. CC<sub>x</sub>IF can be cleared by software by writing it to 0 or by reading the captured data stored in the TIMx\_CCRx register. CC<sub>x</sub>OF is cleared when you write it to 0.

The following example shows how to capture the counter value in TIMx\_CCR1 when TI1 input rises. To do this, use the following procedure:

- Select the active input: TIMx\_CCR1 must be linked to the TI1 input, so write the CC1S bits to 01 in the TIMx\_CCMR1 register. As soon as CC1S becomes different from 00, the channel is configured in input and the TIMx\_CCR1 register becomes read-only.
- Program the input filter duration you need with respect to the signal you connect to the timer (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.

- Select the edge of the active transition on the TI1 channel by writing the CC1P and CC1NP bits to 00 in the TIMx\_CCER register (rising edge in this case).
- Program the input prescaler. In our example, we wish the capture to be performed at each valid transition, so the prescaler is disabled (write IC1PS bits to 00 in the TIMx\_CCMR1 register).
- Enable capture from the counter into the capture register by setting the CC1E bit in the TIMx\_CCER register.
- If needed, enable the related interrupt request by setting the CC1IE bit in the TIMx\_DIER register, and/or the DMA request by setting the CC1DE bit in the TIMx\_DIER register.

When an input capture occurs:

- The TIMx\_CCR1 register gets the value of the counter on the active transition.
- CC1IF flag is set (interrupt flag). CC1OF is also set if at least two consecutive captures occurred whereas the flag was not cleared.
- An interrupt is generated depending on the CC1IE bit.
- A DMA request is generated depending on the CC1DE bit.

In order to handle the overcapture, it is recommended to read the data before the overcapture flag. This is to avoid missing an overcapture which could happen after reading the flag and before reading the data.

*Note: IC interrupt and/or DMA requests can be generated by software by setting the corresponding CCxG bit in the TIMx\_EGR register.*

### 17.3.6 PWM input mode

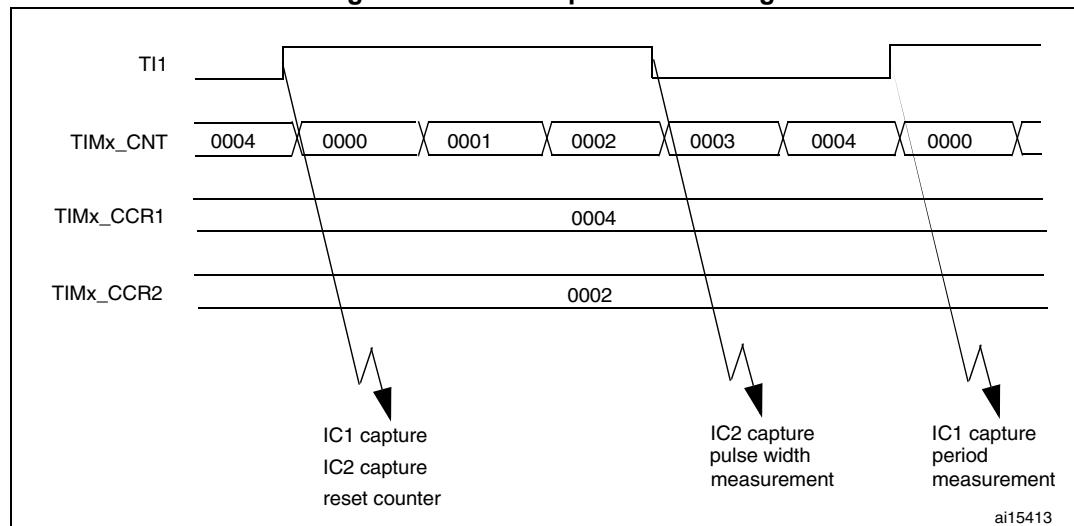
This mode is a particular case of input capture mode. The procedure is the same except:

- 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):

- Select the active input for TIMx\_CCR1: write the CC1S bits to 01 in the TIMx\_CCMR1 register (TI1 selected).
- Select the active polarity for TI1FP1 (used both for capture in TIMx\_CCR1 and counter clear): write the CC1P to '0' and the CC1NP bit to '0' (active on rising edge).
- Select the active input for TIMx\_CCR2: write the CC2S bits to 10 in the TIMx\_CCMR1 register (TI1 selected).
- Select the active polarity for TI1FP2 (used for capture in TIMx\_CCR2): write the CC2P bit to '1' and the CC2NP bit to '0' (active on falling edge).
- Select the valid trigger input: write the TS bits to 101 in the TIMx\_SMCR register (TI1FP1 selected).
- Configure the slave mode controller in reset mode: write the SMS bits to 100 in the TIMx\_SMCR register.
- Enable the captures: write the CC1E and CC2E bits to '1' in the TIMx\_CCER register.

**Figure 185. PWM input mode timing**



### 17.3.7 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.

### 17.3.8 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).

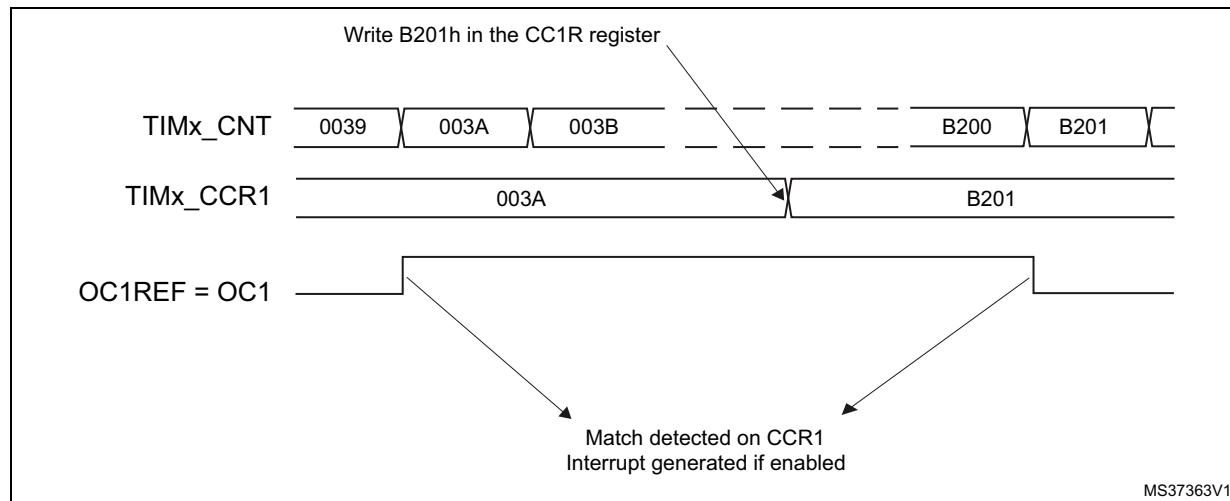
The TIMx\_CCRx registers can be programmed with or without preload registers using the OCxPE bit in the TIMx\_CCMRx register.

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 186](#).

**Figure 186. Output compare mode, toggle on OC1**

### 17.3.9 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 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  $\text{TIMx\_CCRx} \leq \text{TIMx\_CNT}$  or  $\text{TIMx\_CNT} \leq \text{TIMx\_CCRx}$  (depending on the direction of the counter). However, to comply with the ETRF (OCREF can be cleared by an external event through the ETR signal until the next PWM period), the OCREF signal is asserted only:

- When the result of the comparison changes, or
- When the output compare mode (OCxM bits in TIMx\_CCMRx register) switches from the "frozen" configuration (no comparison, OCxM='000) to one of the PWM modes (OCxM='110 or '111).

This forces the PWM by software while the timer is running.

The timer is able to generate PWM in edge-aligned mode or center-aligned mode depending on the CMS bits in the TIMx\_CR1 register.

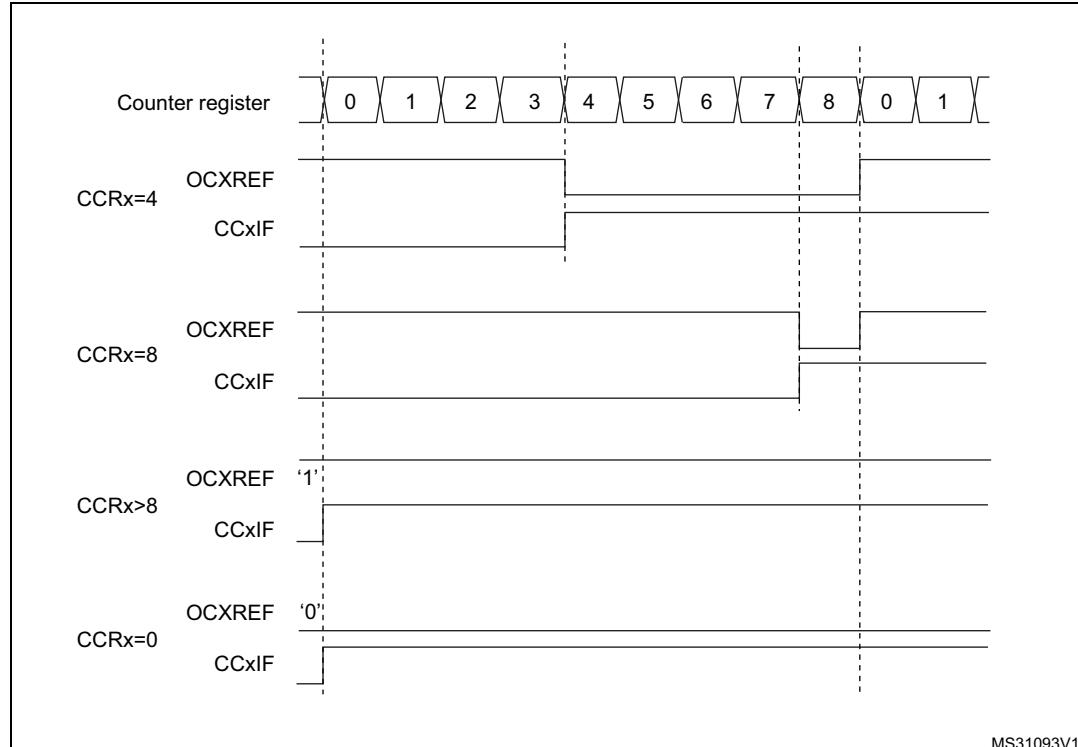
### PWM edge-aligned mode

#### Upcounting configuration

Upcounting is active when the DIR bit in the TIMx\_CR1 register is low. Refer to [Upcounting mode on page 523](#).

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 187](#) shows some edge-aligned PWM waveforms in an example where TIMx\_ARR=8.

**Figure 187. 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 526](#).

In PWM mode 1, the reference signal ocxref is low as long as TIMx\_CNT>TIMx\_CCRx else it becomes high. If the compare value in TIMx\_CCRx is greater than the auto-reload value in TIMx\_ARR, then ocxref is held at '1'. 0% PWM is not possible in this mode.

### PWM center-aligned mode

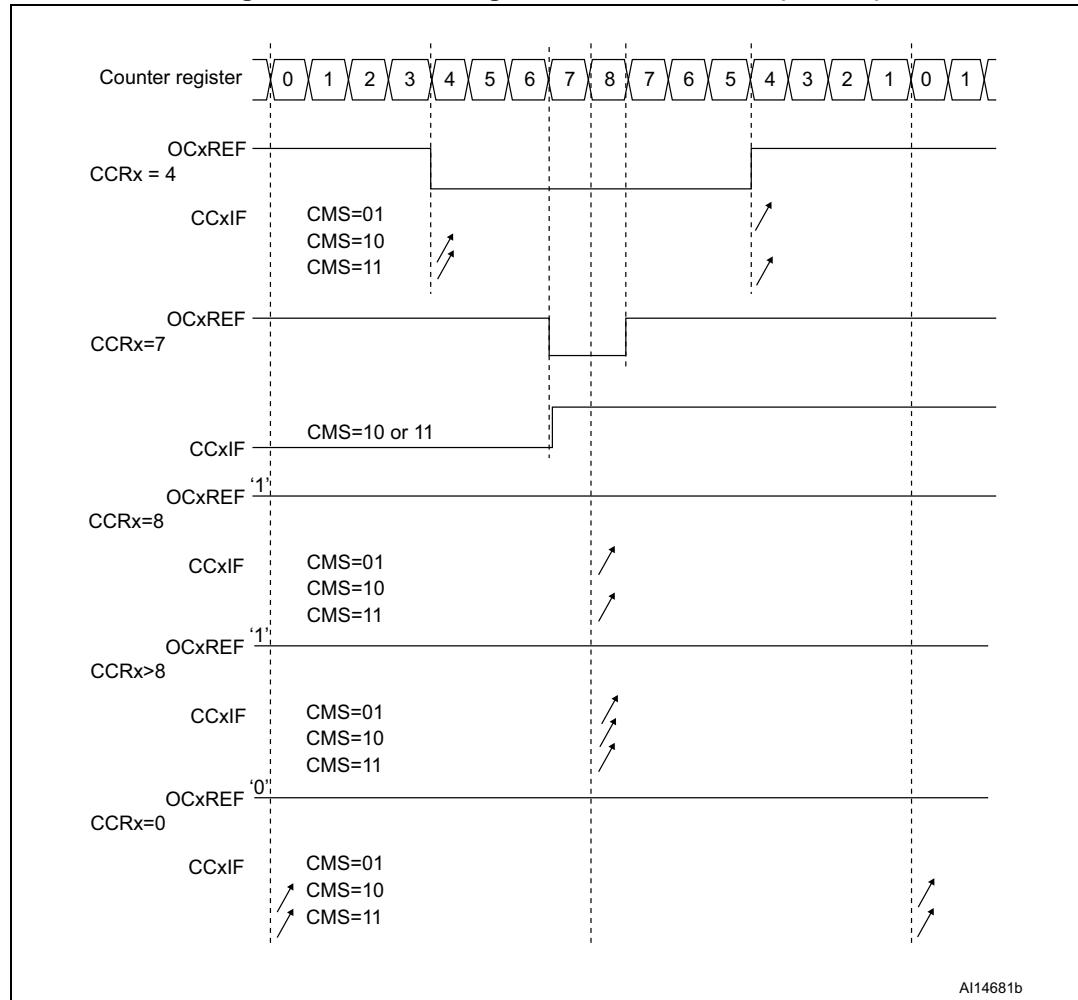
Center-aligned mode is active when the CMS bits in TIMx\_CR1 register are different from '00 (all the remaining configurations having the same effect on the ocxref/OCx signals). The compare flag is set when the counter counts up, when it counts down or both when it counts up and down depending on the CMS bits configuration. The direction bit (DIR) in the

TIMx\_CR1 register is updated by hardware and must not be changed by software. Refer to [Center-aligned mode \(up/down counting\) on page 528](#).

[Figure 188](#) 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 188. 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  $\text{TIMx\_CR1}$  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 ( $\text{TIMx\_CNT} > \text{TIMx\_ARR}$ ). For example, if the counter was counting up, it continues to count up.
  - The direction is updated if you write 0 or write the  $\text{TIMx\_ARR}$  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  $\text{TIMx\_EGR}$  register) just before starting the counter and not to write the counter while it is running.

### 17.3.10 One-pulse mode

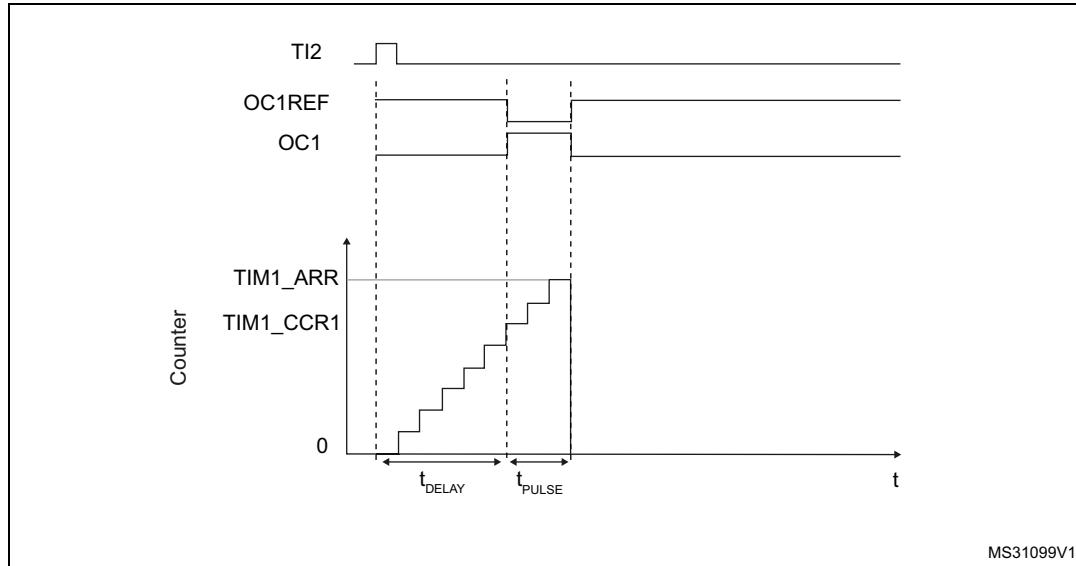
One-pulse mode (OPM) is a particular case of the previous modes. It allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length after a programmable delay.

Starting the counter can be controlled through the slave mode controller. Generating the waveform can be done in output compare mode or PWM mode. You select One-pulse mode by setting the OPM bit in the  $\text{TIMx\_CR1}$  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:  $\text{CNT} < \text{CCR}_x \leq \text{ARR}$  (in particular,  $0 < \text{CCR}_x$ ),
- In downcounting:  $\text{CNT} > \text{CCR}_x$ .

**Figure 189. Example of one-pulse mode**



For example you may want to generate a positive pulse on OC1 with a length of  $t_{\text{PULSE}}$  and after a delay of  $t_{\text{DELAY}}$  as soon as a positive edge is detected on the TI2 input pin.

Let's use TI2FP2 as trigger 1:

- Map TI2FP2 on TI2 by writing CC2S=01 in the TIMx\_CCMR1 register.
- TI2FP2 must detect a rising edge, write CC2P=0 and CC2NP='0' in the TIMx\_CCER register.
- Configure TI2FP2 as trigger for the slave mode controller (TRGI) by writing TS=110 in the TIMx\_SMCR register.
- TI2FP2 is used to start the counter by writing SMS to '110 in the TIMx\_SMCR register (trigger mode).

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

- The  $t_{DELAY}$  is defined by the value written in the TIMx\_CCR1 register.
- The  $t_{PULSE}$  is defined by the difference between the auto-reload value and the compare value ( $TIMx\_ARR - TIMx\_CCR + 1$ ).
- 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.

#### **17.3.11 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.

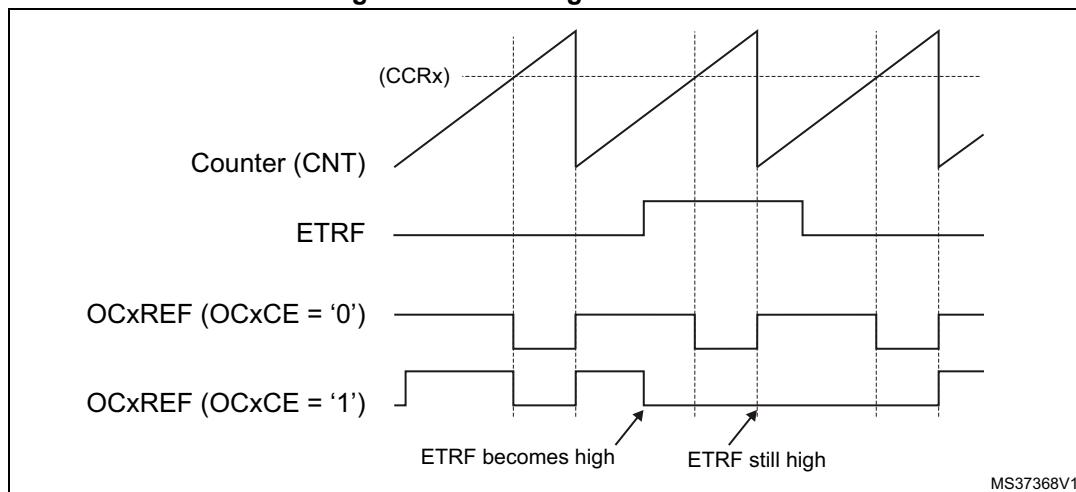
This function can only be used in output compare and PWM modes, and does not work in forced mode.

For example, the ETR 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 190* shows the behavior of the OCxREF signal when the ETRF input becomes high, for both values of the OCxCE enable bit. In this example, the timer TIMx is programmed in PWM mode.

**Figure 190. Clearing TIMx OCxREF**



1. In case of a PWM with a 100% duty cycle (if CCRx>ARR), OCxREF is enabled again at the next counter overflow.

### 17.3.12 Encoder interface mode

To select Encoder Interface mode write SMS='001 in the TIMx\_SMCR register if the counter is counting on TI2 edges only, SMS=010 if it is counting on TI1 edges only and SMS=011 if it is counting on both TI1 and TI2 edges.

Select the TI1 and TI2 polarity by programming the CC1P and CC2P bits in the TIMx\_CCER register. When needed, you can program the input filter as well.

The two inputs TI1 and TI2 are used to interface to an incremental encoder. Refer to *Table 111*. The counter is clocked by each valid transition on TI1FP1 or TI2FP2 (TI1 and TI2 after input filter and polarity selection, TI1FP1=TI1 if not filtered and not inverted, TI2FP2=TI2 if not filtered and not inverted) assuming that it is enabled (CEN bit in TIMx\_CR1 register written to '1'). The sequence of transitions of the two inputs is evaluated and generates count pulses as well as the direction signal. Depending on the sequence the counter counts up or down, the DIR bit in the TIMx\_CR1 register is modified by hardware accordingly. The DIR bit is calculated at each transition on any input (TI1 or TI2), whatever the counter is counting on TI1 only, TI2 only or both TI1 and TI2.

Encoder interface mode acts simply as an external clock with direction selection. This means that the counter just counts continuously between 0 and the auto-reload value in the TIMx\_ARR register (0 to ARR or ARR down to 0 depending on the direction). So you must configure TIMx\_ARR before starting. In the same way, the capture, compare, prescaler, trigger output features continue to work as normal.

In this mode, the counter is modified automatically following the speed and the direction of the incremental encoder and its content, therefore, always represents the encoder's position. The count direction correspond to the rotation direction of the connected sensor. The table summarizes the possible combinations, assuming TI1 and TI2 don't switch at the same time.

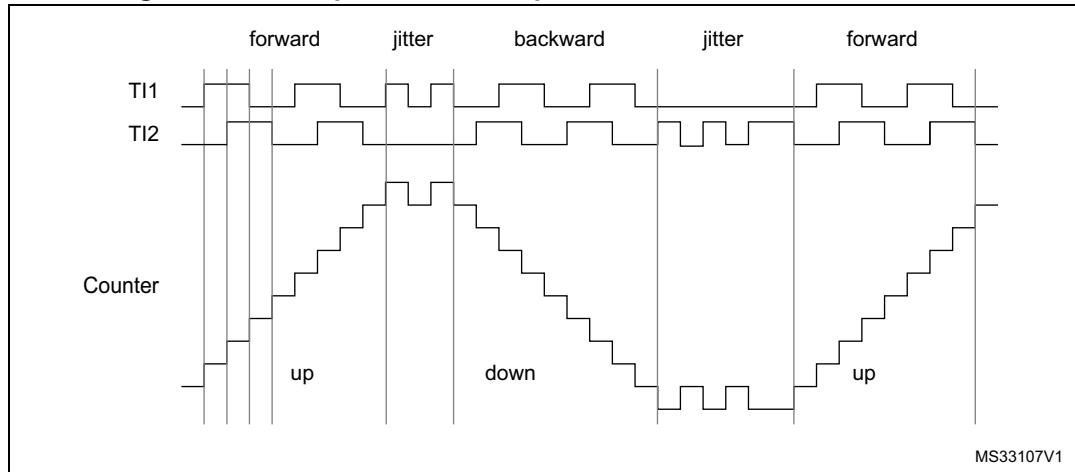
**Table 111. Counting direction versus encoder signals**

<b>Active edge</b>	<b>Level on opposite signal (TI1FP1 for TI2, TI2FP2 for TI1)</b>	<b>TI1FP1 signal</b>		<b>TI2FP2 signal</b>	
		<b>Rising</b>	<b>Falling</b>	<b>Rising</b>	<b>Falling</b>
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

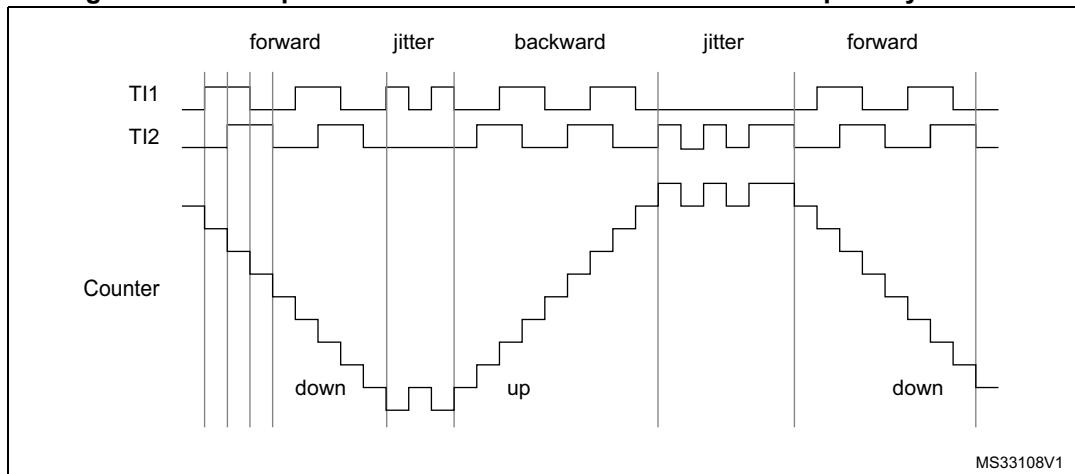
An external incremental encoder can be connected directly to the MCU without external interface logic. However, comparators are normally be used to convert the encoder's differential outputs to digital signals. This greatly increases noise immunity. The third encoder output which indicate the mechanical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

*Figure 191* gives an example of counter operation, showing count signal generation and direction control. It also shows how input jitter is compensated where both edges are selected. This might occur if the sensor is positioned near to one of the switching points. For this example we assume that the configuration is the following:

- CC1S= '01' (TIMx\_CCMR1 register, TI1FP1 mapped on TI1)
- CC2S= '01' (TIMx\_CCMR2 register, TI2FP2 mapped on TI2)
- CC1P= '0', CC1NP = '0', IC1F ='0000' (TIMx\_CCER register, TI1FP1 noninverted, TI1FP1=TI1)
- CC2P= '0', CC2NP = '0', IC2F ='0000' (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 191. Example of counter operation in encoder interface mode**

*Figure 192* gives an example of counter behavior when TI1FP1 polarity is inverted (same configuration as above except CC1P=1).

**Figure 192. 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.

### 17.3.13 Timer input XOR function

The TI1S bit in the TIM\_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.

### 17.3.14 Timers and external trigger synchronization

The TIMx Timers can be synchronized with an external trigger in several modes: Reset mode, Gated mode and Trigger mode.

#### Slave mode: Reset mode

The counter and its prescaler can be reinitialized in response to an event on a trigger input. Moreover, if the URS bit from the TIMx\_CR1 register is low, an update event UEV is generated. Then all the preloaded registers (TIMx\_ARR, TIMx\_CCRx) are updated.

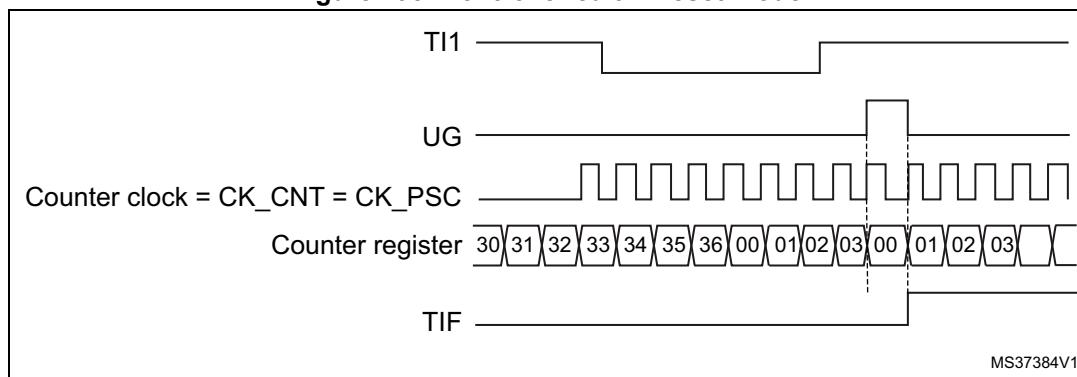
In the following example, the upcounter is cleared in response to a rising edge on TI1 input:

- Configure the channel 1 to detect rising edges on TI1. Configure the input filter duration (in this example, we 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 193. Control circuit in reset mode**



### Slave mode: Gated mode

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

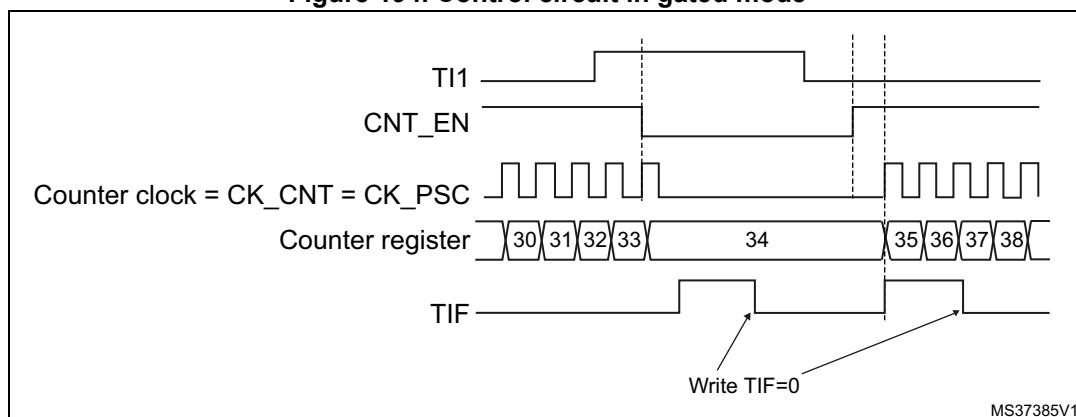
In the following example, the upcounter counts only when TI1 input is low:

- Configure the channel 1 to detect low levels on TI1. Configure the input filter duration (in this example, we don't need any filter, so we keep IC1F=0000). The capture prescaler is not used for triggering, so you don't need to configure it. The CC1S bits select the input capture source only, CC1S=01 in TIMx\_CCMR1 register. Write CC1P=1 in TIMx\_CCER register to validate the polarity (and detect low level only).
- Configure the timer in gated mode by writing SMS=101 in TIMx\_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx\_SMCR register.
- Enable the counter by writing CEN=1 in the TIMx\_CR1 register (in gated mode, the counter doesn't start if CEN=0, whatever is the trigger input level).

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

The delay between the rising edge on TI1 and the actual stop of the counter is due to the resynchronization circuit on TI1 input.

**Figure 194. Control circuit in gated mode**



- The configuration "CCxP=CCxNP=1" (detection of both rising and falling edges) does not have any effect in gated mode because gated mode acts on a level and not on an edge.

### Slave mode: Trigger mode

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

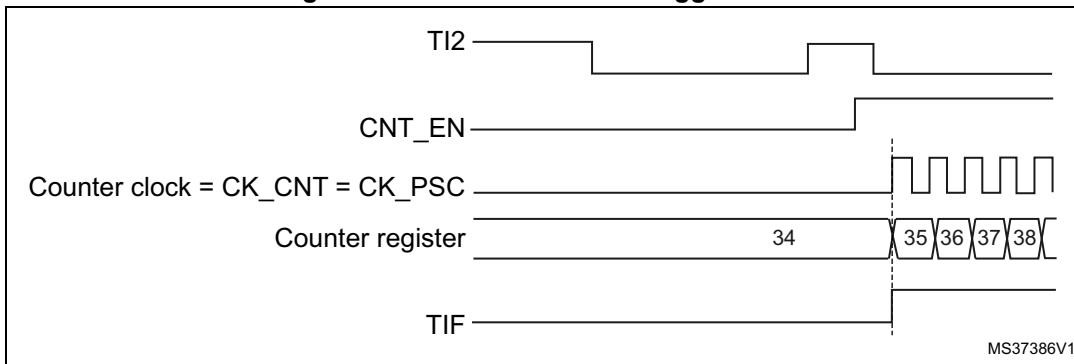
In the following example, the upcounter starts in response to a rising edge on TI2 input:

- Configure the channel 2 to detect rising edges on TI2. Configure the input filter duration (in this example, we don't need any filter, so we keep IC2F=0000). The capture prescaler is not used for triggering, so you don't need to configure it. CC2S bits are selecting the input capture source only, CC2S=01 in TIMx\_CCMR1 register. Write CC2P=1 in TIMx\_CCER register to validate the polarity (and detect low level only).
- Configure the timer in trigger mode by writing SMS=110 in TIMx\_SMCR register. Select TI2 as the input source by writing TS=110 in TIMx\_SMCR register.

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

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

**Figure 195. Control circuit in trigger mode**



### Slave mode: External Clock mode 2 + trigger mode

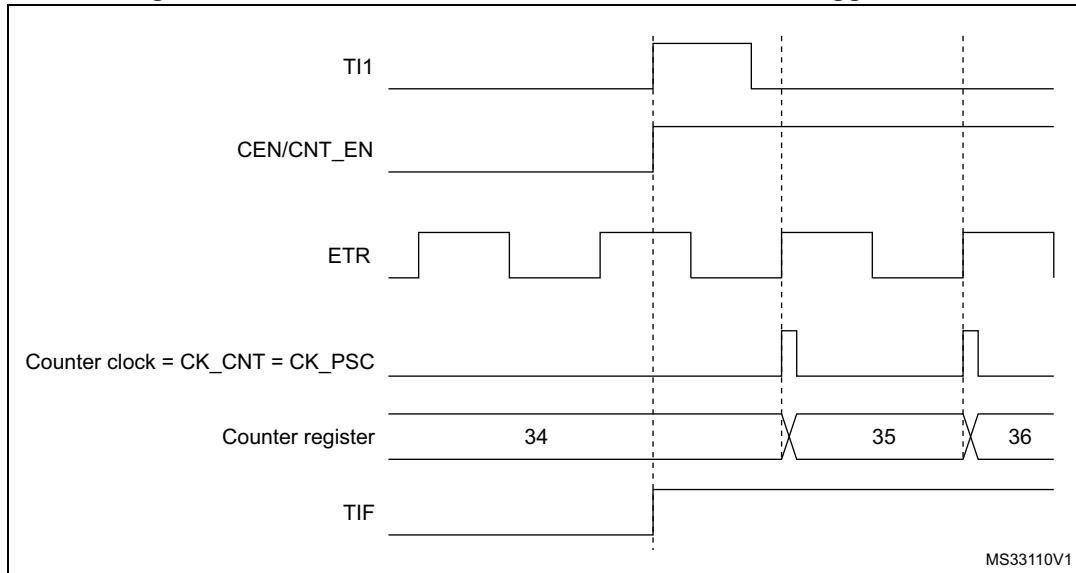
The external clock mode 2 can be used in addition to another slave mode (except external clock mode 1 and encoder mode). In this case, the ETR signal is used as external clock input, and another input can be selected as trigger input when operating in reset mode, gated mode or trigger mode. It is recommended not to select ETR as TRGI through the TS bits of TIMx\_SMCR register.

In the following example, the upcounter is incremented at each rising edge of the ETR signal as soon as a rising edge of TI1 occurs:

1. Configure the external trigger input circuit by programming the TIMx\_SMCR register as follows:
  - ETF = 0000: no filter
  - ETPS = 00: prescaler disabled
  - ETP = 0: detection of rising edges on ETR and ECE=1 to enable the external clock mode 2.
2. Configure the channel 1 as follows, to detect rising edges on TI:
  - IC1F = 0000: no filter.
  - The capture prescaler is not used for triggering and does not need to be configured.
  - CC1S = 01 in TIMx\_CCMR1 register to select only the input capture source
  - CC1P = 0 in TIMx\_CCER register to validate the polarity (and detect rising edge only).
3. Configure the timer in trigger mode by writing SMS=110 in TIMx\_SMCR register. Select TI1 as the input source by writing TS=101 in TIMx\_SMCR register.

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

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

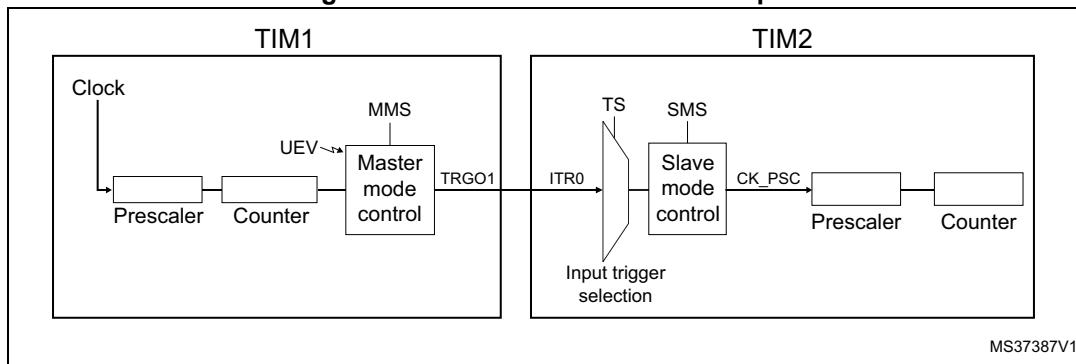
**Figure 196. Control circuit in external clock mode 2 + trigger mode**

### 17.3.15 Timer synchronization

The TIMx timers are linked together internally for timer synchronization or chaining. When one Timer is configured in Master Mode, it can reset, start, stop or clock the counter of another Timer configured in Slave Mode.

*Figure 197* presents an overview of the trigger selection and the master mode selection blocks.

#### Using one timer as prescaler for another

**Figure 197. Master/Slave timer example**

For example, you can configure Timer 1 to act as a prescaler for Timer 2. Refer to [Figure 197](#). To do this:

- Configure Timer 1 in master mode so that it outputs a periodic trigger signal on each update event UEV. If you write MMS=010 in the TIM1\_CR2 register, a rising edge is output on TRGO1 each time an update event is generated.
- To connect the TRGO1 output of Timer 1 to Timer 2, Timer 2 must be configured in slave mode using ITR0 as internal trigger. You select this through the TS bits in the TIM2\_SMCR register (writing TS=000).
- Then you put the slave mode controller in external clock mode 1 (write SMS=111 in the TIM2\_SMCR register). This causes Timer 2 to be clocked by the rising edge of the periodic Timer 1 trigger signal (which correspond to the timer 1 counter overflow).
- Finally both timers must be enabled by setting their respective CEN bits (TIMx\_CR1 register).

**Note:** *If OCx is selected on Timer 1 as trigger output (MMS=1xx), its rising edge is used to clock the counter of timer 2.*

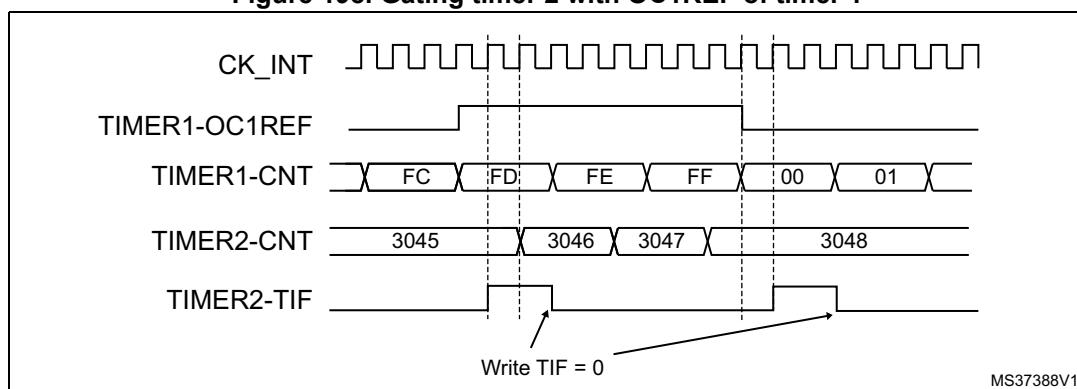
### Using one timer to enable another timer

In this example, we control the enable of Timer 2 with the output compare 1 of Timer 1. Refer to [Figure 197](#) for connections. Timer 2 counts on the divided internal clock only when OC1REF of Timer 1 is high. Both counter clock frequencies are divided by 3 by the prescaler compared to CK\_INT ( $f_{CK\_CNT} = f_{CK\_INT}/3$ ).

- Configure Timer 1 master mode to send its Output Compare 1 Reference (OC1REF) signal as trigger output (MMS=100 in the TIM1\_CR2 register).
- Configure the Timer 1 OC1REF waveform (TIM1\_CCMR1 register).
- Configure Timer 2 to get the input trigger from Timer 1 (TS=000 in the TIM2\_SMCR register).
- Configure Timer 2 in gated mode (SMS=101 in TIM2\_SMCR register).
- Enable Timer 2 by writing '1 in the CEN bit (TIM2\_CR1 register).
- Start Timer 1 by writing '1 in the CEN bit (TIM1\_CR1 register).

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

**Figure 198. Gating timer 2 with OC1REF of timer 1**



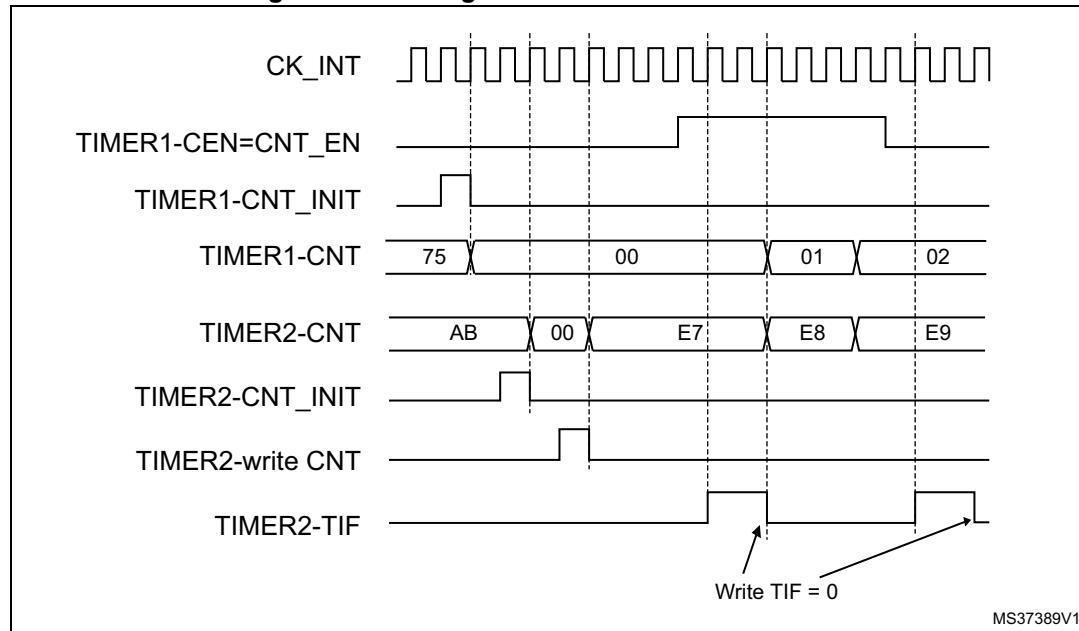
In the example in [Figure 198](#), the Timer 2 counter and prescaler are not initialized before being started. So they start counting from their current value. It is possible to start from a given value by resetting both timers before starting Timer 1. You can then write any value

you want in the timer counters. The timers can easily be reset by software using the UG bit in the TIMx\_EGR registers.

In the next example, we synchronize Timer 1 and Timer 2. Timer 1 is the master and starts from 0. Timer 2 is the slave and starts from 0xE7. The prescaler ratio is the same for both timers. Timer 2 stops when Timer 1 is disabled by writing '0' to the CEN bit in the TIM1\_CR1 register:

- Configure Timer 1 master mode to send its Output Compare 1 Reference (OC1REF) signal as trigger output (MMS=100 in the TIM1\_CR2 register).
- Configure the Timer 1 OC1REF waveform (TIM1\_CCMR1 register).
- Configure Timer 2 to get the input trigger from Timer 1 (TS=000 in the TIM2\_SMCR register).
- Configure Timer 2 in gated mode (SMS=101 in TIM2\_SMCR register).
- Reset Timer 1 by writing '1' in UG bit (TIM1\_EGR register).
- Reset Timer 2 by writing '1' in UG bit (TIM2\_EGR register).
- Initialize Timer 2 to 0xE7 by writing '0xE7' in the timer 2 counter (TIM2\_CNT).
- Enable Timer 2 by writing '1' in the CEN bit (TIM2\_CR1 register).
- Start Timer 1 by writing '1' in the CEN bit (TIM1\_CR1 register).
- Stop Timer 1 by writing '0' in the CEN bit (TIM1\_CR1 register).

**Figure 199. Gating timer 2 with Enable of timer 1**



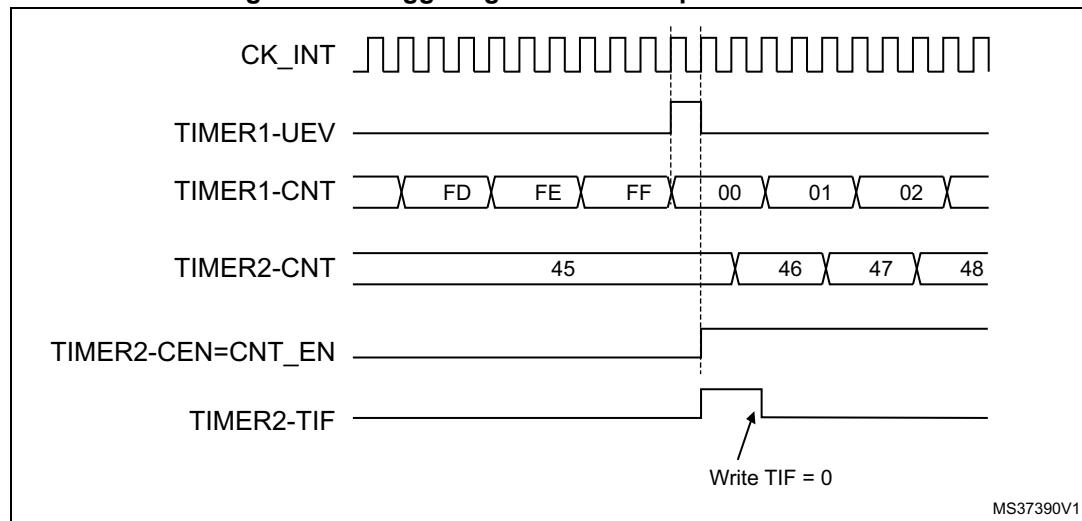
### Using one timer to start another timer

In this example, we set the enable of Timer 2 with the update event of Timer 1. Refer to [Figure 197](#) for connections. Timer 2 starts counting from its current value (which can be nonzero) on the divided internal clock as soon as the update event is generated by Timer 1. When Timer 2 receives the trigger signal its CEN bit is automatically set and the counter

counts until we write '0' to the CEN bit in the TIM2\_CR1 register. Both counter clock frequencies are divided by 3 by the prescaler compared to CK\_INT ( $f_{CK\_CNT} = f_{CK\_INT}/3$ ).

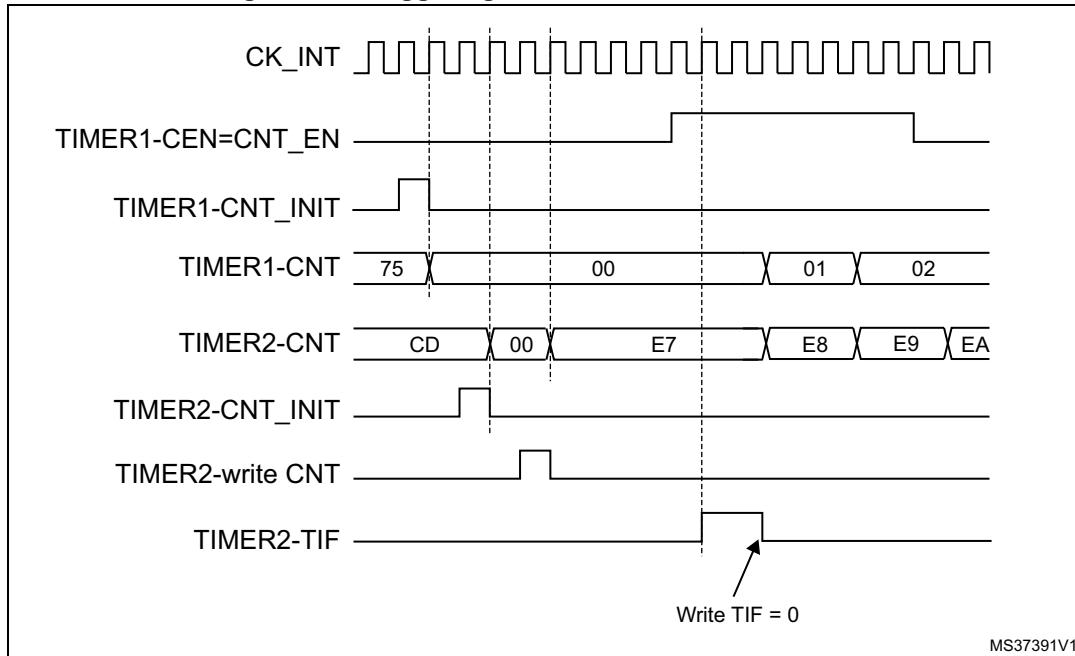
- Configure Timer 1 master mode to send its Update Event (UEV) as trigger output (MMS=010 in the TIM1\_CR2 register).
- Configure the Timer 1 period (TIM1\_ARR registers).
- Configure Timer 2 to get the input trigger from Timer 1 (TS=000 in the TIM2\_SMCR register).
- Configure Timer 2 in trigger mode (SMS=110 in TIM2\_SMCR register).
- Start Timer 1 by writing '1' in the CEN bit (TIM1\_CR1 register).

**Figure 200. Triggering timer 2 with update of timer 1**



As in the previous example, you can initialize both counters before starting counting. [Figure 201](#) shows the behavior with the same configuration as in [Figure 198](#), but in trigger mode instead of gated mode (SMS=110 in the TIM2\_SMCR register).

Figure 201. Triggering timer 2 with Enable of timer 1



### Using one timer as prescaler for another timer

For example, you can configure Timer 1 to act as a prescaler for Timer 2. Refer to [Figure 197](#) for connections. To do this:

- Configure Timer 1 master mode to send its Update Event (UEV) as trigger output (MMS=010 in the TIM1\_CR2 register). then it outputs a periodic signal on each counter overflow.
- Configure the Timer 1 period (TIM1\_ARR registers).
- Configure Timer 2 to get the input trigger from Timer 1 (TS=000 in the TIM2\_SMCR register).
- Configure Timer 2 in external clock mode 1 (SMS=111 in TIM2\_SMCR register).
- Start Timer 2 by writing '1 in the CEN bit (TIM2\_CR1 register).
- Start Timer 1 by writing '1 in the CEN bit (TIM1\_CR1 register).

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

In this example, we set the enable of timer 1 when its TI1 input rises, and the enable of Timer 2 with the enable of Timer 1. Refer to [Figure 197](#) for connections. To ensure the

counters are aligned, Timer 1 must be configured in Master/Slave mode (slave with respect to TI1, master with respect to Timer 2):

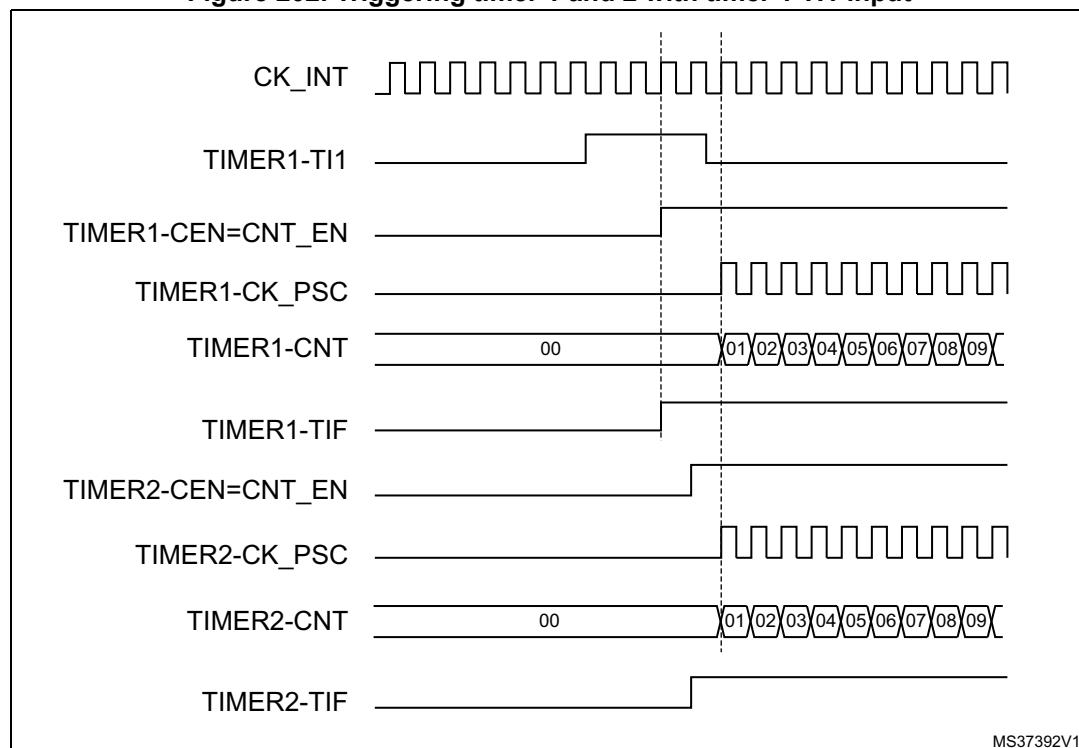
- Configure Timer 1 master mode to send its Enable as trigger output (MMS=001 in the TIM1\_CR2 register).
- Configure Timer 1 slave mode to get the input trigger from TI1 (TS=100 in the TIM1\_SMCR register).
- Configure Timer 1 in trigger mode (SMS=110 in the TIM1\_SMCR register).
- Configure the Timer 1 in Master/Slave mode by writing MSM=1 (TIM1\_SMCR register).
- Configure Timer 2 to get the input trigger from Timer 1 (TS=000 in the TIM2\_SMCR register).
- Configure Timer 2 in trigger mode (SMS=110 in the TIM2\_SMCR register).

When a rising edge occurs on TI1 (Timer 1), both counters starts counting synchronously on the internal clock and both TIF flags are set.

#### Note:

*In this example both timers are initialized before starting (by setting their respective UG bits). Both counters starts from 0, but you can easily insert an offset between them by writing any of the counter registers (TIMx\_CNT). You can see that the master/slave mode insert a delay between CNT\_EN and CK\_PSC on timer 1.*

**Figure 202. Triggering timer 1 and 2 with timer 1 TI1 input**



### 17.3.16 Debug mode

When the microcontroller enters debug mode (Cortex®-M4 with FPU core - halted), the TIMx counter either continues to work normally or stops, depending on DBG\_TIMx\_STOP configuration bit inDBGMCU module. For more details, refer to [Section 33.16.2: Debug support for timers, watchdog, bxCAN and I2C](#).

## 17.4 TIM2 to TIM5 registers

Refer to [Section 1.1 on page 51](#) for a list of abbreviations used in register descriptions.

The 32-bit peripheral registers have to be written by words (32 bits). All other 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).

### 17.4.1 TIMx control register 1 (TIMx\_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:10 Reserved, must be kept at reset value.

Bits 9:8 **CKD**: Clock division

This bit-field indicates the division ratio between the timer clock (CK\_INT) frequency and sampling clock used by the digital filters (ETR, TIx),

- 00:  $t_{DTS} = t_{CK\_INT}$
- 01:  $t_{DTS} = 2 \times t_{CK\_INT}$
- 10:  $t_{DTS} = 4 \times t_{CK\_INT}$
- 11: Reserved

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

- 0: TIMx\_ARR register is not buffered
- 1: TIMx\_ARR register is buffered

Bits 6:5 **CMS**: 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.

### 17.4.2 TIMx 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.	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)

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)

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.

### 17.4.3 TIMx slave mode control register (TIMx\_SMCR)

Address offset: 0x08

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
ETP	ECE	ETPS[1:0]		ETF[3:0]				MSM	TS[2:0]				Res.	SMS[2:0]		
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW		RW	RW	RW

Bit 15 **ETP**: External trigger polarity

This bit selects whether ETR or  $\overline{ETR}$  is used for trigger operations

0: ETR is noninverted, active at high level or rising edge

1: ETR is inverted, active at low level or falling edge

Bit 14 **ECE**: External clock enable

This bit enables External clock mode 2.

0: External clock mode 2 disabled

1: External clock mode 2 enabled. The counter is clocked by any active edge on the ETRF signal.

1: Setting the ECE bit has the same effect as selecting external clock mode 1 with TRGI connected to ETRF (SMS=111 and TS=111).

2: It is possible to simultaneously use external clock mode 2 with the following slave modes: reset mode, gated mode and trigger mode. Nevertheless, TRGI must not be connected to ETRF in this case (TS bits must not be 111).

3: If external clock mode 1 and external clock mode 2 are enabled at the same time, the external clock input is ETRF.

Bits 13:12 **ETPS**: 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 112](#) for more details on ITRx meaning for each Timer.*Note: These bits must be changed only when they are not used (e.g. when SMS=000) to avoid wrong edge detections at the transition.*

Bit 3 Reserved, must be kept at reset value.

Bits 2:0 **SMS:** Slave mode selection

When external signals are selected the active edge of the trigger signal (TRGI) is linked to the polarity selected on the external input (see Input Control register and Control Register description).

000: Slave mode disabled - if CEN = '1 then the prescaler is clocked directly by the internal clock.

001: Encoder mode 1 - Counter counts up/down on TI2FP2 edge depending on TI1FP1 level.

010: Encoder mode 2 - Counter counts up/down on TI1FP1 edge depending on TI2FP2 level.

011: Encoder mode 3 - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input.

100: Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter and generates an update of the registers.

101: Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high. The counter stops (but is not reset) as soon as the trigger becomes low. Both start and stop of the counter are controlled.

110: Trigger Mode - The counter starts at a rising edge of the trigger TRGI (but it is not reset). Only the start of the counter is controlled.

111: External Clock Mode 1 - Rising edges of the selected trigger (TRGI) clock the counter.

*Note: The gated mode must not be used if TI1F\_ED is selected as the trigger input (TS=100). Indeed, TI1F\_ED outputs 1 pulse for each transition on TI1F, whereas the gated mode checks the level of the trigger signal.***Table 112. TIMx internal trigger connections**

Slave TIM	ITR0 (TS = 000)	ITR1 (TS = 001)	ITR2 (TS = 010)	ITR3 (TS = 011)
TIM2	TIM1	TIM8	TIM3	TIM4
TIM3	TIM1	TIM2	TIM5	TIM4
TIM4	TIM1	TIM2	TIM3	TIM8
TIM5	TIM2	TIM3	TIM4	TIM8

#### 17.4.4 TIMx DMA/Interrupt enable register (TIMx\_DIER)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res.	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, always read as 0

Bit 12 **CC4DE**: Capture/Compare 4 DMA request enable

- 0: CC4 DMA request disabled.
- 1: CC4 DMA request enabled.

Bit 11 **CC3DE**: Capture/Compare 3 DMA request enable

- 0: CC3 DMA request disabled.
- 1: CC3 DMA request enabled.

Bit 10 **CC2DE**: Capture/Compare 2 DMA request enable

- 0: CC2 DMA request disabled.
- 1: CC2 DMA request enabled.

Bit 9 **CC1DE**: Capture/Compare 1 DMA request enable

- 0: CC1 DMA request disabled.
- 1: CC1 DMA request enabled.

Bit 8 **UDE**: Update DMA request enable

- 0: Update DMA request disabled.
- 1: Update DMA request enabled.

Bit 7 Reserved, 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

#### 17.4.5 TIMx 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.	Res.	Res.	CC4OF	CC3OF	CC2OF	CC1OF	Res.	Res.	TIF	Res.	CC4IF	CC3IF	CC2IF	CC1IF	UIF
			rc_w0	rc_w0	rc_w0	rc_w0			rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

Bits 15:13 Reserved, must be kept at reset value.

Bit 12 **CC4OF**: Capture/Compare 4 overcapture flag  
refer to CC1OF description

Bit 11 **CC3OF**: Capture/Compare 3 overcapture flag  
refer to CC1OF description

Bit 10 **CC2OF**: Capture/compare 2 overcapture flag  
refer to CC1OF description

Bit 9 **CC1OF**: Capture/Compare 1 overcapture flag  
This flag is set by hardware only when the corresponding channel is configured in input capture mode. It is cleared by software by writing it to '0'.  
0: No overcapture has been detected  
1: The counter value has been captured in TIMx\_CCR1 register while CC1IF flag was already set

Bits 8:7 Reserved, must be kept at reset value.

Bit 6 **TIF**: Trigger interrupt flag

This flag is set by hardware on trigger event (active edge detected on TRGI input when the slave mode controller is enabled in all modes but gated mode. It is set when the counter starts or stops when gated mode is selected. It is cleared by software.

- 0: No trigger event occurred
- 1: Trigger interrupt pending

Bit 5 Reserved, must be kept at reset value.

Bit 4 **CC4IF**: Capture/Compare 4 interrupt flag  
refer to CC1IF description

Bit 3 **CC3IF**: Capture/Compare 3 interrupt flag  
refer to CC1IF description

Bit 2 **CC2IF**: Capture/Compare 2 interrupt flag  
refer to CC1IF description

Bit 1 **CC1IF**: Capture/compare 1 interrupt flag

**If channel CC1 is configured as output:**

This flag is set by hardware when the counter matches the compare value, with some exception in center-aligned mode (refer to the CMS bits in the TIMx\_CR1 register description). It is cleared by software.

0: No match

1: The content of the counter TIMx\_CNT matches the content of the TIMx\_CCR1 register. When the contents of TIMx\_CCR1 are greater than the contents of TIMx\_ARR, the CC1IF bit goes high on the counter overflow (in upcounting and up/down-counting modes) or underflow (in downcounting mode)

**If channel CC1 is configured as input:**

This bit is set by hardware on a capture. It is cleared by software or by reading the TIMx\_CCR1 register.

0: No input capture occurred

1: The counter value has been captured in TIMx\_CCR1 register (An edge has been detected on IC1 which matches the selected polarity)

Bit 0 **UIF**: Update interrupt flag

“ This bit is set by hardware on an update event. It is cleared by software.  
0: No update occurred.

1: Update interrupt pending. This bit is set by hardware when the registers are updated:

“ At overflow or underflow (for TIM2 to TIM5) 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.

### 17.4.6 TIMx 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.	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).

### 17.4.7 TIMx capture/compare mode register 1 (TIMx\_CCMR1)

Address offset: 0x18

Reset value: 0x0000

The channels can be used in input (capture mode) or in output (compare mode). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OCxx describes its function when the channel is configured in output, ICxx describes its function when the channel is configured in input. So you must take care that the same bit can have a different meaning for the input stage and for the output stage.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]		
	IC2F[3:0]				IC2PSC[1:0]			IC1F[3:0]				IC1PSC[1:0]			
rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw

#### Output compare mode

Bit 15 **OC2CE**: Output compare 2 clear enable

Bits 14:12 **OC2M[2:0]**: Output compare 2 mode

Bit 11 **OC2PE**: Output compare 2 preload enable

Bit 10 **OC2FE**: Output compare 2 fast enable

Bits 9:8 **CC2S[1:0]**: Capture/Compare 2 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output

01: CC2 channel is configured as input, IC2 is mapped on TI2

10: CC2 channel is configured as input, IC2 is mapped on TI1

11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode is working only if an internal trigger input is selected through the TS bit (TIMx\_SMCR register)

Note: CC2S bits are writable only when the channel is OFF (CC2E = 0 in TIMx\_CCER).

Bit 7 **OC1CE**: Output compare 1 clear enable

OC1CE: Output Compare 1 Clear Enable

0: OC1Ref is not affected by the ETRF input

1: OC1Ref is cleared as soon as a High level is detected on ETRF input

**Bits 6:4 OC1M:** Output compare 1 mode

These bits define the behavior of the output reference signal OC1REF from which OC1 and OC1N are derived. OC1REF is active high whereas OC1 and OC1N active level depends on CC1P and CC1NP bits.

000: Frozen - The comparison between the output compare register TIMx\_CCR1 and the counter TIMx\_CNT has no effect on the outputs.(this mode is used to generate a timing base).

001: Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

011: Toggle - OC1REF toggles when TIMx\_CNT=TIMx\_CCR1.

100: Force inactive level - OC1REF is forced low.

101: Force active level - OC1REF is forced high.

110: PWM mode 1 - In upcounting, channel 1 is active as long as TIMx\_CNT<TIMx\_CCR1 else inactive. In downcounting, channel 1 is inactive (OC1REF='0) as long as TIMx\_CNT>TIMx\_CCR1 else active (OC1REF=1).

111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIMx\_CNT<TIMx\_CCR1 else active. In downcounting, channel 1 is active as long as TIMx\_CNT>TIMx\_CCR1 else inactive.

*Note: In PWM mode 1 or 2, the OCREF level changes only when the result of the comparison changes or when the output compare mode switches from “frozen” mode to “PWM” mode.*

**Bit 3 OC1PE:** Output compare 1 preload enable

0: Preload register on TIMx\_CCR1 disabled. TIMx\_CCR1 can be written at anytime, the new value is taken in account immediately.

1: Preload register on TIMx\_CCR1 enabled. Read/Write operations access the preload register. TIMx\_CCR1 preload value is loaded in the active register at each update event.

*Note: 1: These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx\_BDTR register) and CC1S=00 (the channel is configured in output).*

*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 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).*

### 17.4.8 TIMx capture/compare mode register 2 (TIMx\_CCMR2)

Address offset: 0x1C

Reset value: 0x0000

Refer to the above CCMR1 register description.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
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

Bit 15 **OC4CE**: Output compare 4 clear enable

Bits 14:12 **OC4M**: Output compare 4 mode

Bit 11 **OC4PE**: Output compare 4 preload enable

Bit 10 **OC4FE**: Output compare 4 fast enable

Bits 9:8 **CC4S**: Capture/Compare 4 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC4 channel is configured as output

01: CC4 channel is configured as input, IC4 is mapped on TI4

10: CC4 channel is configured as input, IC4 is mapped on TI3

11: CC4 channel is configured as input, IC4 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC4S bits are writable only when the channel is OFF (CC4E = 0 in TIMx\_CCER).*

Bit 7 **OC3CE**: Output compare 3 clear enable

Bits 6:4 **OC3M**: Output compare 3 mode

Bit 3 **OC3PE**: Output compare 3 preload enable

Bit 2 **OC3FE**: Output compare 3 fast enable

Bits 1:0 **CC3S**: Capture/Compare 3 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC3 channel is configured as output

01: CC3 channel is configured as input, IC3 is mapped on TI3

10: CC3 channel is configured as input, IC3 is mapped on TI4

11: CC3 channel is configured as input, IC3 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC3S bits are writable only when the channel is OFF (CC3E = 0 in TIMx\_CCER).*

### Input capture mode

Bits 15:12 **IC4F**: Input capture 4 filter

Bits 11:10 **IC4PSC**: Input capture 4 prescaler

Bits 9:8 **CC4S**: Capture/Compare 4 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC4 channel is configured as output

01: CC4 channel is configured as input, IC4 is mapped on TI4

10: CC4 channel is configured as input, IC4 is mapped on TI3

11: CC4 channel is configured as input, IC4 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC4S bits are writable only when the channel is OFF (CC4E = 0 in TIMx\_CCER).*

Bits 7:4 **IC3F**: Input capture 3 filter

Bits 3:2 **IC3PSC**: Input capture 3 prescaler

Bits 1:0 **CC3S**: Capture/Compare 3 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC3 channel is configured as output

01: CC3 channel is configured as input, IC3 is mapped on TI3

10: CC3 channel is configured as input, IC3 is mapped on TI4

11: CC3 channel is configured as input, IC3 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx\_SMCR register)

*Note: CC3S bits are writable only when the channel is OFF (CC3E = 0 in TIMx\_CCER).*

### 17.4.9 TIMx capture/compare enable register (TIMx\_CCER)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
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 113. Output control bit for standard OCx channels**

CCxE bit	OCx output state
0	Output Disabled (OCx=0, OCx_EN=0)
1	OCx=OCxREF + Polarity, OCx_EN=1

**Note:** The state of the external IO pins connected to the standard OCx channels depends on the OCx channel state and the GPIO registers.

### 17.4.10 TIMx counter (TIMx\_CNT)

Address offset: 0x24

Reset value: 0x0000

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

Bits 15:0 **CNT[15:0]**: Counter value

### 17.4.11 TIMx prescaler (TIMx\_PSC)

Address offset: 0x28

Reset value: 0x0000

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

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK\_CNT is equal to  $f_{CK\_PSC} / (PSC[15:0] + 1)$ .

PSC contains the value to be loaded in the active prescaler register at each update event.

### 17.4.12 TIMx auto-reload register (TIMx\_ARR)

Address offset: 0x2C

Reset value: 0x0000

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

Bits 15:0 **ARR[15:0]**: Auto-reload value

ARR is the value to be loaded in the actual auto-reload register.

Refer to the [Section 17.3.1: Time-base unit on page 521](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

### 17.4.13 TIMx capture/compare register 1 (TIMx\_CCR1)

Address offset: 0x34

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR1[31:16] (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).

### 17.4.14 TIMx capture/compare register 2 (TIMx\_CCR2)

Address offset: 0x38

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR2[31:16] (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\_CCMR register (bit OC2PE). Else the preload value is copied in the active capture/compare 2 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signalled on OC2 output.

**If channel CC2 is configured as input:**

CCR2 is the counter value transferred by the last input capture 2 event (IC2).

### 17.4.15 TIMx capture/compare register 3 (TIMx\_CCR3)

Address offset: 0x3C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR3[31:16] (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\_CCMR register (bit OC3PE). Else the preload value is copied in the active capture/compare 3 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signaled on OC3 output.

**If channel CC3 is configured as input:**

CCR3 is the counter value transferred by the last input capture 3 event (IC3).

### 17.4.16 TIMx capture/compare register 4 (TIMx\_CCR4)

Address offset: 0x40

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR4[31:16] (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\_CCMR 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).

### 17.4.17 TIMx DMA control register (TIMx\_DCR)

Address offset: 0x48

Reset value: 0x0000

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

Bits 15:13 Reserved, must be kept at reset value.

Bits 12:8 **DBL[4:0]**: DMA burst length

This 5-bit vector defines the 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.

### 17.4.18 TIMx DMA address for full transfer (TIMx\_DMAR)

Address offset: 0x4C

Reset value: 0x0000

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

Bits 15:0 **DMAB[15:0]**: DMA register for burst accesses

A read or write operation to the DMAR register accesses the register located at the address (TIMx\_CR1 address) + (DBA + DMA index) × 4

where TIMx\_CR1 address is the address of the control register 1, DBA is the DMA base address configured in TIMx\_DCR register, DMA index is automatically controlled by the DMA transfer, and ranges from 0 to DBL (DBL configured in TIMx\_DCR).

#### Example of how to use the DMA burst feature

In this example the timer DMA burst feature is used to update the contents of the CCRx registers ( $x = 2, 3, 4$ ) with the DMA transferring half words into the CCRx registers.

This is done in the following steps:

1. Configure the corresponding DMA channel as follows:
  - DMA channel peripheral address is the DMAR register address
  - DMA channel memory address is the address of the buffer in the RAM containing the data to be transferred by DMA into CCRx registers.
  - Number of data to transfer = 3 (See note below).
  - Circular mode disabled.
2. Configure the DCR register by configuring the DBA and DBL bit fields as follows:  
DBL = 3 transfers, DBA = 0xE.
3. Enable the TIMx update DMA request (set the UDE bit in the DIER register).
4. Enable TIMx
5. Enable the DMA channel

**Note:**

*This example is for the case where every CCRx register to be updated once. If every CCRx register is to be updated twice for example, the number of data to transfer should be 6. Let's take the example of a buffer in the RAM containing data1, data2, data3, data4, data5 and data6. The data is transferred to the CCRx registers as follows: on the first update DMA request, data1 is transferred to CCR2, data2 is transferred to CCR3, data3 is transferred to CCR4 and on the second update DMA request, data4 is transferred to CCR2, data5 is transferred to CCR3 and data6 is transferred to CCR4.*

#### 17.4.19 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		Res.									
				rw	rw										

Bits 15:12 Reserved, must be kept at reset value.

Bits 11:10 **ITR1\_RMP**: 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.

### 17.4.20 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		Res.	Res.	Res.	Res.	Res.	Res.							
								rw	rw						

Bits 15:8 Reserved, must be kept at reset value.

Bits 7:6 **TI4\_RMP**: 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 TIM5\_CH4 input for calibration purposes.

Wakeup interrupt should be enabled.

Bits 5:0 Reserved, must be kept at reset value.

### 17.4.21 TIMx register map

TIMx registers are mapped as described in the table below:

**Table 114. TIM2 to TIM5 register map and reset values**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	<b>TIMx_CR1</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	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.	Res.	Res.	Res.	Res.			
0x04	<b>TIMx_CR2</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	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.	Res.	Res.	Res.	Res.			
0x08	<b>TIMx_SMCR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	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.	Res.	Res.	Res.	Res.			
0x0C	<b>TIMx_DIER</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	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.	Res.	Res.	Res.	Res.			
0x10	<b>TIMx_SR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	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.	Res.	Res.	Res.	Res.			
0x14	<b>TIMx_EGR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
	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.	Res.	Res.	Res.	Res.			
0x18	<b>TIMx_CCMR1</b> 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.			
	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.	Res.	Res.	Res.	Res.			
	<b>TIMx_CCMR1</b> 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.			
	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.	Res.	Res.	Res.	Res.			
0x1C	<b>TIMx_CCMR2</b> 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.			
	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.	Res.	Res.	Res.	Res.			
	<b>TIMx_CCMR2</b> 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.			
	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.	Res.	Res.	Res.	Res.			
0x20	<b>TIMx_CCER</b>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
	Reset value	Res.	Res.	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			
0x24	<b>TIMx_CNT</b>	CNT[31:16] (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	0	0			

Table 114. TIM2 to TIM5 register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x28	<b>TIMx_PSC</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PSC[15:0]			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x2C	<b>TIMx_ARR</b>	ARR[31:16] (TIM2 and TIM5 only, reserved on the other timers)												ARR[15:0]																			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
0x30	Reserved	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.				
0x34	<b>TIMx_CCR1</b>	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	0				
0x38	<b>TIMx_CCR2</b>	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	0				
0x3C	<b>TIMx_CCR3</b>	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	0				
0x40	<b>TIMx_CCR4</b>	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	0				
0x44	Reserved	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.					
0x48	<b>TIMx_DCR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DBL[4:0]				
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
0x4C	<b>TIMx_DMAR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DMAB[15:0]				
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
0x50	<b>TIM2_OR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ITR1_RMP				
	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				
0x50	<b>TIM5_OR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	IT4_RMP				
	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				

Refer to [Section 2.2.2: Memory map and register boundary addresses](#) for the register boundary addresses.

## 18 General-purpose timers (TIM9 to TIM14)

### 18.1 TIM9 to TIM14 introduction

The TIM9 to TIM14 general-purpose timers consist of a 16-bit auto-reload counter driven by a programmable prescaler.

They may be used for a variety of purposes, including measuring the pulse lengths of input signals (input capture) or generating output waveforms (output compare, 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 to TIM14 timers are completely independent, and do not share any resources. They can be synchronized together as described in [Section 18.3.12](#).

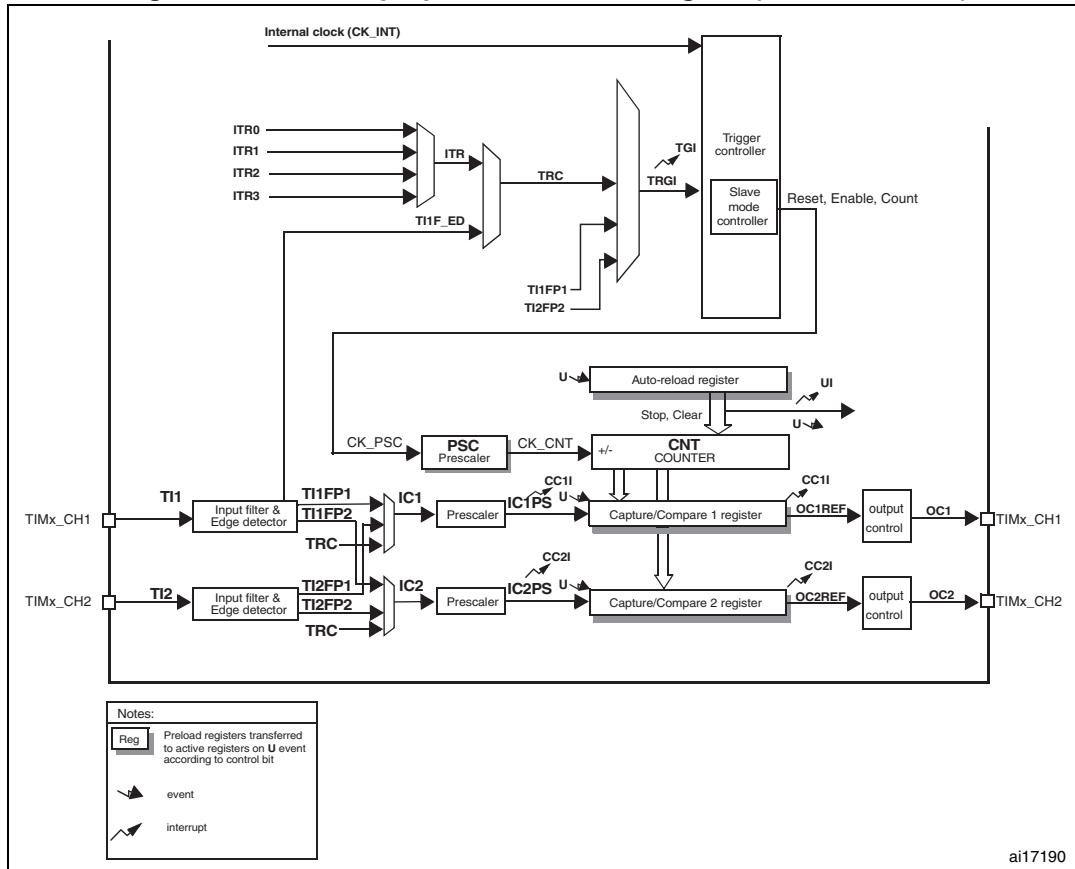
### 18.2 TIM9 to TIM14 main features

#### 18.2.1 TIM9/TIM12 main features

The features of the TIM9 to TIM14 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 203. General-purpose timer block diagram (TIM9 and TIM12)

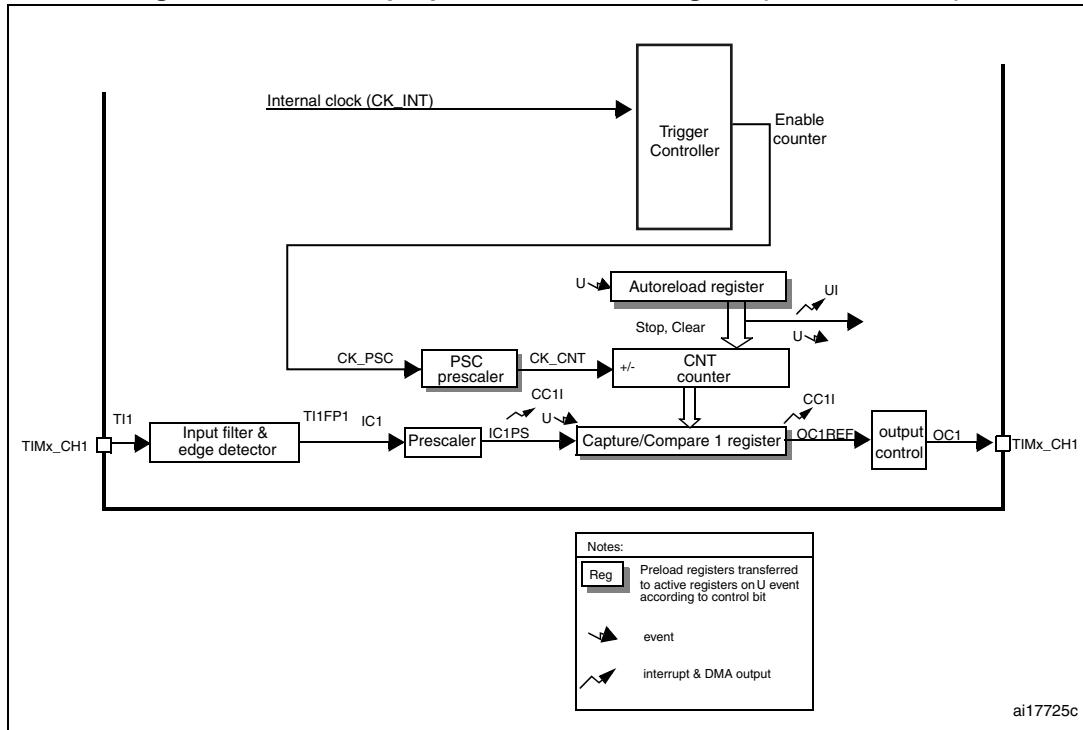


### 18.2.2 TIM10/TIM11 and TIM13/TIM14 main features

The features of general-purpose timers TIM10/TIM11 and 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 204. General-purpose timer block diagram (TIM10/11/13/14)



## 18.3 TIM9 to TIM14 functional description

### 18.3.1 Time-base unit

The main block of the timer is a 16-bit counter with its related auto-reload register. The counter counts up.

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.

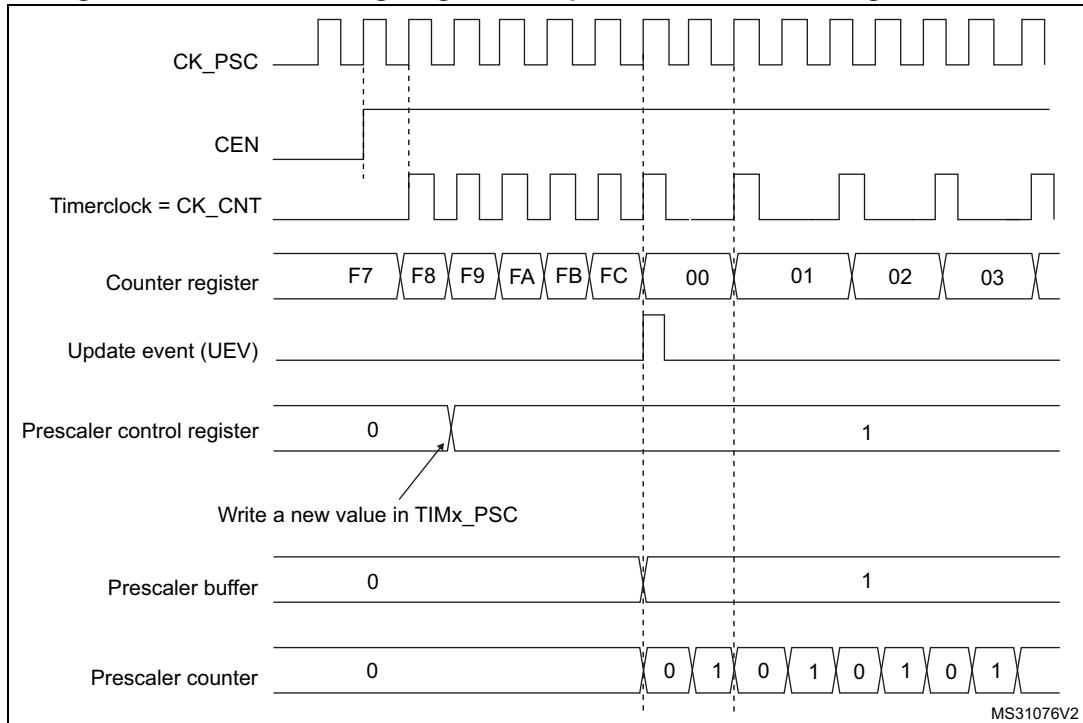
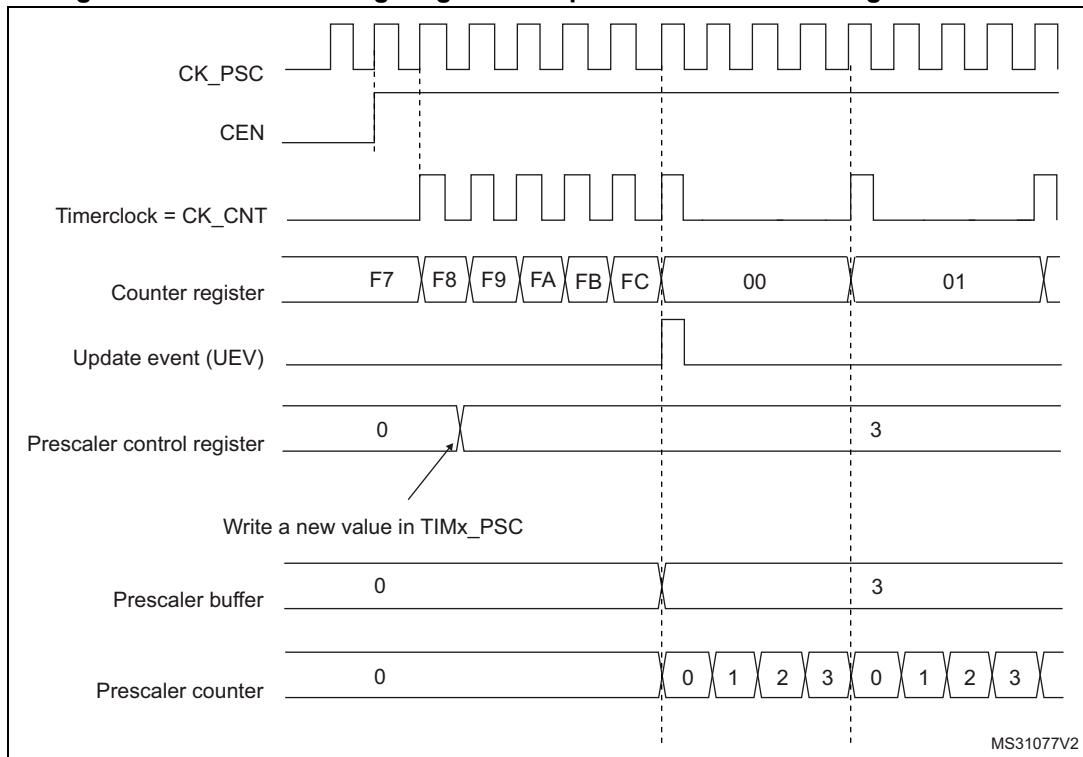
The counter is clocked by the prescaler output CK\_CNT, which is enabled only when the counter enable bit (CEN) in TIMx\_CR1 register is set (refer also to the slave mode controller description to get more details on counter enabling).

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 205* and *Figure 206* give some examples of the counter behavior when the prescaler ratio is changed on the fly.

**Figure 205. Counter timing diagram with prescaler division change from 1 to 2****Figure 206. 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.

Setting the UG bit in the TIMx\_EGR register (by software or by using the slave mode controller on TIM9 and TIM12) also generates an update event.

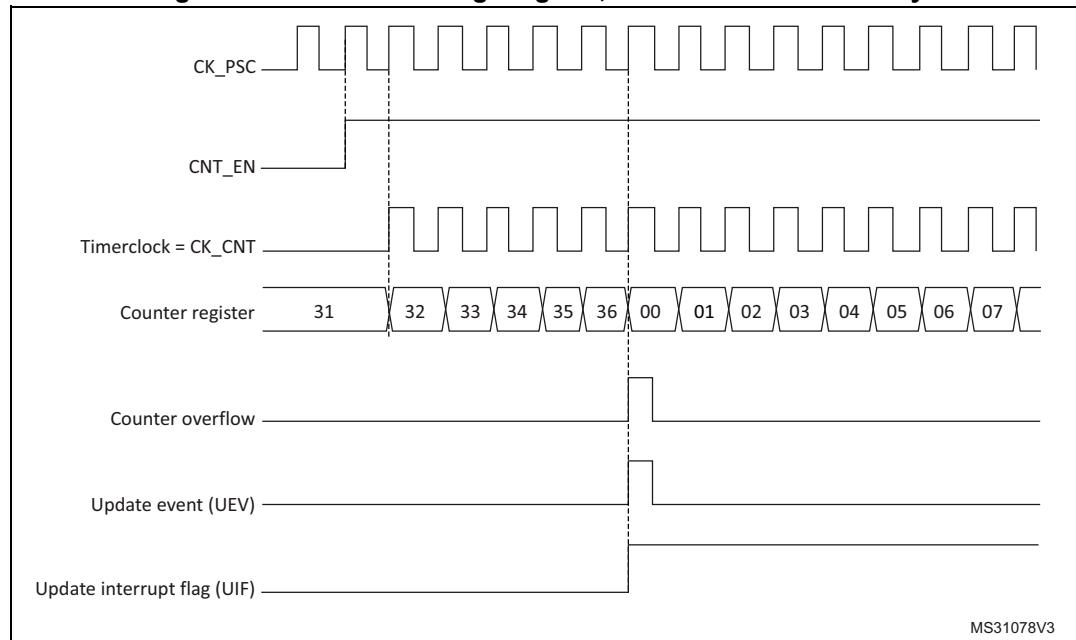
The UEV event can be disabled by software by setting the UDIS bit in the TIMx\_CR1 register. This is to avoid updating the shadow registers while writing new values in the preload registers. Then no update event occurs until the UDIS bit has been written to 0. However, the counter restarts from 0, as 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.

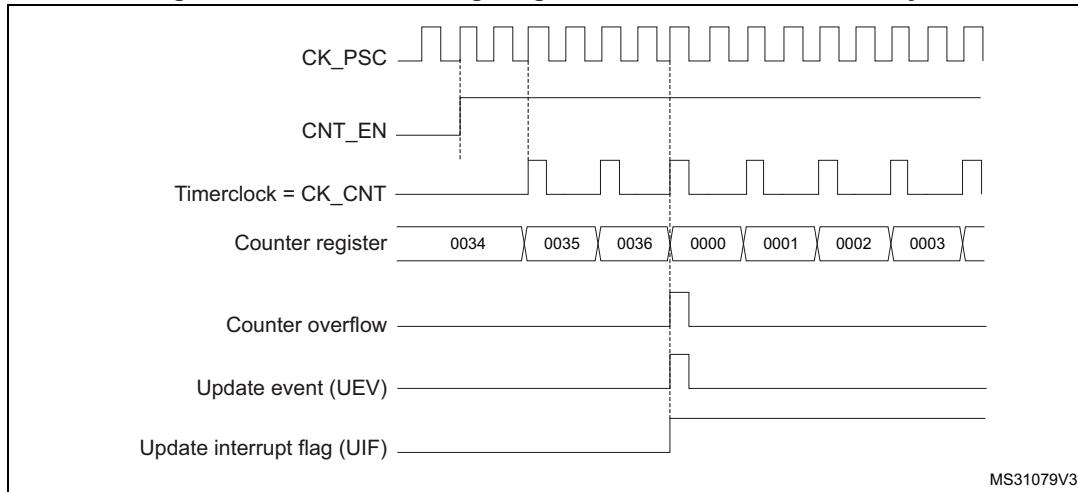
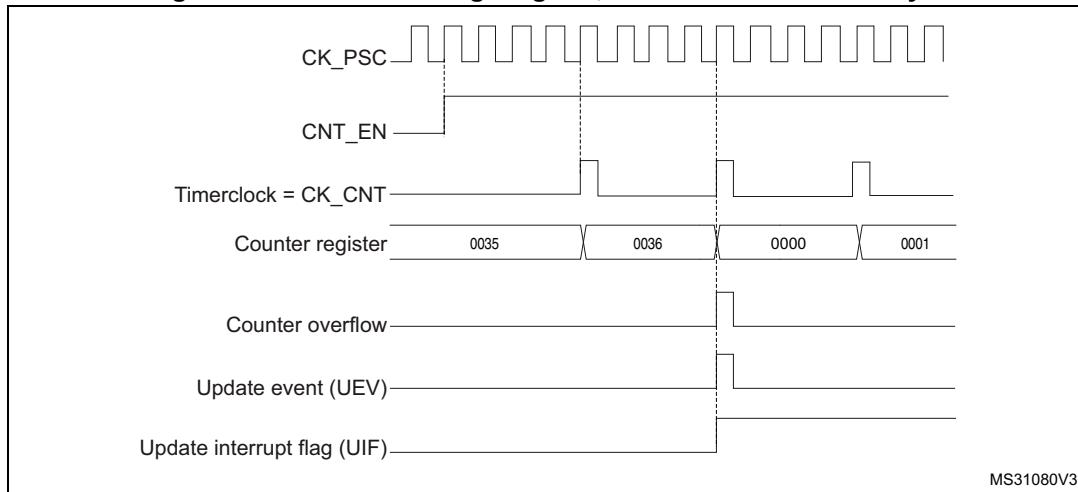
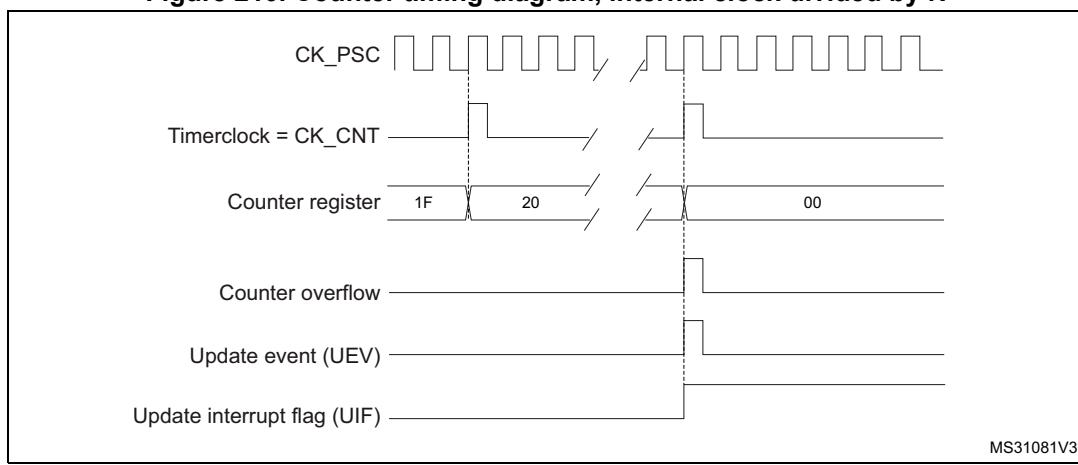
When an update event occurs, all the registers are updated and the update flag (UIF bit in TIMx\_SR register) is set (depending on the URS bit):

- 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).

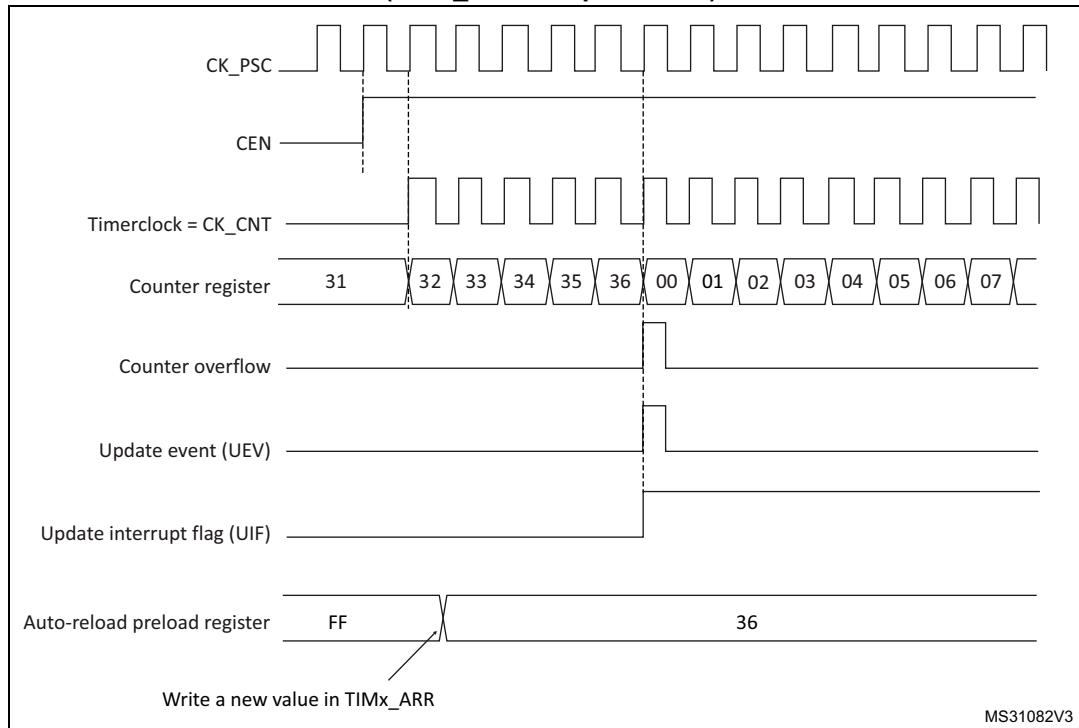
The following figures show some examples of the counter behavior for different clock frequencies when TIMx\_ARR=0x36.

**Figure 207. Counter timing diagram, internal clock divided by 1**

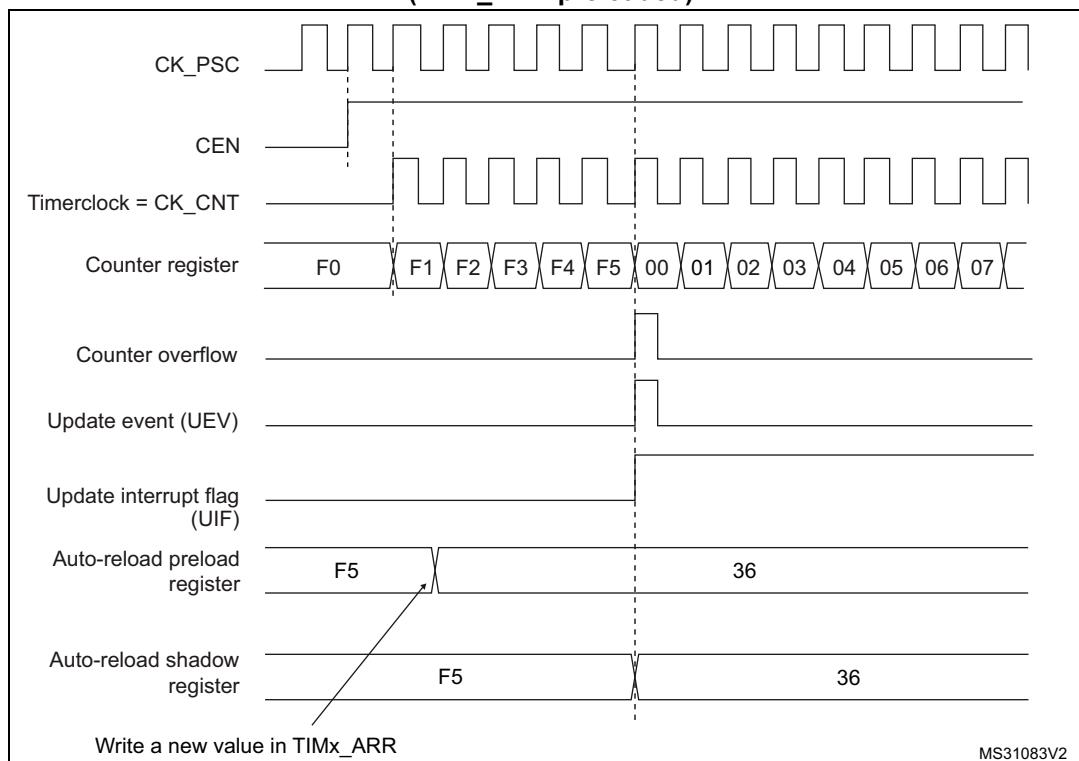


**Figure 208. Counter timing diagram, internal clock divided by 2****Figure 209. Counter timing diagram, internal clock divided by 4****Figure 210. Counter timing diagram, internal clock divided by N**

**Figure 211. Counter timing diagram, update event when ARPE=0  
(TIMx\_ARR not preloaded)**



**Figure 212. Counter timing diagram, update event when ARPE=1  
(TIMx\_ARR preloaded)**



### 18.3.3 Clock selection

The counter clock can be provided by the following clock sources:

- Internal clock (CK\_INT)
- External clock mode1 (for **TIM9** and **TIM12**): external input pin (TIx)
- Internal trigger inputs (ITRx) (for **TIM9** and **TIM12**): connecting the trigger output from another timer. Refer to [Using one timer as prescaler for another](#) for more details.

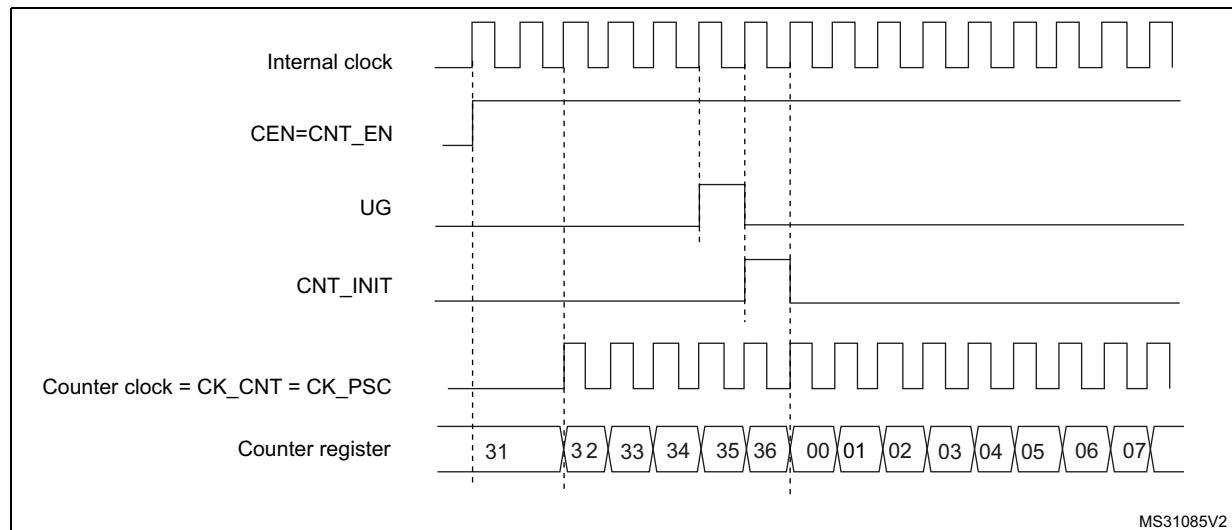
#### Internal clock source (CK\_INT)

The internal clock source is the default clock source for TIM10/TIM11 and TIM13/TIM14.

For TIM9 and 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 213](#) shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

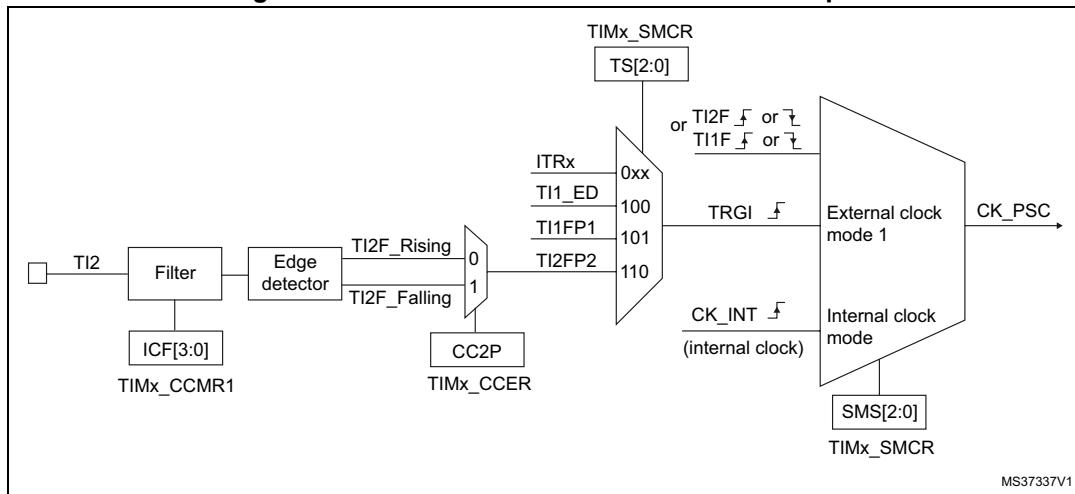
**Figure 213. Control circuit in normal mode, internal clock divided by 1**



#### External clock source mode 1(TIM9 and 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 214. 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.

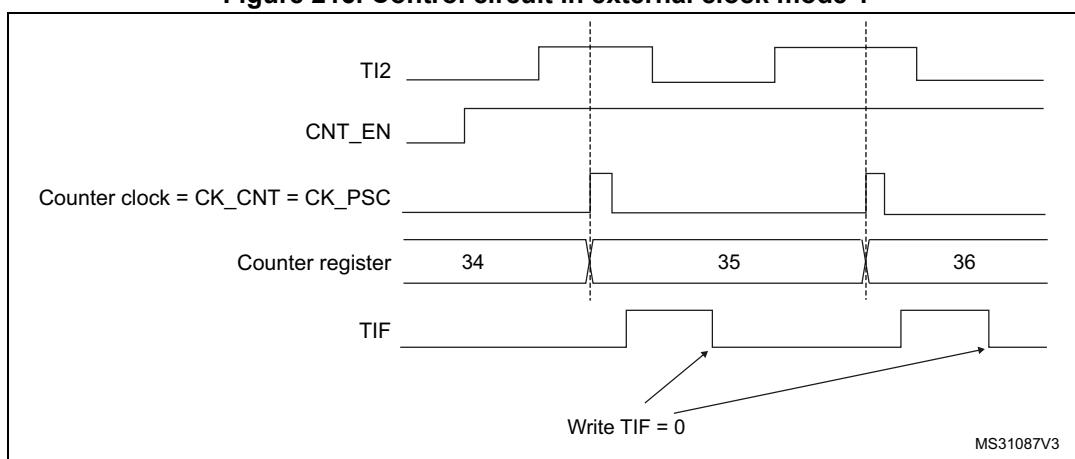
Note:

*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 215. Control circuit in external clock mode 1



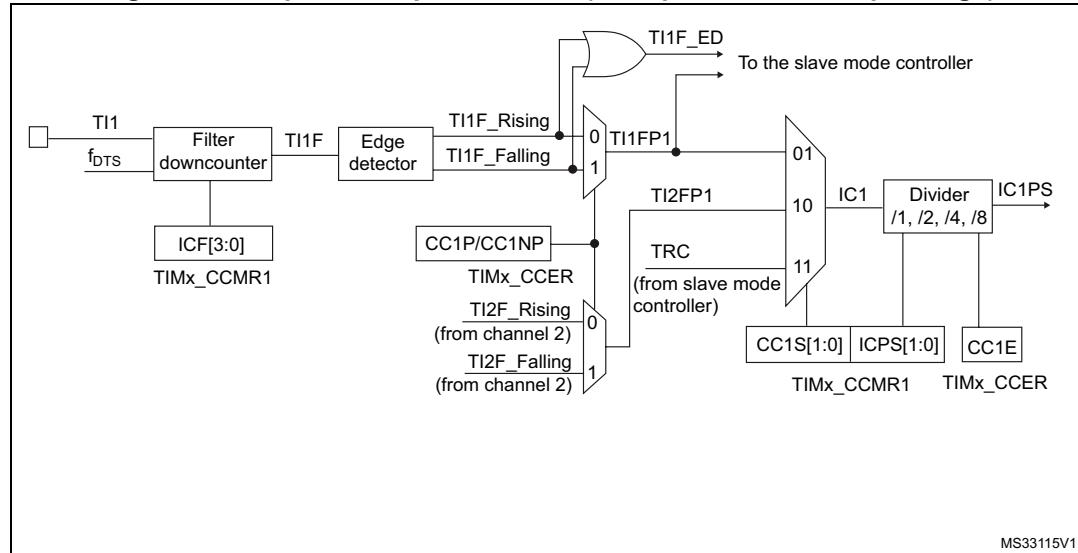
### 18.3.4 Capture/compare channels

Each Capture/Compare channel is built around a capture/compare register (including a shadow register), a input stage for capture (with digital filter, multiplexing and prescaler) and an output stage (with comparator and output control).

*Figure 216* to *Figure 218* give an overview of one capture/compare channel.

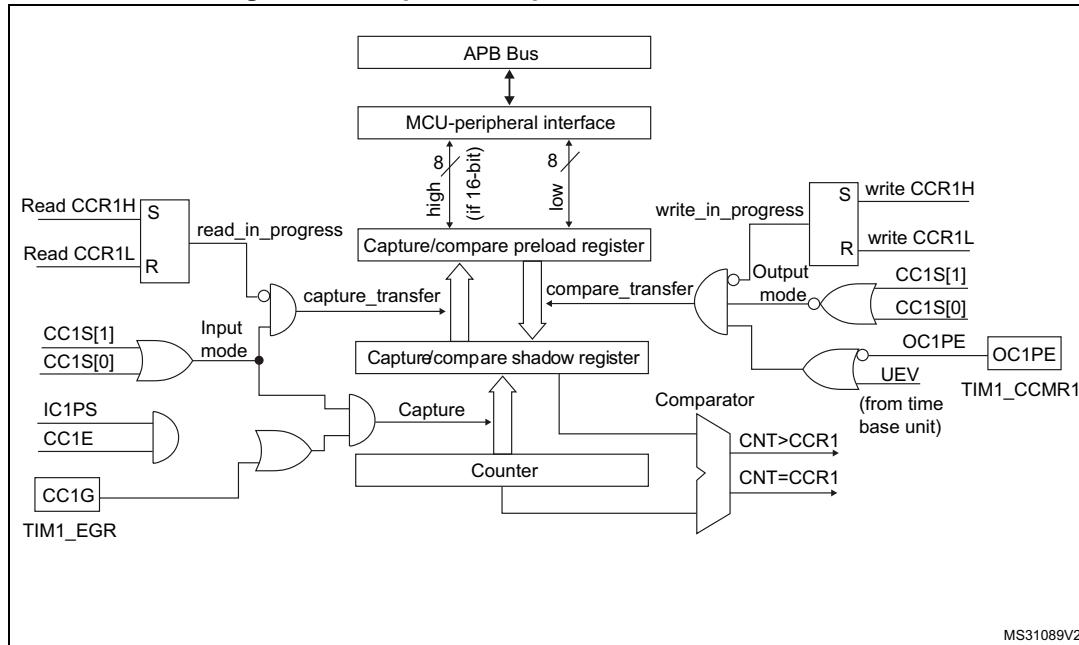
The input stage samples the corresponding TIx input to generate a filtered signal TIxF. Then, an edge detector with polarity selection generates a signal (TIxFPx) which can be used as trigger input by the slave mode controller or as the capture command. It is prescaled before the capture register (ICxPS).

**Figure 216. Capture/compare channel (example: channel 1 input stage)**



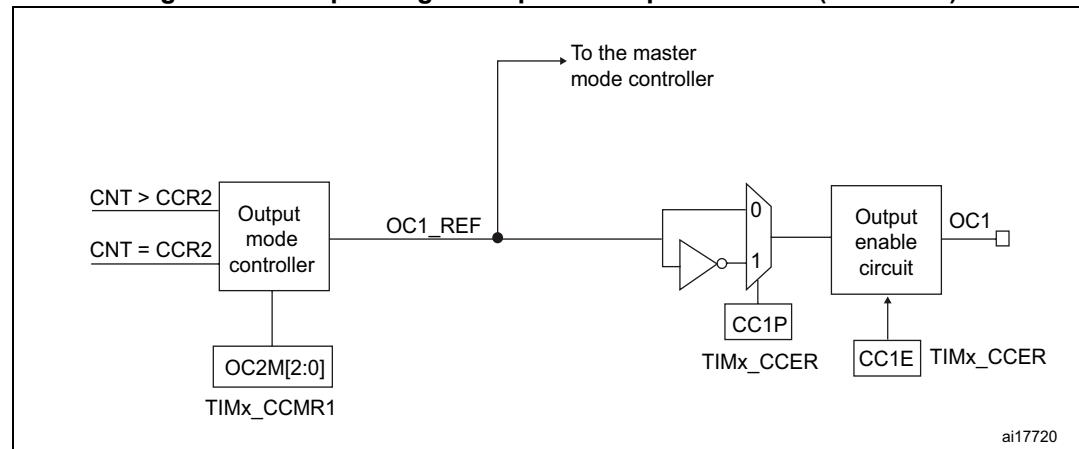
The output stage generates an intermediate waveform which is then used for reference: OCxRef (active high). The polarity acts at the end of the chain.

Figure 217. Capture/compare channel 1 main circuit



MS31089V2

Figure 218. Output stage of capture/compare channel (channel 1)



ai17720

The capture/compare block is made of one preload register and one shadow register. Write and read always access the preload register.

In capture mode, captures are actually done in the shadow register, which is copied into the preload register.

In compare mode, the content of the preload register is copied into the shadow register which is compared to the counter.

### 18.3.5 Input capture mode

In Input capture mode, the Capture/Compare Registers (TIMx\_CCRx) are used to latch the value of the counter after a transition detected by the corresponding IC<sub>x</sub> signal. When a capture occurs, the corresponding CC<sub>x</sub>IF flag (TIMx\_SR register) is set and an interrupt or a DMA request can be sent if they are enabled. If a capture occurs while the CC<sub>x</sub>IF flag was already high, then the over-capture flag CC<sub>x</sub>OF (TIMx\_SR register) is set. CC<sub>x</sub>IF can be

cleared by software by writing it to '0' or by reading the captured data stored in the TIMx\_CCRx register. 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.*

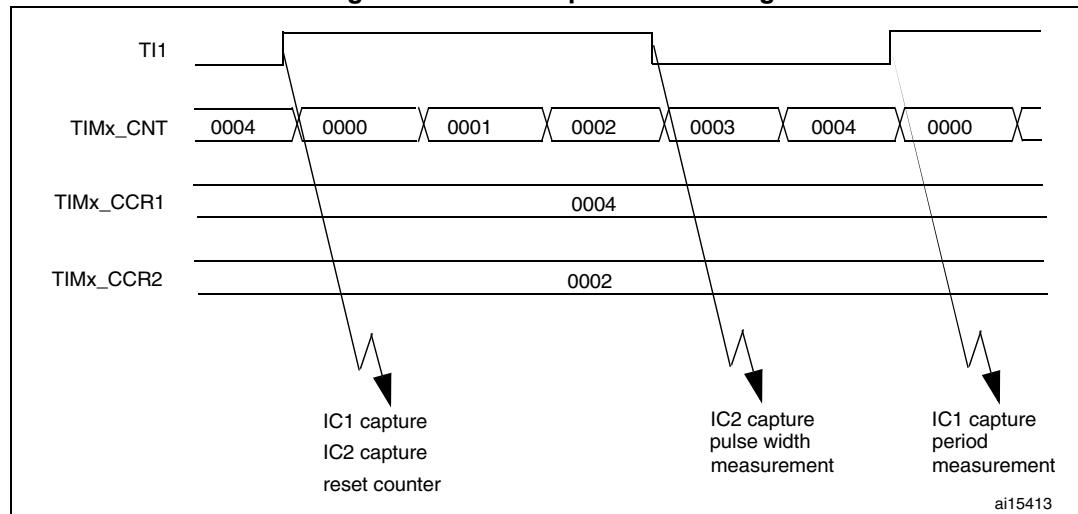
### 18.3.6 PWM input mode (only for TIM9/12)

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 219. 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.

### 18.3.7 Forced output mode

In output mode (CCxS bits = '00' in the TIMx\_CCMRx register), each output compare signal (OCxREF and then OCx) can be forced to active or inactive level directly by software, independently of any comparison between the output compare register and the counter.

To force an output compare signal (OCXREF/OCx) to its active level, 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.

For example: CCxP='0' (OCx active high) => OCx is forced to high level.

The OCxREF signal can be forced low by writing the OCxM bits to '100' in the TIMx\_CCMRx register.

Anyway, the comparison between the TIMx\_CCRx shadow register and the counter is still performed and allows the flag to be set. Interrupt requests can be sent accordingly. This is described in the output compare mode section below.

### 18.3.8 Output compare mode

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

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

1. Assigns the corresponding output pin to a programmable value defined by the output compare mode (OCxM bits in the TIMx\_CCMRx register) and the output polarity (CCxP bit in the TIMx\_CCER register). The output pin can keep its level (OCXM='000'), be set active (OCXM='001'), be set inactive (OCXM='010') or can toggle (OCXM='011') on match.
2. Sets a flag in the interrupt status register (CCxIF bit in the TIMx\_SR register).
3. Generates an interrupt if the corresponding interrupt mask is set (CCXIE bit in the TIMx\_DIER register).

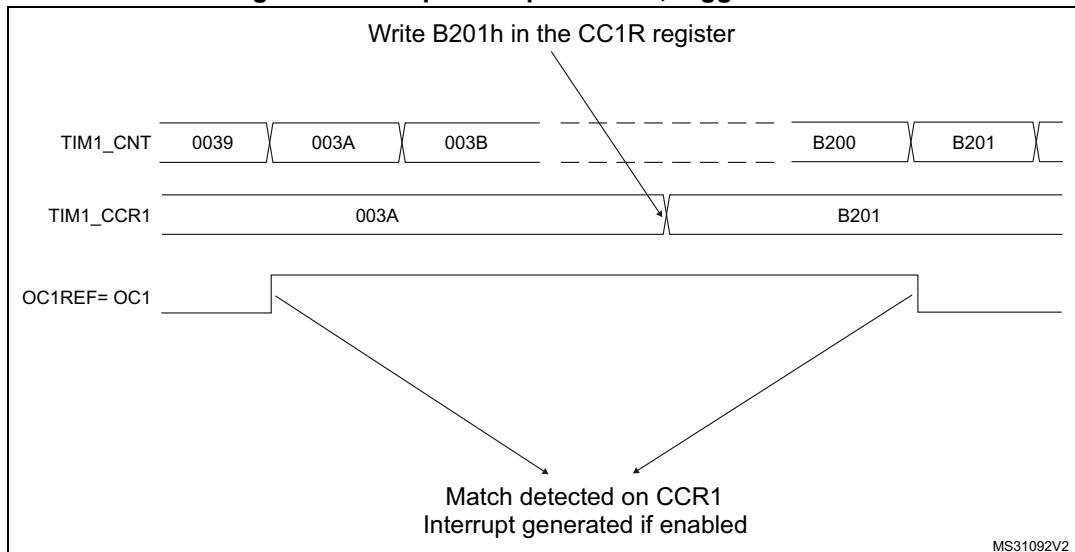
The TIMx\_CCRx registers can be programmed with or without preload registers using the OCxPE bit in the TIMx\_CCMRx register.

In output compare mode, the update event UEV has no effect on OCxREF and OCx output. The timing resolution is one count of the counter. Output compare mode can also be used to output a single pulse (in One-pulse mode).

Procedure:

1. Select the counter clock (internal, external, prescaler).
2. Write the desired data in the TIMx\_ARR and TIMx\_CCRx registers.
3. Set the CCxIE bit if an interrupt request is to be generated.
4. Select the output mode. For example:
  - Write OCxM = '011' to toggle OCx output pin when CNT matches CCRx
  - Write OCxPE = '0' to disable preload register
  - Write CCxP = '0' to select active high polarity
  - Write CCxE = '1' to enable the output
5. Enable the counter by setting the CEN bit in the TIMx\_CR1 register.

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 220](#).

**Figure 220. Output compare mode, toggle on OC1.**

### 18.3.9 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.

The OCx polarity is software programmable using the CCxP bit in the TIMx\_CCER register. It can be programmed as active high or active low. The OCx output is enabled by the CCxE bit in the TIMx\_CCER register. Refer to the TIMx\_CCERx register description for more details.

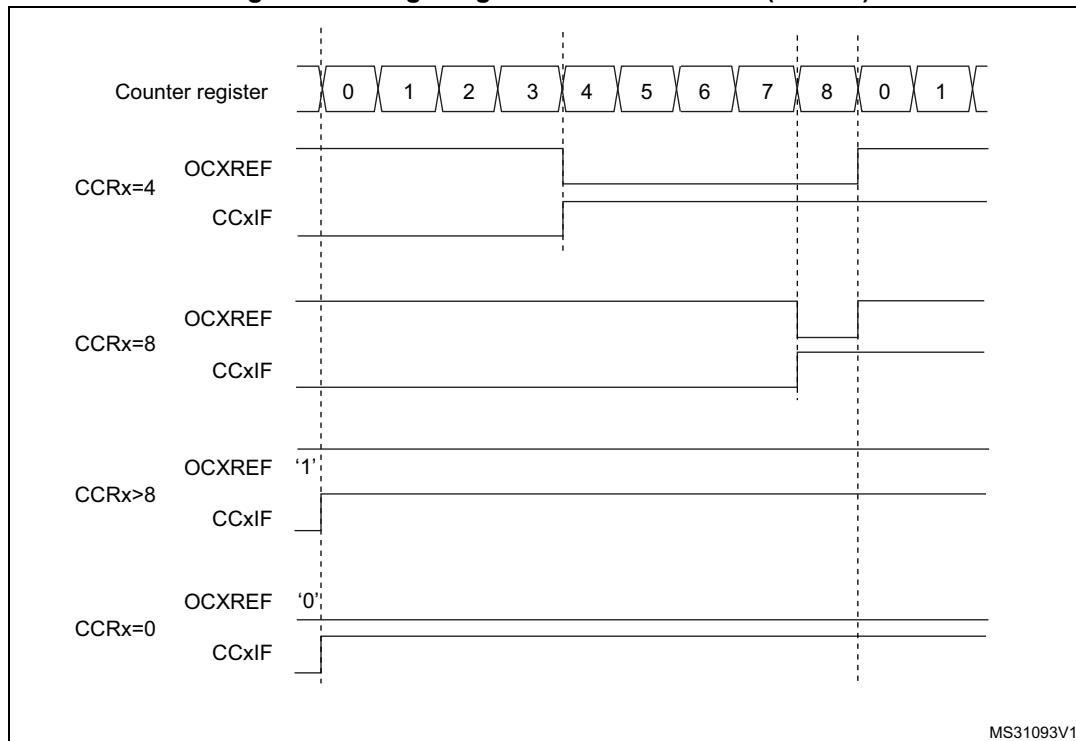
In PWM mode (1 or 2), TIMx\_CNT and TIMx\_CCRx are always compared to determine whether  $\text{TIMx_CNT} \leq \text{TIMx_CCRx}$ .

The timer is able to generate PWM in edge-aligned mode only since the counter is upcounting.

#### PWM edge-aligned mode

In the following example, we consider PWM mode 1. The reference PWM signal OCxREF is high as long as  $\text{TIMx_CNT} < \text{TIMx_CCRx}$  else it becomes low. If the compare value in TIMx\_CCRx is greater than the auto-reload value (in TIMx\_ARR) then OCxREF is held at '1'. If the compare value is 0 then OCxRef is held at '0'. [Figure 221](#) shows some edge-aligned PWM waveforms in an example where TIMx\_ARR=8.

Figure 221. Edge-aligned PWM waveforms (ARR=8)



### 18.3.10 One-pulse mode

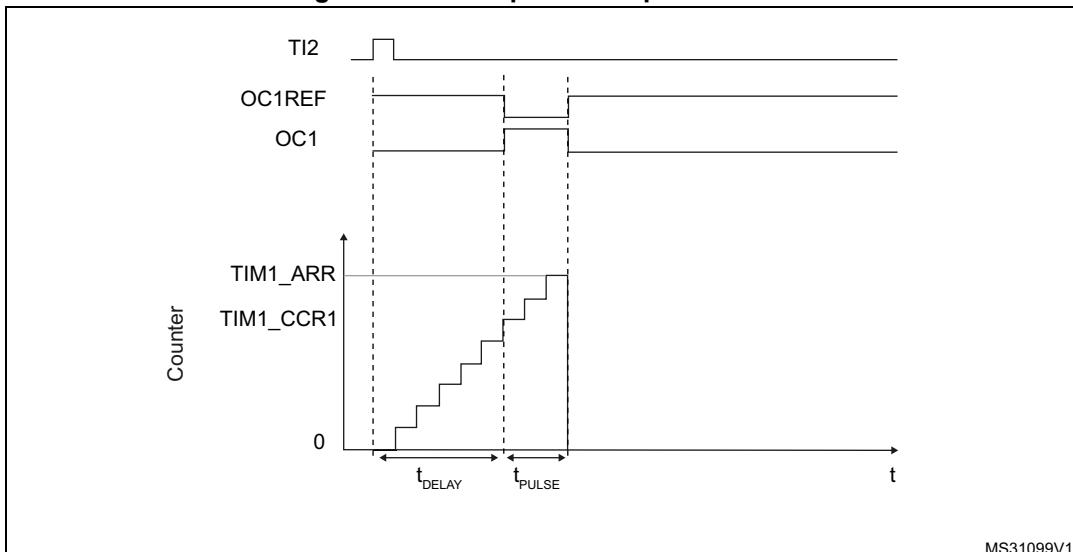
One-pulse mode (OPM) is a particular case of the previous modes. It allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length after a programmable delay.

Starting the counter can be controlled through the slave mode controller. Generating the waveform can be done in output compare mode or PWM mode. 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 222. 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.

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.

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.

### 18.3.11 TIM9/12 external trigger synchronization

The TIM9/12 timers can be synchronized with an external trigger in several modes: Reset mode, Gated mode and Trigger mode.

#### Slave mode: Reset mode

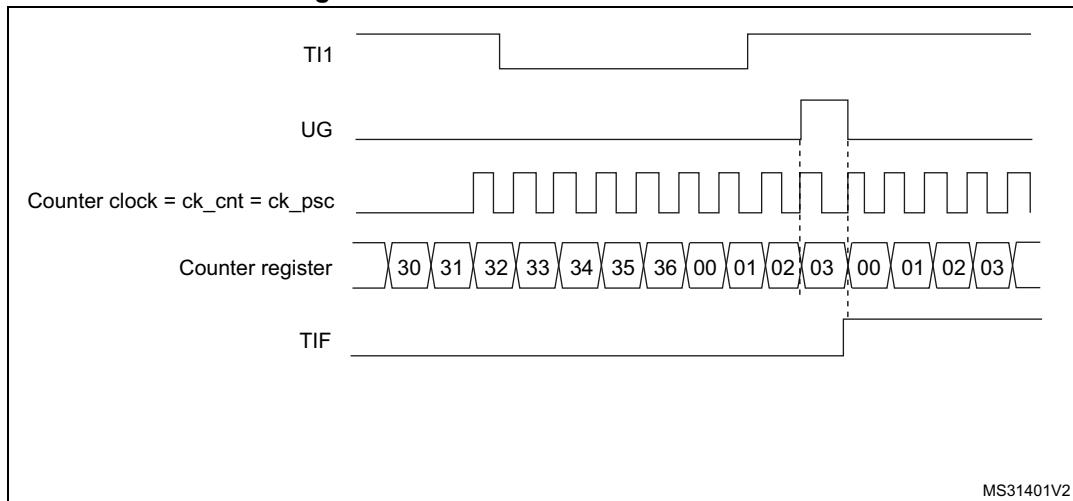
The counter and its prescaler can be reinitialized in response to an event on a trigger input. Moreover, if the URS bit from the TIMx\_CR1 register is low, an update event UEV is generated. Then all the preloaded registers (TIMx\_ARR, TIMx\_CCRx) are updated.

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. 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 223. Control circuit in reset mode**

### Slave mode: Gated mode

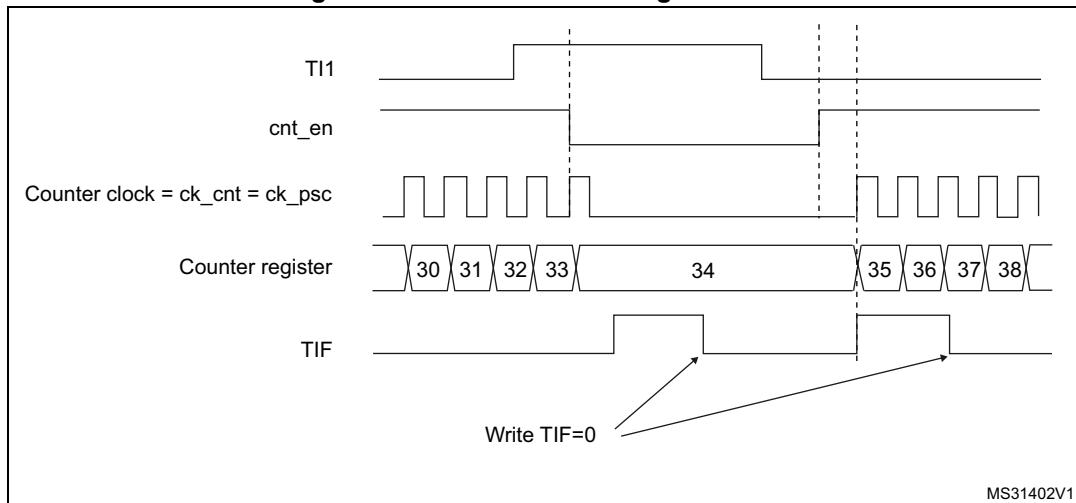
The counter can be enabled depending on the level of a selected input.

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 224. Control circuit in gated mode****Slave mode: Trigger mode**

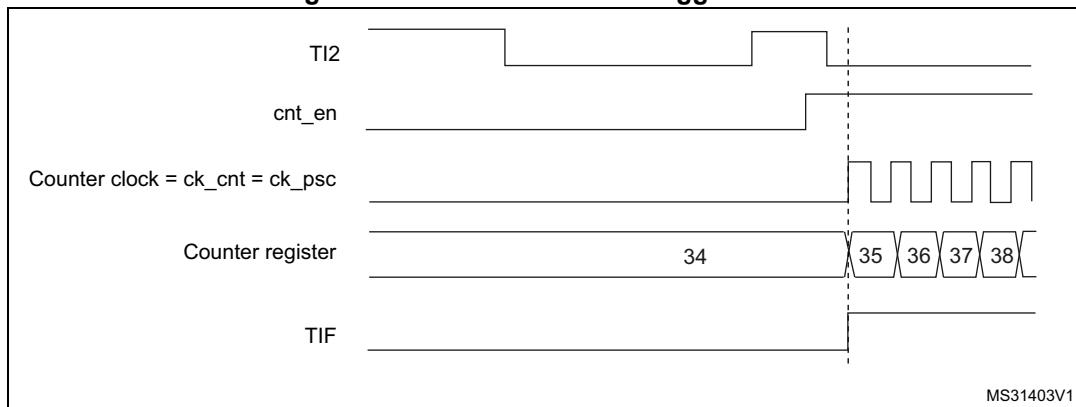
The counter can start in response to an event on a selected input.

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 225. Control circuit in trigger mode**

### 18.3.12 Timer synchronization (TIM9/12)

The TIM timers are linked together internally for timer synchronization or chaining. Refer to [Section 17.3.15: Timer synchronization](#) for details.

### 18.3.13 Debug mode

When the microcontroller enters debug mode (Cortex®-M4 with FPU 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 33.16.2: Debug support for timers, watchdog, bxCAN and I2C](#).

## 18.4 TIM9 and TIM12 registers

Refer to [Section 1.1 on page 51](#) 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).

### 18.4.1 TIM9/12 control register 1 (TIMx\_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	CKD[1:0]		ARPE	Res.	Res.	Res.	OPM	URS	UDIS	CEN

Bits 15:10 Reserved, must be kept at reset value.

Bits 9:8 **CKD**: 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:

- Counter overflow
- Setting the UG bit

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.

### 18.4.2 TIM9/12 slave mode control register (TIMx\_SMCR)

Address offset: 0x08

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	MSM	TS[2:0]				Res.	SMS[2:0]									
								rw	rw	rw	rw		rw	rw	rw	

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**: 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 (TI1FP2)
- 111: Reserved.

See [Table 115](#) 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 2:0 **SMS**: Slave mode selection

When external signals are selected, the active edge of the trigger signal (TRGI) is linked to the polarity selected on the external input (see Input control register and Control register descriptions).

- 000: Slave mode disabled - if CEN = 1 then the prescaler is clocked directly by the internal clock
- 001: Reserved
- 010: Reserved
- 011: Reserved
- 100: Reset mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter and generates an update of the registers
- 101: Gated mode - The counter clock is enabled when the trigger input (TRGI) is high. The counter stops (but is not reset) as soon as the trigger becomes low. Counter starts and stops are both controlled
- 110: Trigger mode - The counter starts on a rising edge of the trigger TRGI (but it is not reset). Only the start of the counter is controlled
- 111: External clock mode 1 - Rising edges of the selected trigger (TRGI) clock the counter

*Note:* The Gated mode must not be used if TI1F\_ED is selected as the trigger input (TS='100'). Indeed, TI1F\_ED outputs 1 pulse for each transition on TI1F, whereas the Gated mode checks the level of the trigger signal.

**Table 115. TIMx internal trigger connections**

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

### 18.4.3 TIM9/12 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	TIE	Res	Res	Res	CC2IE	CC1IE	UIE								

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.

### 18.4.4 TIM9/12 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.	Res.	Res.	Res.	Res.	CC2OF	CC1OF	Res.	Res.	TIF	Res.	Res.	Res.	CC2IF	CC1IF	UIF

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.

### 18.4.5 TIM9/12 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.	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.

### 18.4.6 TIM9/12 capture/compare mode register 1 (TIMx\_CCMR1)

Address offset: 0x18

Reset value: 0x0000

The channels can be used in input (capture mode) or in output (compare mode). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits 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.

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	rw

#### Output compare mode

Bit 15 Reserved, must be kept at reset value.

Bits 14:12 **OC2M[2:0]**: Output compare 2 mode

Bit 11 **OC2PE**: Output compare 2 preload enable

Bit 10 **OC2FE**: Output compare 2 fast enable

Bits 9:8 **CC2S[1:0]**: Capture/Compare 2 selection

This 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**: 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 the active levels of OC1 and OC1N depend on the CC1P and CC1NP bits, respectively.

000: Frozen - The comparison between the output compare register TIMx\_CCR1 and the counter TIMx\_CNT has no effect on the outputs.(this mode is used to generate a timing base).

001: Set channel 1 to active level on match. The OC1REF signal is forced high when the TIMx\_CNT counter matches the capture/compare register 1 (TIMx\_CCR1).

010: 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).

011: Toggle - OC1REF toggles when TIMx\_CNT=TIMx\_CCR1

100: Force inactive level - OC1REF is forced low

101: Force active level - OC1REF is forced high

110: PWM mode 1 - In upcounting, channel 1 is active as long as TIMx\_CNT<TIMx\_CCR1 else it is inactive. In downcounting, channel 1 is inactive (OC1REF='0') as long as TIMx\_CNT>TIMx\_CCR1, else it is active (OC1REF='1')

111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIMx\_CNT<TIMx\_CCR1 else it is active. In downcounting, channel 1 is active as long as TIMx\_CNT>TIMx\_CCR1 else it is inactive.

*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:** 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  $f_{DTS}$  1000:  $f_{SAMPLING}=f_{DTS}/8$ , N=6

0001:  $f_{SAMPLING}=f_{CK\_INT}$ , N=21001:  $f_{SAMPLING}=f_{DTS}/8$ , N=8

0010:  $f_{SAMPLING}=f_{CK\_INT}$ , N=41010:  $f_{SAMPLING}=f_{DTS}/16$ , N=5

0011:  $f_{SAMPLING}=f_{CK\_INT}$ , N=8 1011:  $f_{SAMPLING}=f_{DTS}/16$ , N=6

0100:  $f_{SAMPLING}=f_{DTS}/2$ , N=61100:  $f_{SAMPLING}=f_{DTS}/16$ , N=8

0101:  $f_{SAMPLING}=f_{DTS}/2$ , N=81101:  $f_{SAMPLING}=f_{DTS}/32$ , N=5

0110:  $f_{SAMPLING}=f_{DTS}/4$ , N=61110:  $f_{SAMPLING}=f_{DTS}/32$ , N=6

0111:  $f_{SAMPLING}=f_{DTS}/4$ , N=81111:  $f_{SAMPLING}=f_{DTS}/32$ , 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**: 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).*

### 18.4.7 TIM9/12 capture/compare enable register (TIMx\_CCER)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	CC2NP	Res.	CC2P	CC2E	CC1NP	Res.	CC1P	CC1E							

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:**

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.

*Note: 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 116. 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.

### 18.4.8 TIM9/12 counter (TIMx\_CNT)

Address offset: 0x24

Reset value: 0x0000 0000

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

Bits 15:0 **CNT[15:0]**: Counter value

### 18.4.9 TIM9/12 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.

### 18.4.10 TIM9/12 auto-reload register (TIMx\_ARR)

Address offset: 0x2C

Reset value: 0x0000

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

Bits 15:0 **ARR[15:0]**: Auto-reload value

ARR is the value to be loaded into the actual auto-reload register.

Refer to [Section 18.3.1: Time-base unit](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

### 18.4.11 TIM9/12 capture/compare register 1 (TIMx\_CCR1)

Address offset: 0x34

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR1[15:0]**: Capture/Compare 1 value

**If channel CC1 is configured as output:**

CCR1 is the value to be loaded 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).

### 18.4.12 TIM9/12 capture/compare register 2 (TIMx\_CCR2)

Address offset: 0x38

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR2[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR2[15:0]**: Capture/Compare 2 value

**If channel CC2 is configured as output:**

CCR2 is the value to be loaded 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).

#### **18.4.13 TIM9/12 register map**

TIM9/12 registers are mapped as 16-bit addressable registers as described below:

**Table 117. TIM9/12 register map and reset values**

Table 117. TIM9/12 register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x34	TIMx_CCR1	Res.																															
	Reset value	Res.																															
0x38	TIMx_CCR2	Res.																															
	Reset value	Res.																															
0x3C to 0x4C	Reserved	Res.																															

Refer to [Section 2.2.2 on page 56](#) for the register boundary addresses.

## 18.5 TIM10/11/13/14 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).

### 18.5.1 TIM10/11/13/14 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.	Res.	Res.	CKD[1:0]		ARPE	Res.	Res.	Res.	URS	UDIS	CEN	

Bits 15:10 Reserved, must be kept at reset value.

Bits 9:8 **CKD**: Clock division

This bit-field indicates the division ratio between the timer clock (CK\_INT) frequency and sampling clock used by the digital filters (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 6:3 Reserved, must be kept at reset value.

Bit 2 **URS**: Update request source

This bit is set and cleared by software to select the update interrupt (UEV) sources.

0: Any of the following events generate an UEV if enabled:

- Counter overflow
- Setting the UG bit

1: Only counter overflow generates an UEV if enabled.

Bit 1 **UDIS**: Update disable

This bit is set and cleared by software to enable/disable update interrupt (UEV) event generation.

0: UEV enabled. An UEV is generated by one of the following events:

- Counter overflow
- Setting the UG bit.

Buffered registers are then loaded with their preload values.

1: UEV disabled. No UEV is generated, shadow registers keep their value (ARR, PSC, CCRx). The counter and the prescaler are reinitialized if the UG bit is set.

Bit 0 **CEN**: Counter enable

- 0: Counter disabled
- 1: Counter enabled

### 18.5.2 TIM10/11/13/14 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.	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

### 18.5.3 TIM10/11/13/14 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.	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.

#### 18.5.4 TIM10/11/13/14 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.	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.

### 18.5.5 TIM10/11/13/14 capture/compare mode register 1 (TIMx\_CCMR1)

Address offset: 0x18

Reset value: 0x0000

The channels can be used in input (capture mode) or in output (compare mode). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OCxx describes its function when the channel is configured in output, ICxx describes its function when the channel is configured in input. So you must take care that the same bit can have a different meaning for the input stage and for the output stage.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
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 15:7 Reserved, must be kept at reset value.

Bits 6:4 **OC1M:** Output compare 1 mode

These bits define the behavior of the output reference signal OC1REF from which OC1 is derived. OC1REF is active high whereas OC1 active level depends on CC1P bit.

000: Frozen. The comparison between the output compare register TIMx\_CCR1 and the counter TIMx\_CNT has no effect on the outputs.

001: Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx\_CNT matches the capture/compare register 1 (TIMx\_CCR1).

011: Toggle - OC1REF toggles when TIMx\_CNT = TIMx\_CCR1.

100: Force inactive level - OC1REF is forced low.

101: Force active level - OC1REF is forced high.

110: PWM mode 1 - Channel 1 is active as long as TIMx\_CNT < TIMx\_CCR1 else inactive.

111: PWM mode 2 - Channel 1 is inactive as long as TIMx\_CNT < TIMx\_CCR1 else active.

*Note: In PWM mode 1 or 2, the OCREF level changes when the result of the comparison changes or when the output compare mode switches from frozen to PWM mode.*

Bit 3 **OC1PE:** Output compare 1 preload enable

0: Preload register on TIMx\_CCR1 disabled. TIMx\_CCR1 can be written at anytime, the new value is taken in account immediately.

1: Preload register on TIMx\_CCR1 enabled. Read/Write operations access the preload register. TIMx\_CCR1 preload value is loaded in the active register at each update event.

*Note: The PWM mode can be used without validating the preload register only in one pulse mode (OPM bit set in TIMx\_CR1 register). Else the behavior is not guaranteed.*

Bit 2 **OC1FE:** Output compare 1 fast enable

This bit is used to accelerate the effect of an event on the trigger in input on the CC output.

0: CC1 behaves normally depending on counter and CCR1 values even when the trigger is ON. The minimum delay to activate CC1 output when an edge occurs on the trigger input is 5 clock cycles.

1: An active edge on the trigger input acts like a compare match on CC1 output. OC is then set to the compare level independently of the result of the comparison. Delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. OC1FE acts only if the channel is configured in PWM1 or PWM2 mode.

Bits 1:0 **CC1S:** 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:

11:

*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 $f_{DTS}$	$f_{SAMPLING} = f_{DTS}/8$ , N=6
0001: $f_{SAMPLING} = f_{CK\_INT}$	N=21001: $f_{SAMPLING} = f_{DTS}/8$ , N=8
0010: $f_{SAMPLING} = f_{CK\_INT}$	N=41010: $f_{SAMPLING} = f_{DTS}/16$ , N=5
0011: $f_{SAMPLING} = f_{CK\_INT}$	N=81011: $f_{SAMPLING} = f_{DTS}/16$ , N=6
0100: $f_{SAMPLING} = f_{DTS}/2$	N=61100: $f_{SAMPLING} = f_{DTS}/16$ , N=8
0101: $f_{SAMPLING} = f_{DTS}/2$	N=81101: $f_{SAMPLING} = f_{DTS}/32$ , N=5
0110: $f_{SAMPLING} = f_{DTS}/4$	N=61110: $f_{SAMPLING} = f_{DTS}/32$ , N=6
0111: $f_{SAMPLING} = f_{DTS}/4$	N=81111: $f_{SAMPLING} = f_{DTS}/32$ , 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: Reserved

11: Reserved

*Note: CC1S bits are writable only when the channel is OFF (CC1E = 0 in TIMx\_CCER).*

### 18.5.6 TIM10/11/13/14 capture/compare enable register (TIMx\_CCER)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
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 bit selects TI1FP1 and TI2FP1 polarity for trigger or capture operations.

00: noninverted/rising edge

Circuit is sensitive to TI1FP1 rising edge (capture mode), TI1FP1 is not inverted.

01: inverted/falling edge

Circuit is sensitive to TI1FP1 falling edge (capture mode), TI1FP1 is inverted.

10: reserved, do not use this configuration.

11: noninverted/both edges

Circuit is sensitive to both TI1FP1 rising and falling edges (capture mode), TI1FP1 is not inverted.

Bit 0 **CC1E**: Capture/Compare 1 output enable.

**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 118. 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.

### 18.5.7 TIM10/11/13/14 counter (TIMx\_CNT)

Address offset: 0x24

Reset value: 0x0000

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

Bits 15:0 **CNT[15:0]**: Counter value

### 18.5.8 TIM10/11/13/14 prescaler (TIMx\_PSC)

Address offset: 0x28

Reset value: 0x0000

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

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK\_CNT is equal to  $f_{CK\_PSC} / (PSC[15:0] + 1)$ .

PSC contains the value to be loaded in the active prescaler register at each update event.

### 18.5.9 TIM10/11/13/14 auto-reload register (TIMx\_ARR)

Address offset: 0x2C

Reset value: 0x0000

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

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 18.3.1: Time-base unit](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

### 18.5.10 TIM10/11/13/14 capture/compare register 1 (TIMx\_CCR1)

Address offset: 0x34

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **CCR1[15:0]**: Capture/Compare 1 value

**If channel CC1 is configured as output:**

CCR1 is the value to be loaded in the actual capture/compare 1 register (preload value).

It is loaded permanently if the preload feature is not selected in the TIMx\_CCMR1 register (bit OC1PE). Else the preload value is copied in the active capture/compare 1 register when an update event occurs.

The active capture/compare register contains the value to be compared to the counter TIMx\_CNT and signaled on OC1 output.

**If channel CC1 is configured as input:**

CCR1 is the counter value transferred by the last input capture 1 event (IC1).

### 18.5.11 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]														
															rw

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.

00,11: TIM11 Channel1 is connected to the GPIO (refer to the Alternate function mapping table in the datasheets).

01: SPDIFRX\_FRAME\_SYNC is connected to TIM11\_CH1 to measure the clock drift of received SPDIF frames.

10: HSE\_RTC clock (HSE divided by programmable prescaler) is connected to the TIM11\_CH1 input for measurement purposes.

### 18.5.12 TIM10/11/13/14 register map

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

**Table 119. TIM10/11/13/14 register map and reset values**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	8	7	6	5	4	3	2	1	0		
0x00	<b>TIMx_CR1</b>	Res.	CKD [1:0]	8	7	6	5	4	3	2	1	0																	
	Reset value	Res.	0	0	0	0	0	0	0	0	0	0																	
0x08	<b>TIMx_SMCR</b>	Res.	0	0	0	0	0	0	0	0	0	0																	
	Reset value	Res.	0	0	0	0	0	0	0	0	0	0																	
0x0C	<b>TIMx_DIER</b>	Res.	0	0	0	0	0	0	0	0	0	0																	
	Reset value	Res.	0	0	0	0	0	0	0	0	0	0																	
0x10	<b>TIMx_SR</b>	Res.	0	0	0	0	0	0	0	0	0	0																	
	Reset value	Res.	0	0	0	0	0	0	0	0	0	0																	
0x14	<b>TIMx_EGR</b>	Res.	0	0	0	0	0	0	0	0	0	0																	
	Reset value	Res.	0	0	0	0	0	0	0	0	0	0																	
0x18	<b>TIMx_CCMR1</b> Output compare mode	Res.	0	0	0	0	0	0	0	0	0	0																	
	Reset value	Res.	0	0	0	0	0	0	0	0	0	0																	
	<b>TIMx_CCMR1</b> Input capture mode	Res.	0	0	0	0	0	0	0	0	0	0																	
0x1C	Reserved	Res.	0	0	0	0	0	0	0	0	0	0																	
	<b>TIMx_CCER</b>	Res.	0	0	0	0	0	0	0	0	0	0																	
0x20	<b>TIMx_CNT</b>	Res.	CNT[15:0]	0	0	0	0	0	0	0	0	0	0																
	Reset value	Res.	PSC[15:0]	0	0	0	0	0	0	0	0	0	0																
0x24	<b>TIMx_PSC</b>	Res.	ARR[15:0]	0	0	0	0	0	0	0	0	0	0																
	Reset value	Res.	CC1OF	0	0	0	0	0	0	0	0	0	0																
0x28	<b>TIMx_ARR</b>	Res.	OC1M [2:0]	0	0	0	0	0	0	0	0	0	0																
	Reset value	Res.	IC1F[3:0]	0	0	0	0	0	0	0	0	0	0																
0x2C	<b>TIMx_PSC</b>	Res.	ARR[15:0]	0	0	0	0	0	0	0	0	0	0																
	Reset value	Res.	CC1S [1:0]	0	0	0	0	0	0	0	0	0	0																
0x30	Reserved	Res.	CC1G o	0	0	0	0	0	0	0	0	0	0																
		Res.	CC1IE	0	0	0	0	0	0	0	0	0	0																

**Table 119. TIM10/11/13/14 register map and reset values (continued)**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x34	<b>TIMx_CCR1</b>	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		
0x38 to 0x4C	Reserved	Res.																															
0x50	<b>TIMx_OR</b>	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		

Refer to [Section 2.2.2 on page 56](#) for the register boundary addresses.

## 19 Basic timers (TIM6&TIM7)

### 19.1 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 output.

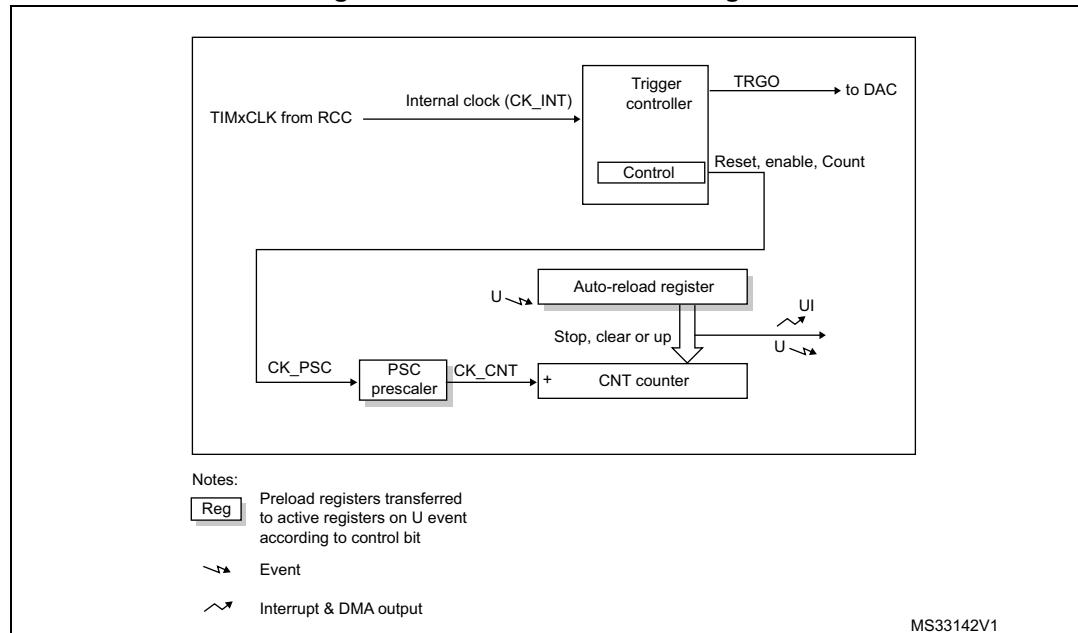
The timers are completely independent, and do not share any resources.

### 19.2 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 65536
- Synchronization circuit to trigger the DAC
- Interrupt/DMA generation on the update event: counter overflow

**Figure 226. Basic timer block diagram**



## 19.3 TIM6&TIM7 functional description

### 19.3.1 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.

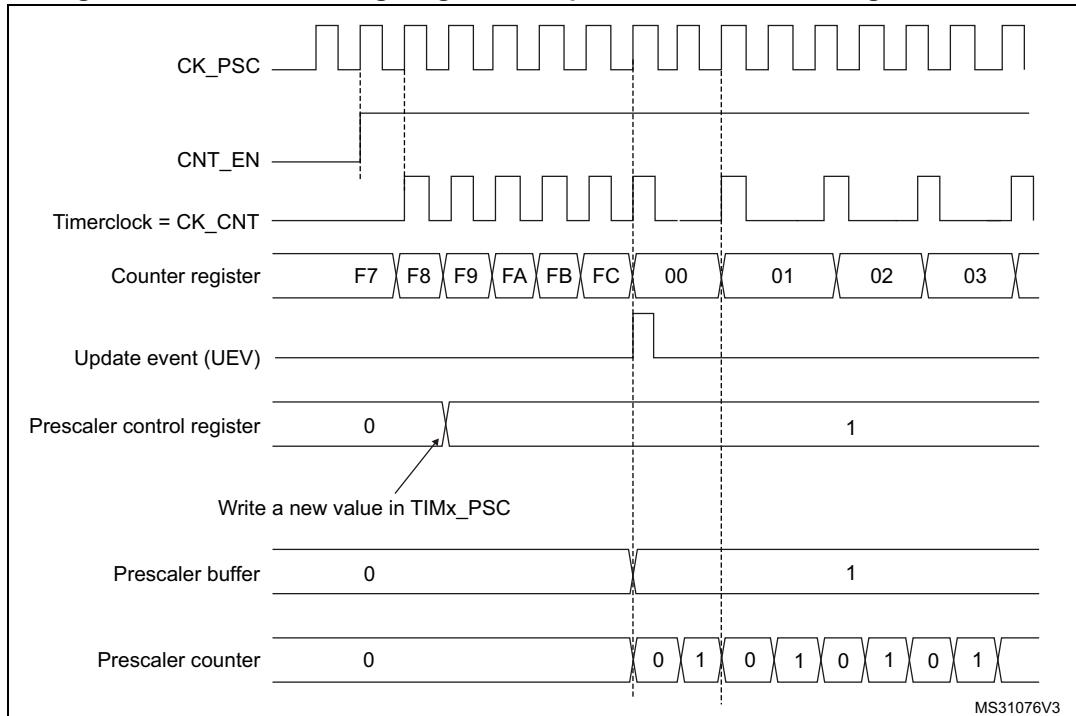
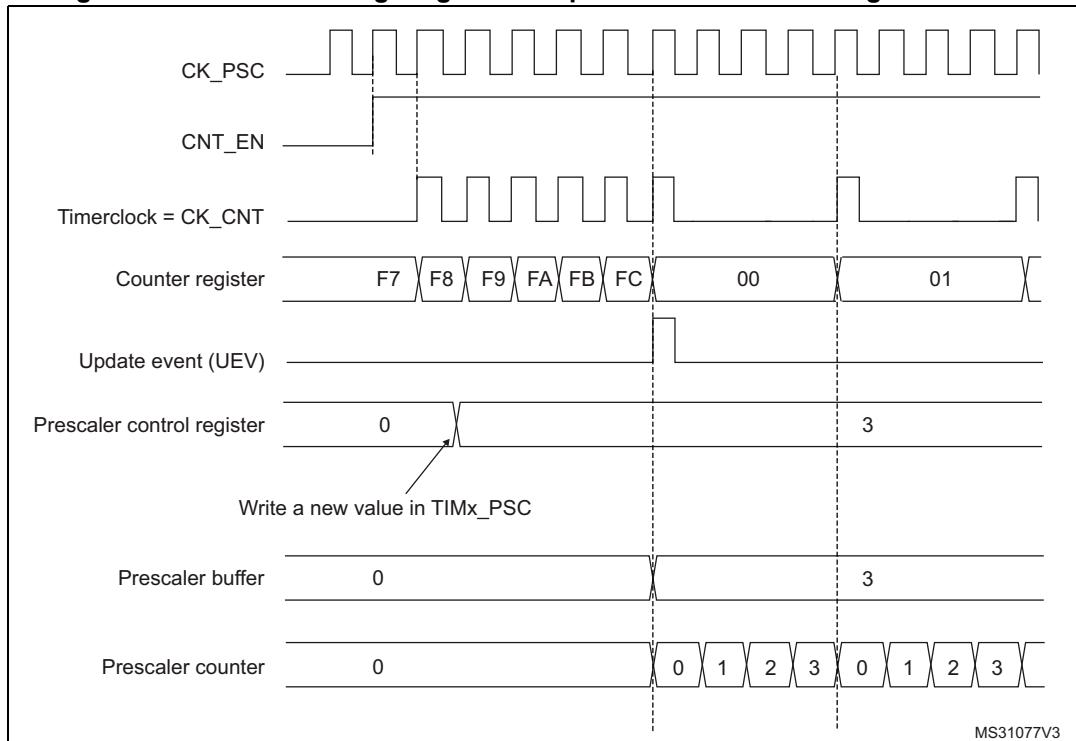
The counter is clocked by the prescaler output CK\_CNT, which is enabled only when the counter enable bit (CEN) in the TIMx\_CR1 register is set.

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 227* and *Figure 228* give some examples of the counter behavior when the prescaler ratio is changed on the fly.

**Figure 227. Counter timing diagram with prescaler division change from 1 to 2****Figure 228. Counter timing diagram with prescaler division change from 1 to 4**

### 19.3.2 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).

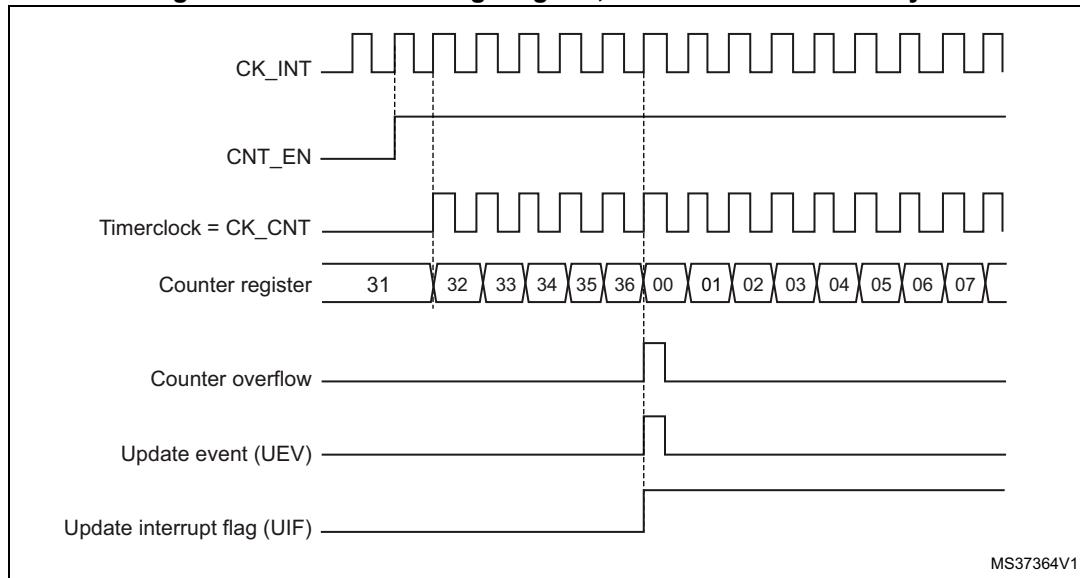
The UEV event can be disabled by software by setting the UDIS bit in the TIMx\_CR1 register. This avoids updating the shadow registers while writing new values into the preload registers. In this way, no update event occurs until the UDIS bit has been written to 0, however, the counter and the prescaler 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).

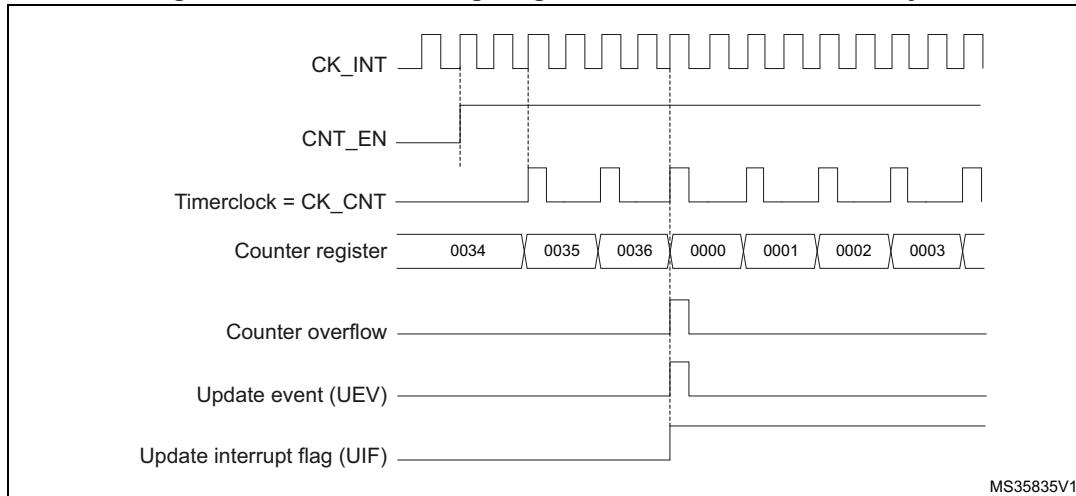
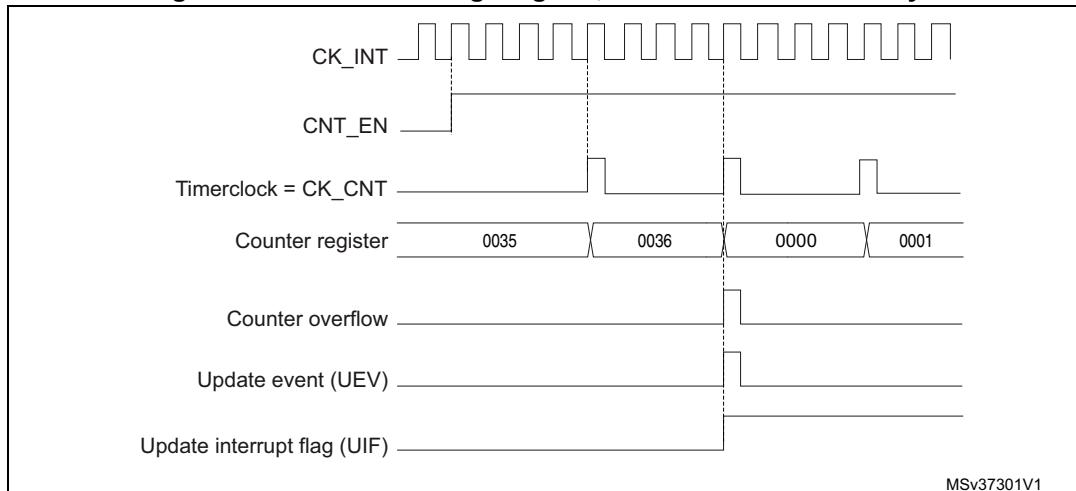
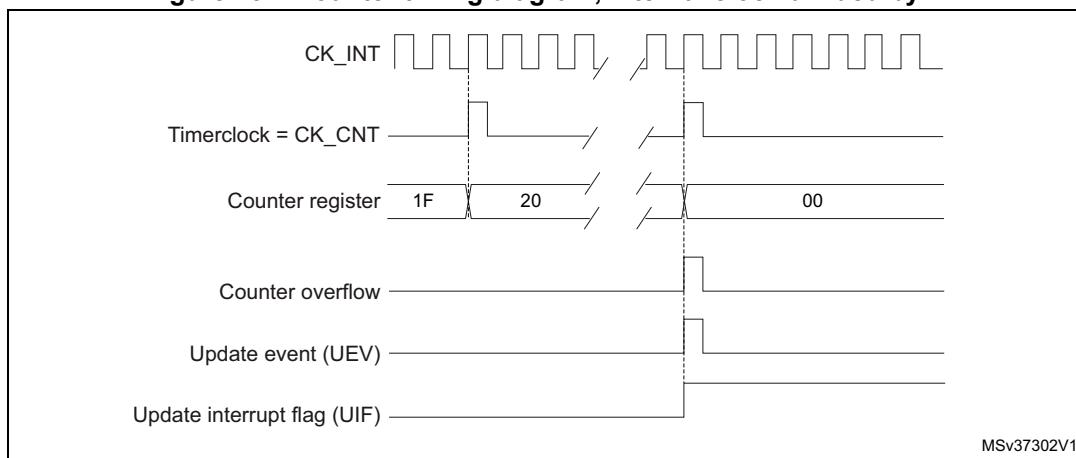
When an update event occurs, all the registers are updated and the update flag (UIF bit in the TIMx\_SR register) is set (depending on the URS bit):

- 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)

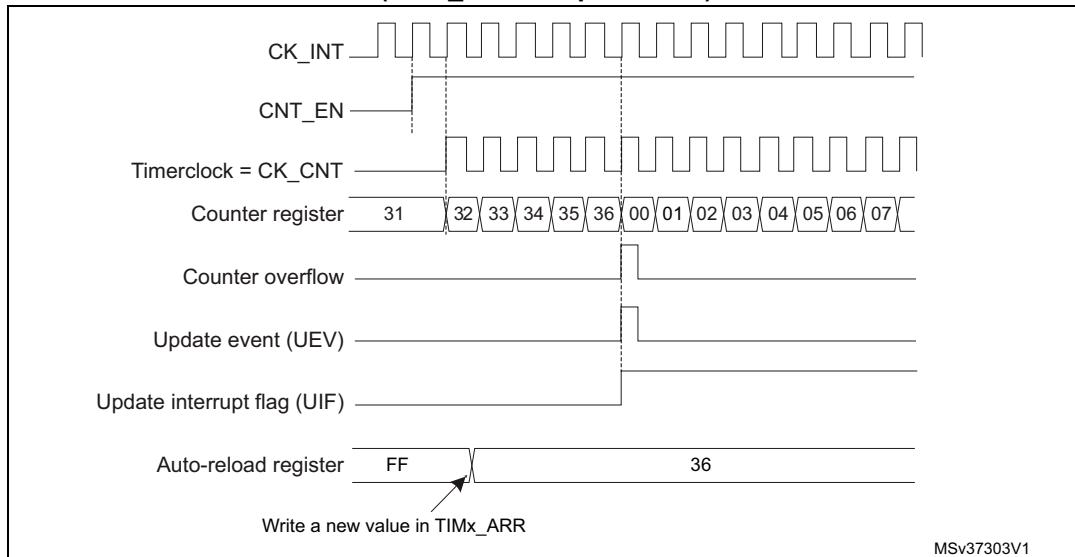
The following figures show some examples of the counter behavior for different clock frequencies when TIMx\_ARR = 0x36.

**Figure 229. Counter timing diagram, internal clock divided by 1**

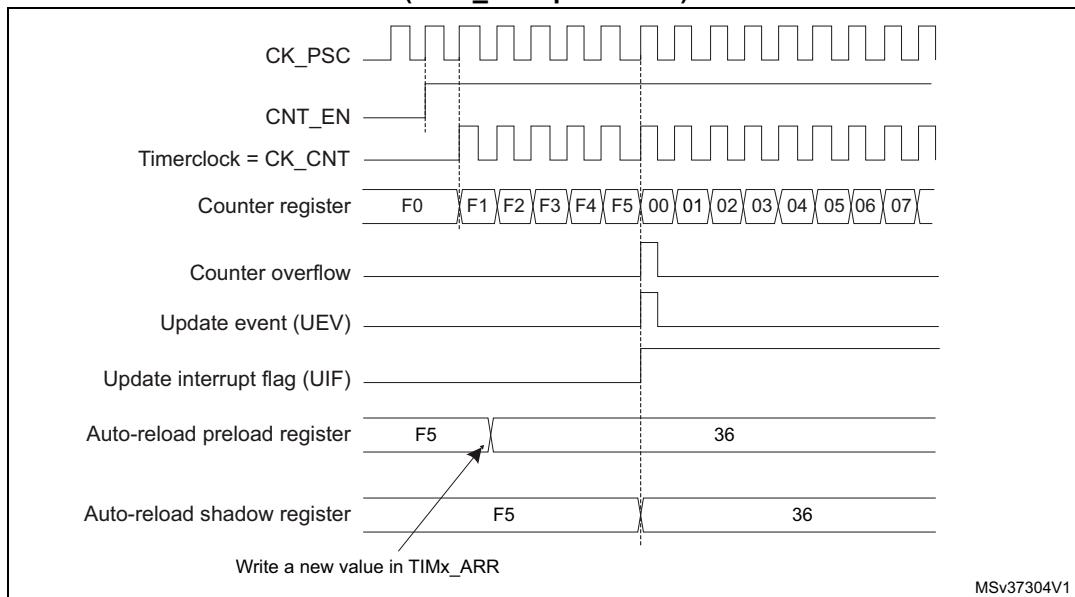


**Figure 230. Counter timing diagram, internal clock divided by 2****Figure 231. Counter timing diagram, internal clock divided by 4****Figure 232. Counter timing diagram, internal clock divided by N**

**Figure 233. Counter timing diagram, update event when ARPE = 0  
(TIMx\_ARR not preloaded)**



**Figure 234. Counter timing diagram, update event when ARPE=1  
(TIMx\_ARR preloaded)**



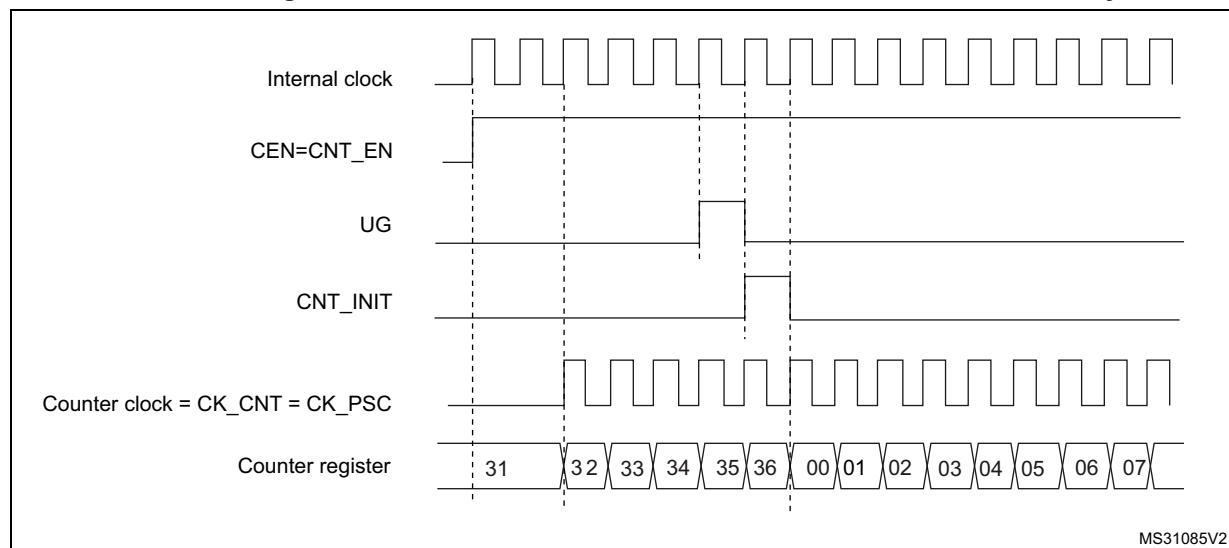
### 19.3.3 Clock source

The counter clock is provided by the Internal clock (CK\_INT) source.

The CEN (in the TIMx\_CR1 register) and UG bits (in the TIMx\_EGR register) are actual control bits and can be changed only by software (except for UG that remains cleared automatically). As soon as the CEN bit is written to 1, the prescaler is clocked by the internal clock CK\_INT.

*Figure 235* shows the behavior of the control circuit and the upcounter in normal mode, without prescaler.

Figure 235. Control circuit in normal mode, internal clock divided by 1



### 19.3.4 Debug mode

When the microcontroller enters the debug mode (Cortex®-M4 with FPU 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 33.16.2: Debug support for timers, watchdog, bxCAN and I2C](#).

## 19.4 TIM6&TIM7 registers

Refer to [Section 1.1 on page 51](#) 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).

### 19.4.1 TIM6&TIM7 control register 1 (TIMx\_CR1)

Address offset: 0x000

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	ARPE	Res.	Res.	Res.	OPM	URS	UDIS	CEN							

Bits 15:8 Reserved, 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 controller

Buffered registers are then loaded with their preload values.

1: UEV disabled. The Update event is not generated, shadow registers keep their value (ARR, PSC). However the counter and the prescaler are reinitialized if the UG bit is set or if a hardware reset is received from the slave mode controller.

**Bit 0 CEN:** Counter enable

0: Counter disabled

1: Counter enabled

*Note: Gated mode can work only if the CEN bit has been previously set by software. However trigger mode can set the CEN bit automatically by hardware.*

CEN is cleared automatically in one-pulse mode, when an update event occurs.

### 19.4.2 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.								

Bits 15:7 Reserved, must be kept at reset value.

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

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

000: **Reset** - the UG bit from the TIMx\_EGR register is used as a trigger output (TRGO). If reset is generated by the trigger input (slave mode controller configured in reset mode) then the signal on TRGO is delayed compared to the actual reset.

001: **Enable** - the Counter enable signal, CNT\_EN, is used as a trigger output (TRGO). It is useful to start several timers at the same time or to control a window in which a slave timer is enabled. The Counter Enable signal is generated by a logic OR between CEN control bit and the trigger input when configured in gated mode.

When the Counter Enable signal is controlled by the trigger input, there is a delay on TRGO, except if the master/slave mode is selected (see the MSM bit description in the TIMx\_SMCR register).

010: **Update** - The update event is selected as a trigger output (TRGO). For instance a master timer can then be used as a prescaler for a slave timer.

Bits 3:0 Reserved, must be kept at reset value.

### 19.4.3 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												

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.

#### 19.4.4 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 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.

#### 19.4.5 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).

#### 19.4.6 TIM6&TIM7 counter (TIMx\_CNT)

Address offset: 0x24

Reset value: 0x0000

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

Bits 15:0      **CNT[15:0]**: Counter value

### 19.4.7 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.

### 19.4.8 TIM6&TIM7 auto-reload register (TIMx\_ARR)

Address offset: 0x2C

Reset value: 0x0000

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

Bits 15:0 **ARR[15:0]**: Auto-reload value

ARR is the value to be loaded into the actual auto-reload register.

Refer to [Section 19.3.1: Time-base unit on page 629](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

### 19.4.9 TIM6&TIM7 register map

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

**Table 120. TIM6&TIM7 register map and reset values**

Offset	Register	Reset value	31
0x00	TIMx_CR1	Res.	Res.
	Reset value	Res.	Res.
0x04	TIMx_CR2	Res.	Res.
	Reset value	Res.	Res.
0x08	Reserved	Res.	Res.
	Reset value	Res.	Res.
0x0C	TIMx_DIER	Res.	Res.
	Reset value	Res.	Res.
0x10	TIMx_SR	Res.	Res.
	Reset value	Res.	Res.
0x14	TIMx_EGR	Res.	Res.
	Reset value	Res.	Res.
0x18	Reserved	Res.	Res.
	Reset value	Res.	Res.
0x1C	Reserved	Res.	Res.
	Reset value	Res.	Res.
0x20	Reserved	Res.	Res.
	Reset value	Res.	Res.
0x24	TIMx_CNT	Res.	Res.
	Reset value	Res.	Res.
0x28	TIMx_PSC	Res.	Res.
	Reset value	Res.	Res.
0x2C	TIMx_ARR	Res.	Res.
	Reset value	Res.	Res.
CNT[15:0]			
PSC[15:0]			
ARR[15:0]			

Refer to [Table 1 on page 56](#) for the register boundary addresses.

## 20 Independent watchdog (IWDG)

### 20.1 IWDG introduction

The devices feature two embedded watchdog peripherals that offer a combination of high safety level, timing accuracy and flexibility of use. Both watchdog peripherals (Independent and Window) serve to detect and resolve malfunctions due to software failure, and to trigger system reset or an interrupt (window watchdog only) when the counter reaches a given timeout value.

The independent watchdog (IWDG) is clocked by its own dedicated low-speed clock (LSI) and thus stays active even if the main clock fails. The window watchdog (WWDG) clock is prescaled from the APB1 clock and has a configurable time-window that can be programmed to detect abnormally late or early application behavior.

The IWDG is best suited for applications that require the watchdog to run as a totally independent process outside the main application, but have lower timing accuracy constraints. The WWDG is best suited for applications that require the watchdog to react within an accurate timing window. For further information on the window watchdog, refer to [Section 21: Window watchdog \(WWDG\)](#).

### 20.2 IWDG main features

- Free-running downcounter
- Clocked from an independent RC oscillator (can operate in Standby and Stop modes)
- Reset (if watchdog activated) when the downcounter value of 0x000 is reached

### 20.3 IWDG functional description

[Figure 236](#) shows the functional blocks of the independent watchdog module.

When the independent watchdog is started by writing the value 0xCCCC in the Key register (IWDG\_KR), the counter starts counting down from the reset value of 0xFFFF. When it reaches the end of count value (0x000) a reset signal is generated (IWDG reset).

Whenever the key value 0xAAAA is written in the IWDG\_KR register, the IWDG\_RLR value is reloaded in the counter and the watchdog reset is prevented.

#### 20.3.1 Hardware watchdog

If the “Hardware watchdog” feature is enabled through the device option bits, the watchdog is automatically enabled at power-on, and will generate a reset unless the Key register is written by the software before the counter reaches end of count.

#### 20.3.2 Register access protection

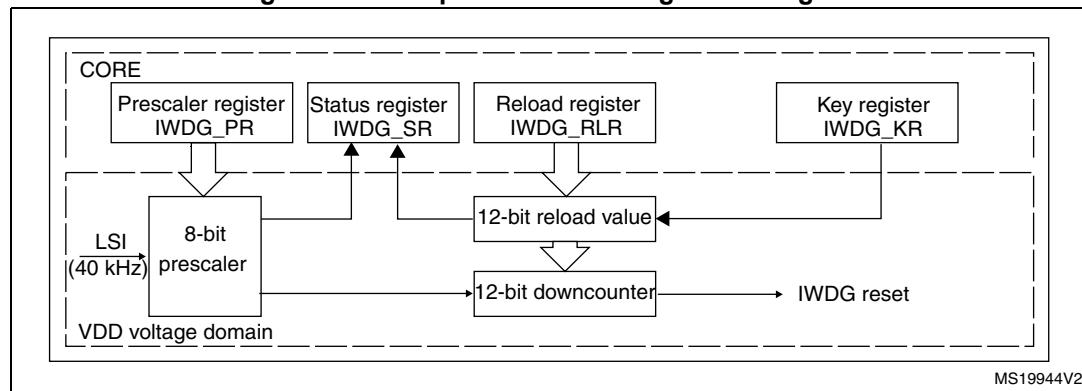
Write access to the IWDG\_PR and IWDG\_RLR registers is protected. To modify them, you must first write the code 0x5555 in the IWDG\_KR register. A write access to this register with a different value will break the sequence and register access will be protected again. This implies that it is the case of the reload operation (writing 0xAAAA).

A status register is available to indicate that an update of the prescaler or the down-counter reload value is on going.

### 20.3.3 Debug mode

When the microcontroller enters debug mode (Cortex®-M4 with FPU core halted), the IWDG counter either continues to work normally or stops, depending on `DBG_IWDG_STOP` configuration bit in DBG module. For more details, refer to [Section 34.16.4: Debug MCU APB1 freeze register \(DBGMCU\\_APB1\\_FZ\)](#).

**Figure 236. Independent watchdog block diagram**



MS19944V2

Note:

*The watchdog function is implemented in the  $V_{DD}$  voltage domain that is still functional in Stop and Standby modes.*

**Table 121. Min/max IWDG timeout period at 32 kHz (LSI)<sup>(1)</sup>**

Prescaler divider	PR[2:0] bits	Min timeout (ms) RL[11:0]= 0x000	Max timeout (ms) RL[11:0]= 0xFFFF
/4	0	0.125	512
/8	1	0.25	1024
/16	2	0.5	2048
/32	3	1	4096
/64	4	2	8192
/128	5	4	16384
/256	6	8	32768

- These timings are given for a 32 kHz clock but the microcontroller internal RC frequency can vary. Refers to LSI oscillator characteristics table in device datasheet for from max and min values.

## 20.4 IWDG registers

Refer to [Section 1.1 on page 51](#) for a list of abbreviations used in register descriptions.

The peripheral registers have to be accessed by half-words (16 bits) or words (32 bits).

### 20.4.1 Key register (IWDG\_KR)

Address offset: 0x000

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 0000h)

These bits must be written by software at regular intervals with the key value AAAAh, otherwise the watchdog generates a reset when the counter reaches 0.

Writing the key value 5555h to enable access to the IWDG\_PR and IWDG\_RLR registers (see [Section 20.3.2](#))

Writing the key value CCCCh starts the watchdog (except if the hardware watchdog option is selected)

## 20.4.2 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

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 20.3.2](#). They are written by software to select the prescaler divider feeding the counter clock. PVU bit of IWDG\_SR must be reset in order to be able to change the prescaler divider.

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

*Note:* Reading this register returns the prescaler value from the VDD voltage domain. This value may not be up to date/valid if a write operation to this register is ongoing. For this reason the value read from this register is valid only when the PVU bit in the IWDG\_SR register is reset.

### 20.4.3 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.	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.	RL[11:0]											
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:12 Reserved, must be kept at reset value.

Bits11:0 **RL[11:0]**: Watchdog counter reload value

These bits are write access protected see [Section 20.3.2](#). They are written by software to define the value to be loaded in the watchdog counter each time the value AAAAh is written in the IWDG\_KR register. The watchdog counter counts down from this value. The timeout period is a function of this value and the clock prescaler. Refer to [Table 121](#).

The RVU bit in the IWDG\_SR register must be reset in order to be able to change the reload value.

*Note: Reading this register returns the reload value from the VDD voltage domain. This value may not be up to date/valid if a write operation to this register is ongoing on this register. For this reason the value read from this register is valid only when the RVU bit in the IWDG\_SR register is reset.*

### 20.4.4 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.	RVU	PVU													
														r	r

Bits 31:2 Reserved, must be kept at reset value.

Bit 1 **RVU**: Watchdog counter reload value update

This bit is set by hardware to indicate that an update of the reload value is ongoing. It is reset by hardware when the reload value update operation is completed in the V<sub>DD</sub> voltage domain (takes up to 5 RC 40 kHz cycles).

Reload value can be updated only when RVU bit is reset.

Bit 0 **PVU**: Watchdog prescaler value update

This bit is set by hardware to indicate that an update of the prescaler value is ongoing. It is reset by hardware when the prescaler update operation is completed in the V<sub>DD</sub> voltage domain (takes up to 5 RC 40 kHz cycles).

Prescaler value can be updated only when PVU bit is reset.

**Note:** If several reload values or prescaler values are used by application, it is mandatory to wait until RVU bit is reset before changing the reload value and to wait until PVU bit is reset before changing the prescaler value. However, after updating the prescaler and/or the reload value it is not necessary to wait until RVU or PVU is reset before continuing code execution (even in case of low-power mode entry, the write operation is taken into account and will complete)

## 20.4.5 IWDG register map

The following table gives the IWDG register map and reset values.

**Table 122. IWDG 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	IWDG_KR	Res	Res																															
	Reset value																		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x04	IWDG_PR	Res	PR[2:0]	0	0	0																												
	Reset value																																	
0x08	IWDG_RLR	Res	RL[11:0]	1	1	1																												
	Reset value																																	
0x0C	IWDG_SR	Res	0	RVU	0	PVU	0																											
	Reset value																																	

Refer to [Section 2.2.2 on page 56](#) for the register boundary addresses.

## 21 Window watchdog (WWDG)

### 21.1 WWDG introduction

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

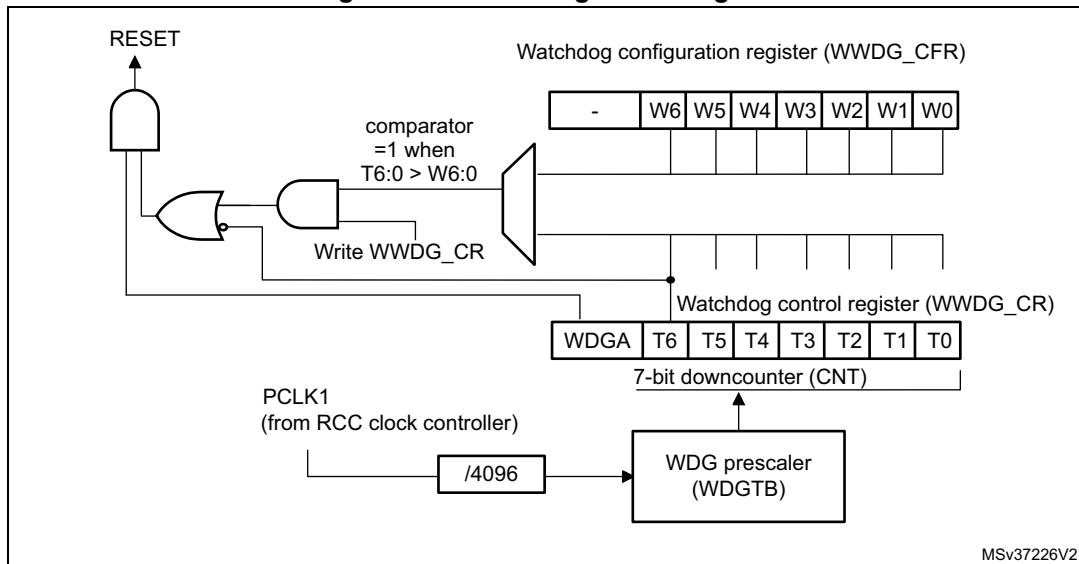
### 21.2 WWDG main features

- Programmable free-running downcounter
- Conditional reset
  - Reset (if watchdog activated) when the downcounter value becomes less than 0x40
  - Reset (if watchdog activated) if the downcounter is reloaded outside the window (see [Figure 238](#))
- Early wakeup interrupt (EWI): triggered (if enabled and the watchdog activated) when the downcounter is equal to 0x40.

### 21.3 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) rolls over from 0x40 to 0x3F (T6 becomes cleared), it initiates a reset. If the software reloads the counter while the counter is greater than the value stored in the window register, then a reset is generated.

Figure 237. Watchdog block diagram



MSv37226V2

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

### Enabling the watchdog

The watchdog is always disabled after a reset. It is enabled by setting the WDGA bit in the WWDG\_CR register, then it cannot be disabled again except by a reset.

### Controlling the downcounter

This downcounter is free-running, 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 238](#)). 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 238](#) 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).

### 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.

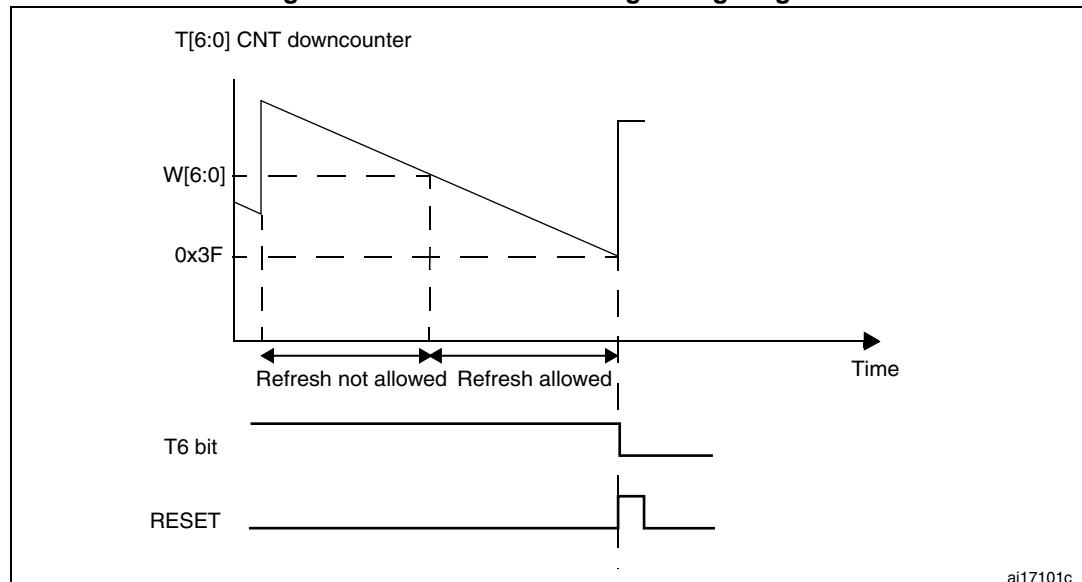
**Note:** When the EWI interrupt cannot be served, e.g. due to a system lock in a higher priority task, the WWDG reset will eventually be generated.

## 21.4 How to program the watchdog timeout

The formula in [Figure 238](#) must be used 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 238. Window watchdog timing diagram**



ai17101c

The formula to calculate the timeout value is given by:

$$t_{\text{WWDG}} = t_{\text{PCLK1}} \times 4096 \times 2^{\text{WDGTB}[1:0]} \times (\text{T5:0} + 1) \quad (\text{ms})$$

where:

$t_{\text{WWDG}}$ : WWDG timeout

$t_{\text{PCLK1}}$ : APB1 clock period measured in ms

4096: value corresponding to internal divider.

As an example, let us assume APB1 frequency is equal to 24 MHz, WDGTB[1:0] is set to 3 and T[5:0] is set to 63:

$$t_{\text{WWDG}} = 1 / 24000 \times 4096 \times 2^3 \times (63 + 1) = 21.85 \text{ ms}$$

Refer to the datasheets for the minimum and maximum values of the  $t_{\text{WWDG}}$ .

## 21.5 Debug mode

When the microcontroller enters debug mode (Cortex<sup>®</sup>-M4 with FPU core halted), the WWDG counter either continues to work normally or stops, depending on DBG\_WWDG\_STOP configuration bit in DBG module. For more details, refer to [Section 33.16.2: Debug support for timers, watchdog, bxCAN and I2C](#).

## 21.6 WWDG registers

Refer to [Section 1.1 on page 51](#) for a list of abbreviations used in register descriptions.

The peripheral registers have to be accessed by half-words (16 bits) or words (32 bits).

### 21.6.1 Control register (WWDG\_CR)

Address offset: 0x00

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						

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. It is decremented every (4096 x  $2^{WDGTB[1:0]}$ ) PCLK1 cycles. A reset is produced when it rolls over from 0x40 to 0x3F (T6 becomes cleared).

## 21.6.2 Configuration register (WWDG\_CFR)

Address offset: 0x04

Reset value: 0x0000 007F

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

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 (PCLK1 div 4096) div 1
- 01: CK Counter Clock (PCLK1 div 4096) div 2
- 10: CK Counter Clock (PCLK1 div 4096) div 4
- 11: CK Counter Clock (PCLK1 div 4096) div 8

Bits 6:0 **W[6:0]**: 7-bit window value

These bits contain the window value to be compared to the downcounter.

## 21.6.3 Status register (WWDG\_SR)

Address offset: 0x08

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'. A write of '1' has no effect. This bit is also set if the interrupt is not enabled.

## 21.6.4 WWDG register map

The following table gives the WWDG register map and reset values.

**Table 123. WWDG register map and reset values**

Offset	Register	Reset value	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
0x00	<b>WWDG_CR</b>	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	
			Res																																			
0x04	<b>WWDG_CFR</b>	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	
			Res																																			
0x08	<b>WWDG_SR</b>	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res
			Res																																			

Refer to [Section 2.2.2 on page 56](#) for the register boundary addresses.

## 22 Real-time clock (RTC)

### 22.1 Introduction

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 also includes an automatic wakeup unit to manage low power modes.

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).

### 22.2 RTC main features

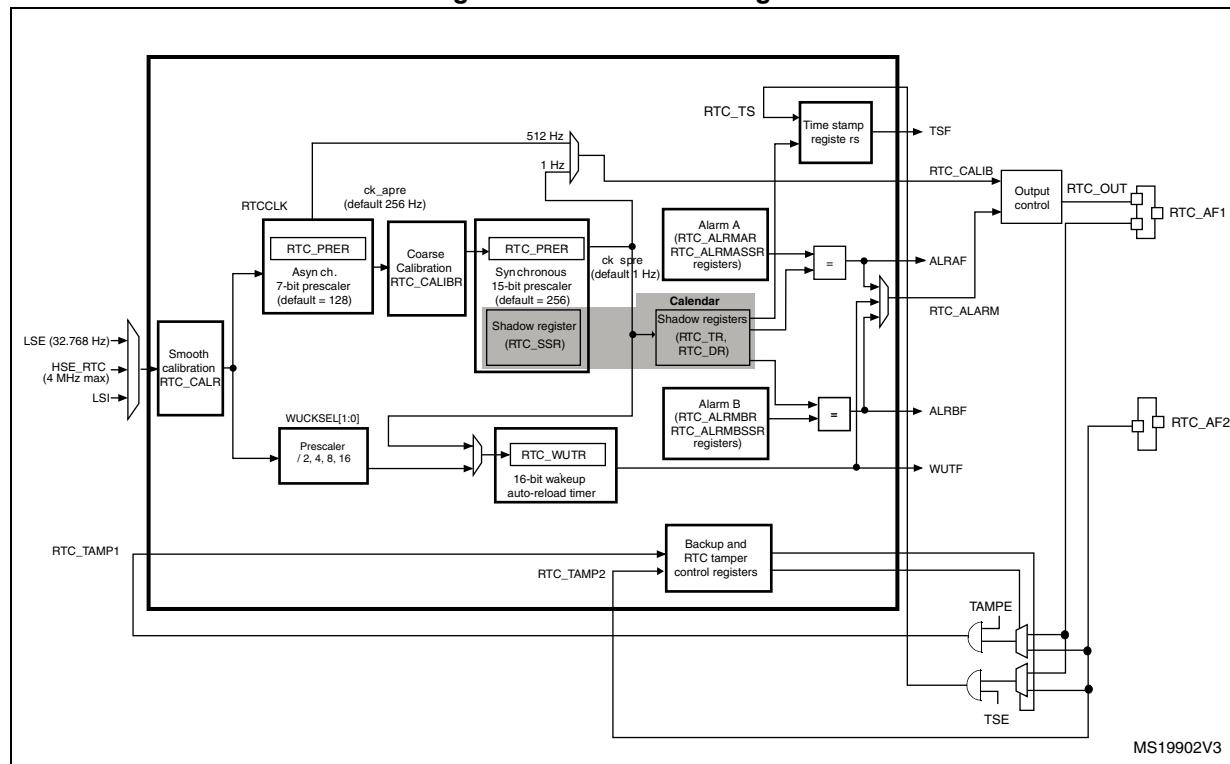
The RTC unit main features are the following (see [Figure 239](#)):

- 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.
- Two programmable alarms with interrupt function. The alarms 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.
- Maskable interrupts/events:
  - Alarm A
  - Alarm B
  - Wakeup interrupt
  - Timestamp
  - Tamper detection
- Digital calibration circuit (periodic counter correction)
  - 5 ppm accuracy

- 0.95 ppm accuracy, obtained in a calibration window of several seconds
- Timestamp function for event saving (1 event)
- Tamper detection:
  - 2 tamper events with configurable filter and internal pull-up.
- 20 backup registers (80 bytes). The backup registers are reset when a tamper detection event occurs.
- Alternate function output (RTC\_OUT) which selects one of the following two outputs:
  - RTC\_CALIB: 512 Hz or 1 Hz clock output (with an LSE frequency of 32.768 kHz). This output is enabled by setting the COE bit in the RTC\_CR register. It is routed to the device RTC\_AF1 function.
  - RTC\_ALARM (Alarm A, Alarm B or wakeup). This output is selected by configuring the OSEL[1:0] bits in the RTC\_CR register. It is routed to the device RTC\_AF1 function.
- RTC alternate function inputs:
  - RTC\_TS: timestamp event detection. It is routed to the device RTC\_AF1 and RTC\_AF2 functions.
  - RTC\_TAMP1: TAMPER1 event detection. It is routed to the device RTC\_AF1 and RTC\_AF2 functions.
  - RTC\_TAMP2: TAMPER2 event detection.
  - RTC\_REFIN: reference clock input (usually the mains, 50 or 60 Hz).

Refer to [Section 7.3.15: Selection of RTC additional functions](#).

**Figure 239. RTC block diagram**



1. On STM32F446xx devices, the RTC\_AF1 and RTC\_AF2 additional function are connected to PC13 and PA0, respectively.

## 22.3 RTC functional description

### 22.3.1 Clock and prescalers

The RTC clock source (RTCCLK) is selected through the clock controller among the LSE clock, the LSI oscillator clock, and the HSE clock. For more information on the RTC clock source configuration, refer to [Section 6: Reset and clock control \(RCC\)](#).

A programmable prescaler stage generates a 1 Hz clock which is used to update the calendar. To minimize power consumption, the prescaler is split into 2 programmable prescalers (see [Figure 239: 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 22.3.4](#) for details).

### 22.3.2 Real-time clock and calendar

The RTC calendar time and date registers are accessed through shadow registers which are synchronized with PCLK1 (APB1 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 two RTCCLK periods, the current calendar value is copied into the shadow registers, and the RSF bit of RTC\_ISR register is set (see [Section 22.6.4](#)). The copy is not performed in Stop and Standby mode. When exiting these modes, the shadow registers are updated after up to 2 RTCCLK periods.

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.

### 22.3.3 Programmable alarms

The RTC unit provides two programmable alarms, Alarm A and Alarm B.

The programmable alarm functions are enabled through the ALRAIE and ALRBIE bits in the RTC\_CR register. The ALRAF and ALRBF flags are set to 1 if the calendar subseconds, seconds, minutes, hours, date or day match the values programmed in the alarm registers RTC\_ALRMASSR/RTC\_ALRMAR and RTC\_ALRMBSSR/RTC\_ALRMBR, respectively. Each calendar field can be independently selected through the MSKx bits of the RTC\_ALRMAR and RTC\_ALRMBR registers, and through the MASKSSx bits of the RTC\_ALRMASSR and RTC\_ALRMBSSR registers. The alarm interrupts are enabled through the ALRAIE and ALRBIE bits in the RTC\_CR register.

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 polarity can be configured through bit POL in the RTC\_CR register.

**Caution:** If the seconds field is selected (MSK0 bit reset in RTC\_ALRMAR or RTC\_ALRMBR), the synchronous prescaler division factor set in the RTC\_PRER register must be at least 3 to ensure correct behavior.

### 22.3.4 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(32.768 kHz), this allows to configure the wakeup interrupt period from 122 µs to 32 s, with a resolution down to 61µs.
- 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 2<sup>16</sup> is added to the 16-bit counter current value. When the initialization sequence is

complete (see [Programming the wakeup timer](#)), 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\_CR2 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 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.

### 22.3.5 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 with the DBP bit of the PWR power control register (PWR\_CR). The DBP bit must be set 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\_ISR[13:8], RTC\_TAFCR, and RTC\_BKPxR.

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 from 1 to 2 RTCCLK clock cycles (due to clock synchronization).
3. To generate a 1 Hz clock for the calendar counter, program first the synchronous prescaler factor in RTC\_PRER register, and then program the asynchronous prescaler

- factor. Even if only one of the two fields needs to be changed, 2 separate write accesses must be performed to the 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 alarm (Alarm A or Alarm B):

1. Clear ALRAE or ALRBIE in RTC\_CR to disable Alarm A or Alarm B.
2. Poll ALRAWF or ALRBWF in RTC\_ISR until it is set to make sure the access to alarm registers is allowed. This takes 1 to 2 RTCCLK clock cycles (due to clock synchronization).
3. Program the Alarm A or Alarm B registers (RTC\_ALRMASSR/RTC\_ALRMAR or RTC\_ALRMBSSR/RTC\_ALRMBR).
4. Set ALRAE or ALRBIE in the RTC\_CR register to enable Alarm A or Alarm B again.

**Note:**

*Each change of the RTC\_CR register is taken into account after 1 to 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 1 to 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. Due to clock synchronization, the WUTWF bit is cleared up to 2 RTCCLK clocks cycles after WUTE is cleared.

### 22.3.6 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 APB1 clock frequency ( $f_{PCLK1}$ ) must be equal to or greater than seven times the  $f_{RTCCLK}$  RTC clock frequency. This ensures a secure behavior of the synchronization mechanism.

If the APB1 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 APB1 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 two 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.

*Note:* 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](#)): the software must wait until RSF is set before reading the RTC\_SSR, RTC\_TR and RTC\_DR registers.*

*After synchronization (refer to [Section 22.3.8](#)): 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.

### 22.3.7 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 registers (RTC\_CALIBR or RTC\_CALR), the RTC shift register (RTC\_SHIFTR), the RTC timestamp registers (RTC\_TSSSR, RTC\_TSTR and RTC\_TSDDR), the RTC tamper and alternate function configuration register (RTC\_TAFCR), 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).

In addition, the RTC keeps on running under system reset if the reset source is different from a backup domain reset. When a backup domain reset occurs, the RTC is stopped and all the RTC registers are set to their reset values.

### 22.3.8 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’s counter. This allows one to calculate the exact time being maintained by the RTC down to a resolution of  $1 / (\text{PREDIV\_S} + 1)$  seconds. As a consequence, the resolution can be improved by increasing the synchronous prescaler value (PREDIV\_S[14:0]. The maximum resolution allowed (30.52 µs with a 32768 Hz clock) is obtained with PREDIV\_S set to 0xFFFF.

However, increasing PREDIV\_S means that PREDIV\_A must be decreased in order to maintain the synchronous prescaler’s output at 1 Hz. In this way, the frequency of the asynchronous prescaler’s 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.

### 22.3.9 RTC reference clock detection

The RTC calendar update can be synchronized to a reference clock RTC\_REFIN, usually the mains (50 or 60 Hz). The RTC\_REFIN reference clock should have a higher precision 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 reference 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 asynchronous prescaler which outputs the ck\_apre 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 reference clock detection is enabled, PREDIV\_A and PREDIV\_S must be set to their default values:

- PREDIV\_A = 0x007F
- PREDIV\_S = 0x00FF

**Note:** *The reference clock detection is not available in Standby mode.*

**Caution:** The reference clock detection feature cannot be used in conjunction with the coarse digital calibration: RTC\_CALIBR must be kept at 0x0000 0000 when REFCKON=1.

### 22.3.10 RTC coarse digital calibration

Two digital calibration methods are available: coarse and smooth calibration. To perform coarse calibration refer to [Section 22.6.7: RTC calibration register \(RTC\\_CALIBR\)](#).

The two calibration methods are not intended to be used together, the application must select one of the two methods. Coarse calibration is provided for compatibility reasons. To perform smooth calibration refer to [Section 22.3.11: RTC smooth digital calibration](#) and to [Section 22.6.16: RTC calibration register \(RTC\\_CALR\)](#)

The coarse digital calibration can be used to compensate crystal inaccuracy by adding (positive calibration) or masking (negative calibration) clock cycles at the output of the asynchronous prescaler (ck\_apre).

Positive and negative calibration are selected by setting the DCS bit in RTC\_CALIBR register to '0' and '1', respectively.

When positive calibration is enabled (DCS = '0'), 2 ck\_apre cycles are added every minute (around 15360 ck\_apre cycles) for 2xDC minutes. This causes the calendar to be updated sooner, thereby adjusting the effective RTC frequency to be a bit higher.

When negative calibration is enabled (DCS = '1'), 1 ck\_apre cycle is removed every minute (around 15360 ck\_apre cycles) for 2xDC minutes. This causes the calendar to be updated later, thereby adjusting the effective RTC frequency to be a bit lower.

DC is configured through bits DC[4:0] of RTC\_CALIBR register. This number ranges from 0 to 31 corresponding to a time interval (2xDC) ranging from 0 to 62.

The coarse digital calibration can be configured only in initialization mode, and starts when the INIT bit is cleared. The full calibration cycle lasts 64 minutes. The first 2xDC minutes of the 64 -minute cycle are modified as just described.

Negative calibration can be performed with a resolution of about 2 ppm while positive calibration can be performed with a resolution of about 4 ppm. The maximum calibration ranges from -63 ppm to 126 ppm.

The calibration can be performed either on the LSE or on the HSE clock.

**Caution:** Digital calibration may not work correctly if PREDIV\_A < 6.

#### Case of RTCCLK=32.768 kHz and PREDIV\_A+1=128

The following description assumes that ck\_apre frequency is 256 Hz obtained with an LSE clock nominal frequency of 32.768 kHz, and PREDIV\_A set to 127 (default value).

The ck\_spre clock frequency is only modified during the first 2xDC minutes of the 64-minute cycle. For example, when DC equals 1, only the first 2 minutes are modified. This means that the first 2xDC minutes of each 64-minute cycle have, once per minute, one second either shortened by 256 or lengthened by 128 RTCCLK cycles, given that each ck\_apre cycle represents 128 RTCCLK cycles (with PREDIV\_A+1=128).

Therefore each calibration step has the effect of adding 512 or subtracting 256 oscillator cycles for every 125829120 RTCCLK cycles (64min x 60 s/min x 32768 cycles/s). This is equivalent to +4.069 ppm or -2.035 ppm per calibration step. As a result, the calibration resolution is +10.5 or -5.27 seconds per month, and the total calibration ranges from +5.45 to -2.72 minutes per month.

In order to measure the clock deviation, a 512 Hz clock is output for calibration. Refer to [Section 22.3.14: Calibration clock output](#).

### 22.3.11 RTC smooth digital calibration

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\_CALRx) 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 (FCAL) given the input frequency (FRTCCLK) 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 performed 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.

### 22.3.12 Timestamp function

Timestamp is enabled by setting the TSE bit of RTC\_CR register to 1.

The calendar is saved in the timestamp registers (RTC\_TSSSR, RTC\_TSTR, RTC\_TSRR) when a timestamp event is detected on the pin to which the TIMESTAMP alternate function is mapped. When a timestamp event occurs, the timestamp flag bit (TSF) in RTC\_ISR register is set.

By setting the TSIE bit in the RTC\_CR register, an interrupt is generated when a timestamp event occurs.

If a new timestamp event is detected while the timestamp flag (TSF) is already set, the timestamp overflow flag (TSOVF) flag is set and the timestamp registers (RTC\_TSTR and RTC\_TSDR) maintain the results of the previous event.

**Note:** *TSF is set 2 ck\_apre cycles after the timestamp event occurs due to synchronization process.*

*There is no delay in the setting of TSOVF. This means that if two timestamp 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 timestamp event occurs immediately after the TSF bit is supposed to be cleared, then both TSF and TSOVF bits are set. To avoid masking a timestamp 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 timestamp to be recorded. See the description of the TAMPTS control bit in [Section 22.6.17: RTC tamper and alternate function configuration register \(RTC\\_TAFCR\)](#). If the timestamp event is on the same pin as a tamper event configured in filtered mode (TAMPFLT set to a non-zero value), the timestamp on tamper detection event mode must be selected by setting TAMPTS='1' in RTC\_TAFCR register.

### TIMESTAMP alternate function

The TIMESTAMP alternate function (RTC\_TS) can be mapped either to RTC\_AF1 or to RTC\_AF2 depending on the value of the TSINSEL bit in the RTC\_TAFCR register (see [Section 22.6.17: RTC tamper and alternate function configuration register \(RTC\\_TAFCR\)](#)). Mapping the timestamp event on RTC\_AF2 is not allowed if RTC\_AF1 is used as TAMPER in filtered mode (TAMPFLT set to a non-zero value).

#### 22.3.13 Tamper detection

Two tamper detection inputs are available. They can be configured either for edge detection, or for level detection with filtering.

### RTC backup registers

The backup registers (RTC\_BKPxR) are twenty 32-bit registers for storing 80 bytes of user application data. They are implemented in the backup domain that remains powered-on by V<sub>BAT</sub> when the V<sub>DD</sub> power is switched off. They are not reset by system reset or when the device wakes up from Standby mode. They are reset by a backup domain reset.

The backup registers are reset when a tamper detection event occurs (see [Section 22.6.20: RTC backup registers \(RTC\\_BKPxR\)](#) and [Tamper detection initialization on page 665](#)).

### Tamper detection initialization

Each tamper detection input is associated with the TAMP1F/TAMP2F flags in the RTC\_ISR2 register. Each input can be enabled by setting the corresponding TAMP1E/TAMP2E bits to 1 in the RTC\_TAFCR register.

A tamper detection event resets all backup registers (RTC\_BKPxR).

By setting the TAMPIE bit in the RTC\_TAFCR register, an interrupt is generated when a tamper detection event occurs.

### Timestamp on tamper event

With TAMPTS set to '1', any tamper event causes a timestamp to occur. In this case, either the TSF bit or the TSOVF bit are set in RTC\_ISR, in the same manner as if a normal timestamp event occurs. The affected tamper flag register (TAMP1F, TAMP2F) is set at the same time that TSF or TSOVF is set.

### Edge detection on tamper inputs

If the TAMPFLT bits are "00", the TAMPER pins generate tamper detection events (RTC\_TAMP[2:1]) when either a rising edge is observed or a falling edge is observed depending on the corresponding TAMPxTRG bit. The internal pull-up resistors on the TAMPER inputs are deactivated when edge detection is selected.

**Caution:** To avoid losing tamper detection events, the signal used for edge detection is logically ANDed with TAMPxE in order to detect a tamper detection event in case it occurs before the TAMPERx pin is enabled.

- When TAMPxTRG = 0: if the TAMPERx alternate function is already high before tamper detection is enabled (TAMPxE bit set to 1), a tamper event is detected as soon as TAMPERx is enabled, even if there was no rising edge on TAMPERx after TAMPxE was set.
- When TAMPxTRG = 1: if the TAMPERx alternate function is already low before tamper detection is enabled, a tamper event is detected as soon as TAMPERx is enabled (even if there was no falling edge on TAMPERx after TAMPxE was set).

After a tamper event has been detected and cleared, the TAMPERx alternate function 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 TAMPERx value still indicates a tamper detection. This is equivalent to a level detection on the TAMPERx alternate function.

**Note:** *Tamper detection is still active when V<sub>DD</sub> power is switched off. To avoid unwanted resetting of the backup registers, the pin to which the TAMPER alternate function is mapped should be externally tied to the correct level.*

### Level detection with filtering on tamper 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 (TAMP1TRG/TAMP2TRG).

The TAMPER inputs are pre-charged 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 tamper 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.*

### TAMPER alternate function detection

The TAMPER1 alternate function (RTC\_TAMP1) can be mapped either to RTC\_AF1(PC13) or RTC\_AF2 (PA0) depending on the value of TAMP1INSEL bit in RTC\_TAFCR register

(see [Section 22.6.17](#)). TAMPE bit must be cleared when TAMP1INSEL is modified to avoid unwanted setting of TAMPF.

The TAMPER 2 alternate function corresponds to RTC\_TAMP2 pin.

### 22.3.14 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 output is not impacted by the calibration value programmed in RTC\_CALIBR register. The RTC\_CALIB duty cycle is irregular: there is a light jitter on falling edges. It is therefore recommended to use rising edges.

If 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.

#### Calibration alternate function output

When the COE bit in the RTC\_CR register is set to 1, the calibration alternate function (RTC\_CALIB) is enabled on RTC\_AF1.

*Note:* When RTC\_CALIB or RTC\_ALARM is selected, RTC\_AF1 is automatically configured in output alternate function.

### 22.3.15 Alarm output

Three functions can be selected on Alarm output: ALRAF, ALRBF and WUTF. These functions reflect the contents of the corresponding flags in the RTC\_ISR register.

The OSEL[1:0] control bits in the RTC\_CR register are used to activate the alarm alternate function output (RTC\_ALARM) in RTC\_AF1, and to select the function which is output on RTC\_ALARM.

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 alternate function output

RTC\_ALARM can be configured in output open drain or output push-pull using the control bit ALARMOUTTYPE in the RTC\_TAFCR register.

*Note:* Once RTC\_ALARM is enabled, it has priority over RTC\_CALIB (COE bit is don't care on RTC\_AF1).

*When RTC\_CALIB or RTC\_ALARM is selected, RTC\_AF1 is automatically configured in output alternate function.*

## 22.4 RTC and low power modes

**Table 124. 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 time stamp 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 time stamp event, and RTC Wakeup cause the device to exit the Standby mode.

## 22.5 RTC interrupts

All RTC interrupts are connected to the EXTI controller.

To enable the RTC Alarm interrupt, the following sequence is required:

1. Configure and enable the EXTI Line 17 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 (Alarm A or Alarm B).

To enable the RTC Wakeup interrupt, the following sequence is required:

1. Configure and enable the EXTI Line 22 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 generate the RTC wakeup timer event.

To enable the RTC Tamper interrupt, the following sequence is required:

1. Configure and enable the EXTI Line 21 in interrupt mode and select the rising edge sensitivity.
2. Configure and Enable the 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 21 in interrupt mode and select the rising edge sensitivity.
2. Configure and Enable the TAMP\_STAMP IRQ channel in the NVIC.
3. Configure the RTC to detect the RTC timestamp event.

**Table 125. Interrupt control bits**

Interrupt event	Event flag	Enable control bit	Exit the Sleep mode	Exit the Stop mode	Exit the Standby mode
Alarm A	ALRAF	ALRAIE	yes	yes <sup>(1)</sup>	yes <sup>(1)</sup>
Alarm B	ALRBF	ALRBIE	yes	yes <sup>(1)</sup>	yes <sup>(1)</sup>
Wakeup	WUTF	WUTIE	yes	yes <sup>(1)</sup>	yes <sup>(1)</sup>
TimeStamp	TSF	TSIE	yes	yes <sup>(1)</sup>	yes <sup>(1)</sup>
Tamper1 detection	TAMP1F	TAMPIE	yes	yes <sup>(1)</sup>	yes <sup>(1)</sup>
Tamper2 detection <sup>(2)</sup>	TAMP2F	TAMPIE	yes	yes <sup>(1)</sup>	yes <sup>(1)</sup>

1. Wakeup from STOP and Standby modes is possible only when the RTC clock source is LSE or LSI.

2. If RTC\_TAMPER2 pin is present. Refer to device datasheet pinout.

## 22.6 RTC registers

Refer to [Section 1.1 on page 51](#) of this reference manual for a list of abbreviations used in register descriptions.

The peripheral registers have to be accessed by words (32 bits).

### 22.6.1 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](#) and [Reading the calendar](#).

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	<b>HT[1:0]</b>		<b>HU[3:0]</b>			
									rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	<b>MNT[2:0]</b>			<b>MNU[3:0]</b>				Res.	<b>ST[2:0]</b>			<b>SU[3:0]</b>			
	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 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

**Note:** *This register is write protected. The write access procedure is described in [RTC register write protection on page 657](#).*

## 22.6.2 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](#) and [Reading the calendar](#).

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]			
								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDU[2:0]			MT	MU[3:0]				Res.	Res.	DT[1:0]		DU[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw

Bits 31-24 Reserved, must be kept at reset value

Bits 23:20 **YT[3:0]**: Year tens in BCD format

Bits 19:16 **YU[3:0]**: Year units in BCD format

Bits 15:13 **WDU[2:0]**: Week day units

000: forbidden

001: Monday

...

111: Sunday

Bit 12 **MT**: Month tens in BCD format

Bits 11:8 **MU**: 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

**Note:** *This register is write protected. The write access procedure is described in [RTC register write protection](#).*

### 22.6.3 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.	Res.	COE	OSEL[1:0]	POL	COSEL	BKP	SUB1H	ADD1H	
								rw	rw	rw	rw	rw	rw	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TSIE	WUTIE	ALRBIE	ALRAIE	TSE	WUTE	ALRBE	ALRAE	DCE	FMT	BYPSHAD	REFCKON	TSEDGE	WUCKSEL[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bit 23 **COE**: Calibration output enable

This bit enables the RTC\_CALIB output

0: Calibration output disabled

1: Calibration output enabled

Bits 22:21 **OSEL[1:0]**: Output selection

These bits are used to select the flag to be routed to RTC\_ALARM output

00: Output disabled

01: Alarm A output enabled

10: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

1: Calibration output is 1 Hz

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 22.3.14: 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 outside initialization mode, 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.

Bit 16 **ADD1H**: Add 1 hour (summer time change)

When this bit is set outside initialization mode, 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

Bit 15 **TSIE**: Timestamp interrupt enable

0: Timestamp Interrupt disable

1: Timestamp 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**: Time stamp enable

0: Time stamp disable

1: Time stamp enable

Bit 10 **WUTE**: Wakeup timer enable

0: Wakeup timer disabled

1: Wakeup timer enabled

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 **DCE**: Coarse digital calibration enable

0: Digital calibration disabled

1: Digital calibration enabled

PREDIV\_A must be 6 or greater

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 APB1 clock is less than seven times the frequency of RTCCLK, BYPSHAD must be set to '1'.*

Bit 4 **REFCKON**: Reference clock detection enable (50 or 60 Hz)

- 0: Reference clock detection disabled
- 1: Reference clock detection enabled

*Note:* *PREDIV\_S* must be 0x00FF.

Bit 3 **TSEDGE**: Timestamp event active edge

- 0: TIMESTAMP rising edge generates a timestamp event
- 1: TIMESTAMP falling edge generates a timestamp 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:* *WUT* = Wakeup unit counter value. *WUT* = (0x0000 to 0xFFFF) + 0x10000 added when *WUCKSEL[2:1] = 11*.

*Bits 7, 6 and 4 of this register can be written in initialization mode only (RTC\_ISR/INITF = 1).*

*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 657](#).*

## 22.6.4 RTC initialization and status register (RTC\_ISR)

Address offset: 0x0C

Backup domain reset value: 0x0000 0007

System reset value: Not affected except INIT, INITF and RSF which are cleared to 0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RECALPF
																r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	TAMP2F	TAMP1F	TSOVF	TSF	WUTF	ALRBF	ALRAF	INIT	INITF	RSF	INITS	SHPF	WUT WF	ALRB WF	ALRA WF	
	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rw	r	rc_w0	r	r	r	r	r	

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

Bit 16 **RECALPF**: Recalibration pending Flag

The RECALPF status flag is automatically set to '1' when software writes to the RTC\_CALR register, indicating that the RTC\_CALR register is blocked. When the new calibration settings are taken into account, this bit returns to '0'. Refer to [Re-calibration on-the-fly](#).

Bit 15 Reserved, must be kept at reset value.

Bit 14 **TAMP2F**: TAMPER2 detection flag

This flag is set by hardware when a tamper detection event is detected on tamper input 2.  
It is cleared by software writing 0.

Bit 13 **TAMP1F**: Tamper detection flag

This flag is set by hardware when a tamper detection event is detected.  
It is cleared by software writing 0.

Bit 12 **TSOVF**: Timestamp overflow flag

This flag is set by hardware when a timestamp 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 timestamp event occurs immediately before the TSF bit is cleared.

Bit 11 **TSF**: Timestamp flag

This flag is set by hardware when a timestamp event occurs.  
This flag is cleared by software by writing 0.

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.  
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 value 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. It is cleared by hardware when the corresponding shift operation has been executed. Writing to SHPF 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. It 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 ALRBIE 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 ALRAF, ALRBF, WUTF and TSF bits are cleared 2 APB clock cycles after programming them to 0.

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 657](#).

## 22.6.5 RTC prescaler register (RTC\_PRER)

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]														
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:

$$\text{ck_spre frequency} = \text{ck_apre frequency}/(\text{PREDIV\_S}+1)$$

**Note:** 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 657](#)

This register is write protected. The write access procedure is described in [RTC register write protection on page 657](#).

## 22.6.6 RTC wakeup timer register (RTC\_WUTR)

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.

**Note:** 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.

**Note:** 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 657](#).

## 22.6.7 RTC calibration register (RTC\_CALIBR)

Address offset: 0x18

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

Bits 31:8 Reserved, must be kept at reset value

Bit 7 **DCS**: Digital calibration sign

- 0: Positive calibration: calendar update frequency is increased
- 1: Negative calibration: calendar update frequency is decreased

Bits 6:5 Reserved, must be kept at reset value.

Bits 4:0 **DC[4:0]**: Digital calibration

DCS = 0 (positive calibration)

00000: +0 ppm

00001: +4 ppm (rounded value)

00010: +8 ppm (rounded value)

..

11111: +126 ppm (rounded value)

DCS = 1 (negative calibration)

00000: -0 ppm

00001: -2 ppm (rounded value)

00010: -4 ppm (rounded value)

..

11111: - 63 ppm (rounded value)

Refer to [Case of RTCCLK=32.768 kHz and PREDIV\\_A+1=128](#) for the exact step value.

**Note:** This register can be written in initialization mode only (RTC\_ISR/INITF = '1').

This register is write protected. The write access procedure is described in [RTC register write protection on page 657](#).

## 22.6.8 RTC alarm A register (RTC\_ALRMAR)

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.

**Note:** 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 657](#).

## 22.6.9 RTC alarm B register (RTC\_ALRMBR)

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

**Note:** 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 657](#).

### 22.6.10 RTC write protection register (RTC\_WPR)

Address offset: 0x24

Backup domain 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.

### 22.6.11 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.
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
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 in the synchronous prescaler's counter. The fraction of a second is given by the formula below:

$$\text{Second fraction} = (\text{PREDIV\_S} - \text{SS}) / (\text{PREDIV\_S} + 1)$$

*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.*

### 22.6.12 RTC shift control register (RTC\_SHIFTR)

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's 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.*

Refer to [Section 22.3.8: RTC synchronization](#).

**Note:** This register is write protected. The write access procedure is described in [RTC register write protection on page 657](#)

### 22.6.13 RTC time stamp time register (RTC\_TSTR)

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	<b>HT[1:0]</b>		<b>HU[3:0]</b>			
									r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	<b>MNT[2:0]</b>			<b>MNU[3:0]</b>				Res.	<b>ST[2:0]</b>			<b>SU[3:0]</b>			
	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.

**Note:** *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.*

### 22.6.14 RTC time stamp date register (RTC\_TSDR)

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
<b>WDU[1:0]</b>			MT	<b>MU[3:0]</b>				Res.	Res.	<b>DT[1:0]</b>		<b>DU[3:0]</b>			
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

*Note:* 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.

### 22.6.15 RTC timestamp sub second register (RTC\_TSSSR)

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.
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
SS[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved

Bits 15:0 **SS**: Sub second value

SS[15:0] is the value of the synchronous prescaler's counter when the timestamp event occurred.

*Note:* 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.

### 22.6.16 RTC calibration register (RTC\_CALR)

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.
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
CALP	CALW8	CALW16	Res.	Res.	Res.	Res.	CALM[8:0]								
rw	rw	rw	r	r	r	r	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 22.3.11: 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.

CALM[1:0] are stuck at "00" when CALW8=1'.

Refer to [Section 22.3.11: 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 22.3.11: 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 22.3.11: RTC smooth digital calibration on page 662](#).

*Note:* This register is write protected. The write access procedure is described in [RTC register write protection](#).

## 22.6.17 RTC tamper and alternate function configuration register (RTC\_TAFCR)

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.	Res.	Res.	Res.	Res.	Res.	Res.	ALARMOUT TYPE	TSIN SEL	TAMP1I NSEL	
													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]				TAMPTS	Res.	Res.	TAMP2 TRG	TAMP2E	TAMPIE	TAMP1TRG	TAMP1E		
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. Always read as 0.

Bit 18 **ALARMOUTTYPE**: RTC\_ALARM output type

- 0: RTC\_ALARM is an open-drain output
- 1: RTC\_ALARM is a push-pull output

Bit 17 **TSINSEL**: TIMESTAMP mapping

- 0: RTC\_AF1 used as TIMESTAMP
- 1: RTC\_AF2 used as TIMESTAMP

Bit 16 **TAMP1INSEL**: TAMPER1 mapping

- 0: RTC\_AF1 used as TAMPER1
- 1: RTC\_AF2 used as TAMPER1

*Note:* TAMP1E must be reset when TAMP1INSEL is changed to avoid unwanted setting of TAMP1F.

Bit 15 **TAMPPUDIS**: TAMPER pull-up disable

This bit determines if each of the tamper pins are pre-charged before each sample.

- 0: Precharge tamper pins before sampling (enable internal pull-up)
- 1: Disable precharge of tamper pins

*Note:*

Bits 14:13 **TAMPPRCH[1:0]**: Tamper 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 tamper inputs.

- 0x0: 1 RTCCLK cycle
- 0x1: 2 RTCCLK cycles
- 0x2: 4 RTCCLK cycles
- 0x3: 8 RTCCLK cycles

Bits 12:11 **TAMPFLT[1:0]**: Tamper filter count

These bits determines the number of consecutive samples at the specified level (TAMP\*TRG) necessary to activate a Tamper event. TAMPFLT is valid for each of the tamper inputs.

- 0x0: Tamper is activated on edge of tamper input transitions to the active level (no internal pull-up on tamper input).
- 0x1: Tamper is activated after 2 consecutive samples at the active level.
- 0x2: Tamper is activated after 4 consecutive samples at the active level.
- 0x3: Tamper 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 tamper 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.

Bits 6:5 Reserved. Always read as 0.

Bit 4 **TAMP2TRG**: Active level for tamper 2

if TAMPFLT != 00

0: TAMPER2 staying low triggers a tamper detection event.

1: TAMPER2 staying high triggers a tamper detection event.

if TAMPFLT = 00:

0: TAMPER2 rising edge triggers a tamper detection event.

1: TAMPER2 falling edge triggers a tamper detection event.

Bit 3 **TAMP2E**: Tamper 2 detection enable

0: Tamper 2 detection disabled

1: Tamper 2 detection enabled

Bit 2 **TAMPIE**: Tamper interrupt enable

0: Tamper interrupt disabled

1: Tamper interrupt enabled

Bit 1 **TAMP1TRG**: Active level for tamper 1

if TAMPFLT != 00:

0: TAMPER1 staying low triggers a tamper detection event.

1: TAMPER1 staying high triggers a tamper detection event.

if TAMPFLT = 00:

0: TAMPER1 rising edge triggers a tamper detection event.

1: TAMPER1 falling edge triggers a tamper detection event.

**Caution:** When TAMPFLT = 0, TAMP1E must be reset when TAMP1TRG is changed to avoid spuriously setting TAMP1F.

Bit 0 **TAMP1E**: Tamper 1 detection enable

0: Tamper 1 detection disabled

1: Tamper 1 detection enabled

## 22.6.18 RTC alarm A sub second register (RTC\_ALRMASSR)

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
SS[14:0]															
Res.															
	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's counter to determine if Alarm A is to be activated. Only bits 0 up MASKSS-1 are compared.

**Note:** *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 657](#)*

### 22.6.19 RTC alarm B sub second register (RTC\_ALRMBSSR)

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.							
r	r	r	r	rw	rw	rw	rw	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.	SS[14:0]														
r	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's counter to determine if Alarm B is to be activated. Only bits 0 up to MASKSS-1 are compared.

**Note:** *This register can be written only when ALRBIE 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](#)*

## 22.6.20 RTC backup registers (RTC\_BKPxR)

Address offset: 0x50 to 0x9C

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	w	rw	rw	rw

Bits 31:0 BKP[31:0]

The application can write or read data to and from these registers.

They are powered-on by  $V_{BAT}$  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.

## 22.6.21 RTC register map

**Table 126. RTC register map and reset values**

Offset	Register	Field	Type	Description
0x00	RTC_TR	Res	Res	Reset value
		Res	Res	RTC_TRIGGER
0x04	RTC_DR	Res	Res	Reset value
		Res	Res	RTC_DATE
0x08	RTC_CR	Res	Res	Reset value
		Res	Res	RTC_CTR
0x0C	RTC_ISR	Res	Res	Reset value
		Res	Res	RTC_ISR
0x10	RTC_PRER	Res	Res	PREDIV_A[6:0]
		Res	Res	PREDIV_S[14:0]
0x14	RTC_WUTR	Res	Res	Reset value
		Res	Res	WUT[15:0]
0x18	RTC_CALIBR	Res	Res	Reset value
		Res	Res	DC[4:0]
0x1C	RTC_ALRMAR	MSK4	MSK4	DU[3:0]
		WDSEL	WDSEL	HT[1:0]
0x20	RTC_ALRMBR	0	0	DU[3:0]
		DT[1:0]	DT[1:0]	HT[1:0]
0x24	RTC_WPR	MSK3	MSK3	HU[3:0]
		PM	PM	MNT[2:0]
0x28	RTC_SSR	0	0	SS[15:0]
		0	0	KEY[7:0]
0x2C	RTC_SHIFTR	ADD1S	ADD1S	SU[3:0]
		Res	Res	SUBFS[14:0]
0x30	RTC_TMR	Res	Res	Reset value
		Res	Res	WCKSEL[2:0]
0x34	RTC_TMR	Res	Res	ST[2:0]
		Res	Res	DU[3:0]
0x38	RTC_TMR	Res	Res	DT[1:0]
		Res	Res	SU[3:0]

**Table 126. RTC register map and reset values (continued)**

Refer to [Section 2.2.2 on page 56](#) for the register boundary addresses.

**Caution:** In [Table 126](#), the reset value is the value after a backup domain reset. The majority of the registers are not affected by a system reset. For more information, refer to [Section 22.3.7: Resetting the RTC](#).

## 23 Fast-mode Plus Inter-integrated circuit (FMPI2C) interface

### 23.1 Introduction

The I<sup>2</sup>C (inter-integrated circuit) bus interface handles communications between the microcontroller and the serial I<sup>2</sup>C bus. It provides multimaster capability, and controls all I<sup>2</sup>C bus-specific sequencing, protocol, arbitration and timing. It supports Standard-mode (Sm), Fast-mode (Fm) and Fast-mode Plus (Fm+).

It is also SMBus (system management bus) and PMBus (power management bus) compatible.

DMA can be used to reduce CPU overload.

### 23.2 FMPI2C main features

- I<sup>2</sup>C bus specification rev03 compatibility:
  - Slave and master modes
  - Multimaster capability
  - Standard-mode (up to 100 kHz)
  - Fast-mode (up to 400 kHz)
  - Fast-mode Plus (up to 1 MHz)
  - 7-bit and 10-bit addressing mode
  - Multiple 7-bit slave addresses (2 addresses, 1 with configurable mask)
  - All 7-bit addresses acknowledge mode
  - General call
  - Programmable setup and hold times
  - Easy to use event management
  - Optional clock stretching
  - Software reset
- 1-byte buffer with DMA capability
- Programmable analog and digital noise filters

The following additional features are also available depending on the product implementation (see [Section 23.3: FMPI2C implementation](#)):

- SMBus specification rev 2.0 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 1.1 standard compatibility
- Independent clock: a choice of independent clock sources allowing the FMPI2C communication speed to be independent from the PCLK reprogramming

### 23.3 FMPI2C implementation

This manual describes the full set of features implemented in FMPI2C1.

**Table 127. STM32F446xx FMPI2C implementation**

I2C features <sup>(1)</sup>	FMPI2C1
Independent clock	X
SMBus	X
Wakeup from Stop mode	-

1. X = supported.

### 23.4 FMPI2C 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 I<sup>2</sup>C bus by a data pin (SDA) and by a clock pin (SCL). It can be connected with a standard (up to 100 kHz), Fast-mode (up to 400 kHz) or Fast-mode Plus (up to 1 MHz) I<sup>2</sup>C bus.

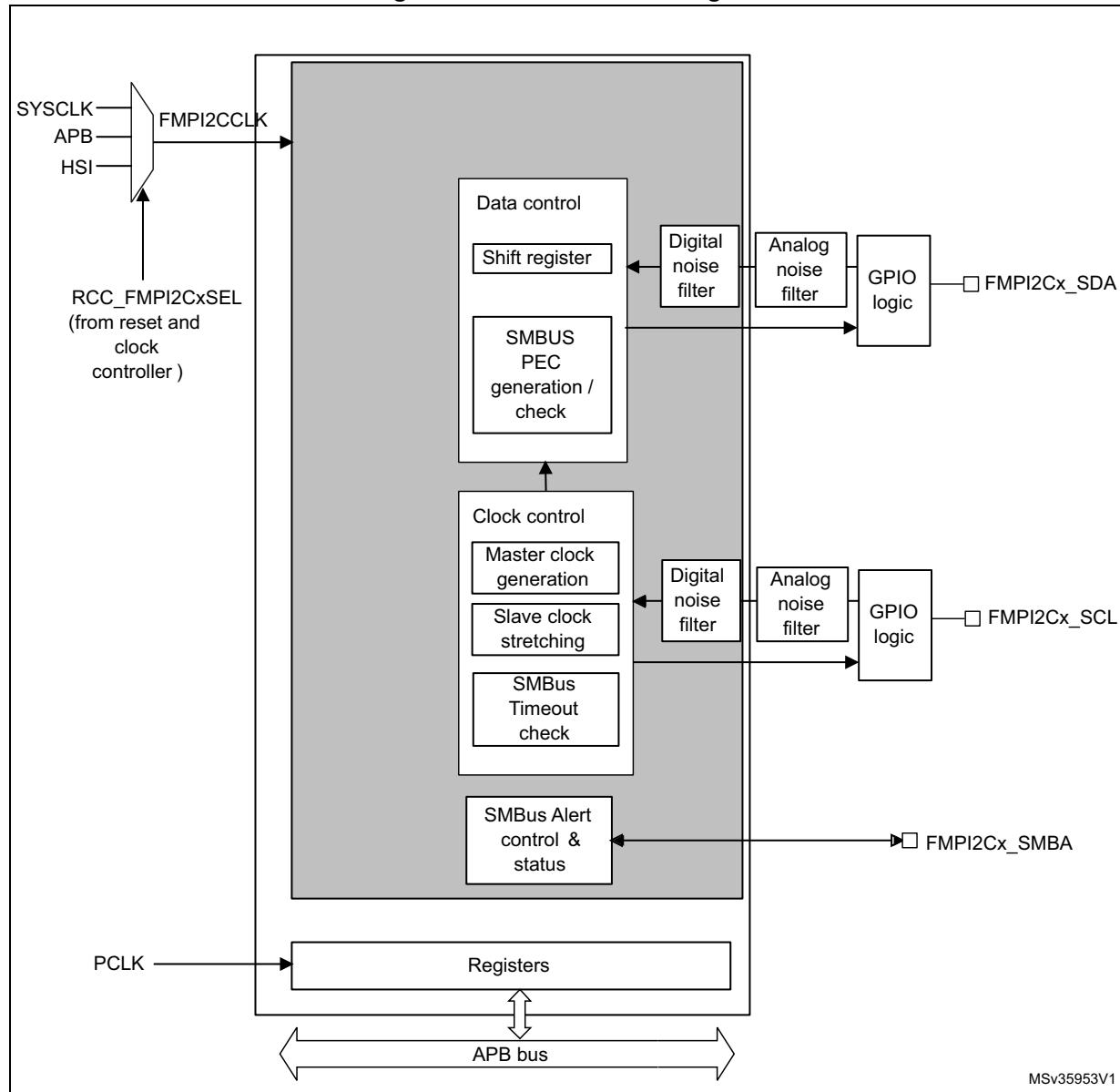
This interface can also be connected to a SMBus with the data pin (SDA) and clock pin (SCL).

If SMBus feature is supported: the additional optional SMBus Alert pin (SMBA) is also available.

### 23.4.1 FMPI2C block diagram

The block diagram of the FMPI2C interface is shown in [Figure 240](#).

**Figure 240. FMPI2C block diagram**



The FMPI2C is clocked by an independent clock source which allows the FMPI2C to operate independently from the PCLK frequency.

This independent clock source can be selected from the following three clock sources:

- PCLK1: APB1 clock (default value)
- HSI: high speed internal oscillator
- SYSCLK: system clock

Refer to [Section 6: Reset and clock control \(RCC\)](#) for more details.

### 23.4.2 FMPI2C clock requirements

The FMPI2C kernel is clocked by FMPI2CCLK.

The FMPI2CCLK 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  $\times t_{I2CCLK}$ .

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 FMPI2C kernel is clocked by PCLK, this clock must respect the conditions for  $t_{I2CCLK}$ .

### 23.4.3 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.

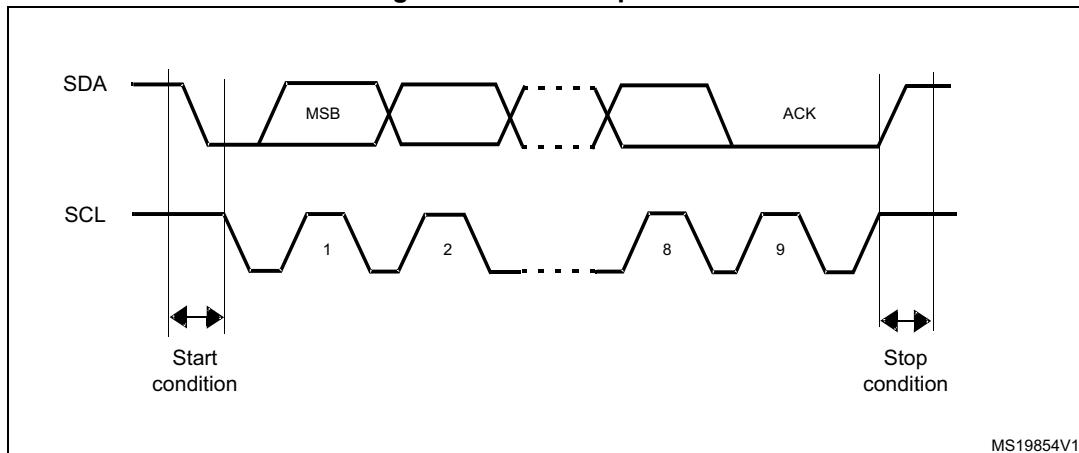
#### Communication flow

In Master mode, the FMPI2C 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 241. I<sup>2</sup>C bus protocol**

Acknowledge can be enabled or disabled by software. The FMPI2C interface addresses can be selected by software.

### 23.4.4 FMPI2C initialization

#### Enabling and disabling the peripheral

The FMPI2C peripheral clock must be configured and enabled in the clock controller (refer to [Section 6: Reset and clock control \(RCC\)](#)).

Then the FMPI2C can be enabled by setting the PE bit in the FMPI2C\_CR1 register.

When the FMPI2C is disabled (PE=0), the I<sup>2</sup>C performs a software reset. Refer to [Section 23.4.5: Software reset](#) for more details.

#### Noise filters

Before enabling the FMPI2C peripheral by setting the PE bit in FMPI2C\_CR1 register, the user must configure the noise filters, if needed. By default, an analog noise filter is present on the SDA and SCL inputs. This analog filter is compliant with the I<sup>2</sup>C specification which requires the suppression of spikes with a pulse width up to 50 ns in Fast-mode and Fast-mode Plus. The user can disable this analog filter by setting the ANFOFF bit, and/or select a digital filter by configuring the DNF[3:0] bit in the FMPI2C\_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 FMPI2CCLK periods. This allows to suppress spikes with a programmable length of 1 to 15 FMPI2CCLK periods.

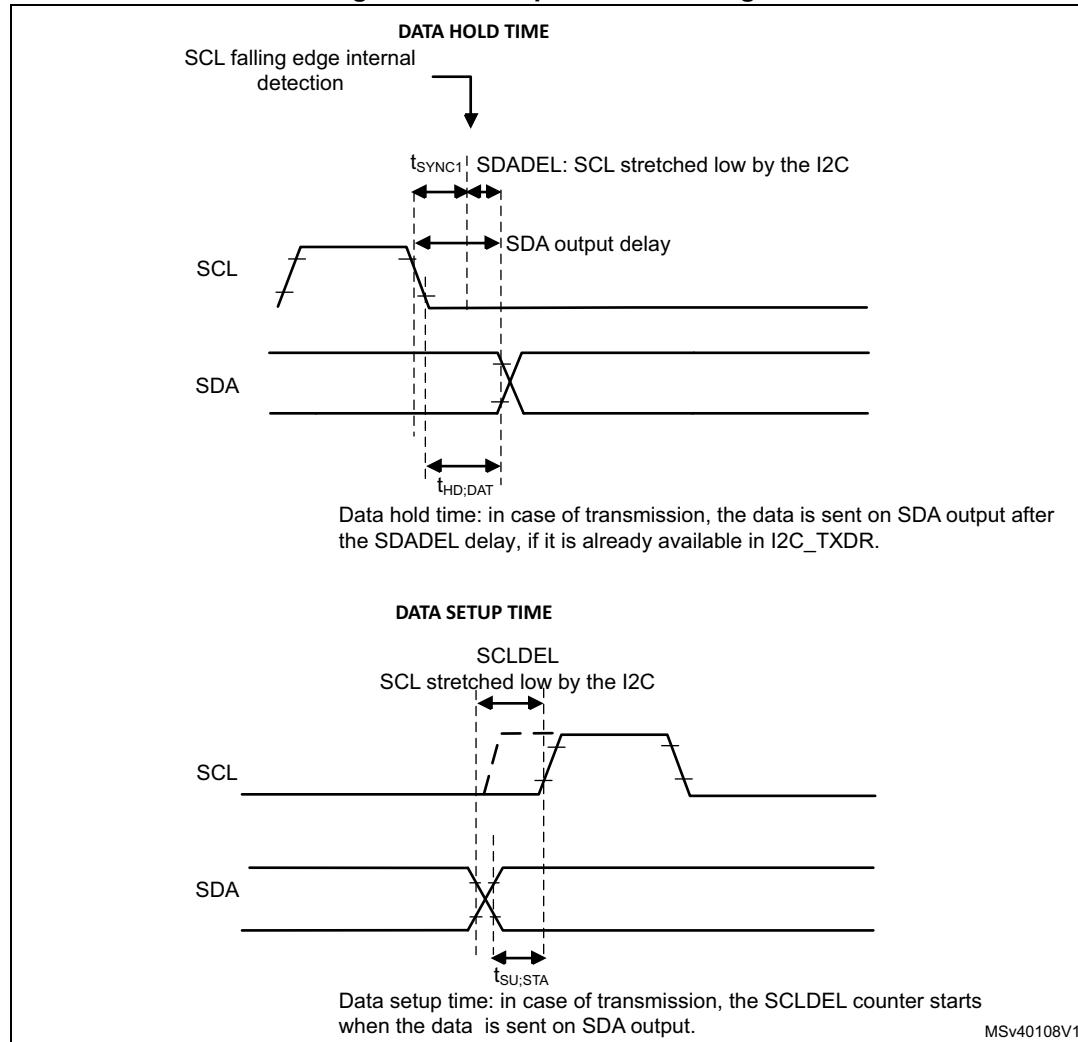
**Caution:** Changing the filter configuration is not allowed when the FMPI2C is enabled.

## FMPI2C 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 FMPI2C\_TIMINGR register.

The STM32CubeMX tool calculates and provides the I2C\_TIMINGR content in the I2C configuration window

**Figure 242. 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 FMPI2CCLK clock (2 to 3 FMPI2CCLK 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  $t_{HD;DAT}$  could 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 128: 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 128: 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 I<sup>2</sup>C 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 I<sup>2</sup>C\_TXDR when SDADEL counter is finished, the I<sup>2</sup>C 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 128. I<sup>2</sup>C-SMBUS specification data setup and hold times**

Symbol	Parameter	Standard-mode (Sm)		Fast-mode (Fm)		Fast-mode Plus (Fm+)		SMBUS		Unit
		Min.	Max	Min.	Max	Min.	Max	Min.	Max	
$t_{HD;DAT}$	<b>Data hold time</b>	0	-	0	-	0	-	0.3	-	$\mu s$
$t_{VD;DAT}$	<b>Data valid time</b>	-	3.45	-	0.9	-	0.45	-	-	
$t_{SU;DAT}$	<b>Data setup time</b>	250	-	100	-	50	-	250	-	$ns$
$t_r$	<b>Rise time of both SDA and SCL signals</b>	-	1000	-	300	-	120	-	1000	
$t_f$	<b>Fall time of both SDA and SCL signals</b>	-	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 FMPI2C\_TIMINGR register.

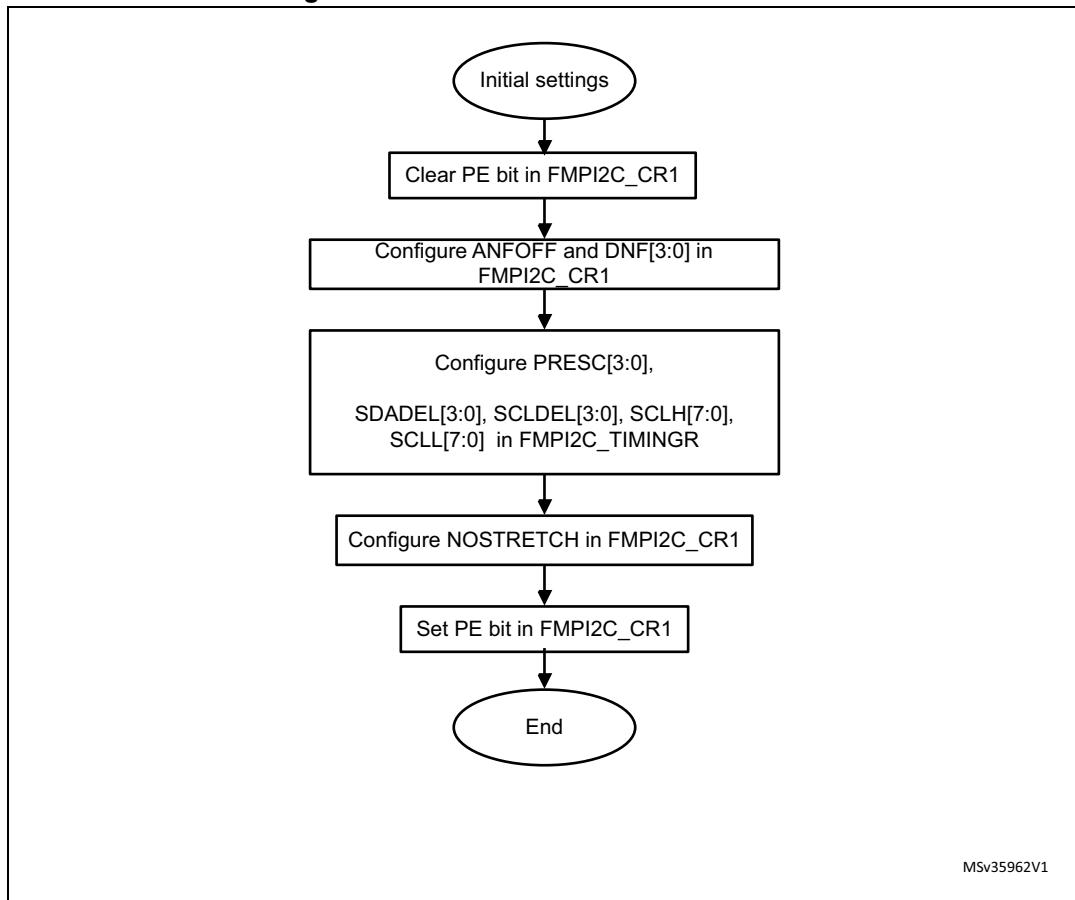
- When the SCL falling edge is internally detected, a delay is inserted before releasing the SCL output. This delay is  $t_{SCLL} = (SCLL+1) \times t_{PRESC}$  where  $t_{PRESC} = (PRESC+1) \times t_{I2CCLK}$ .  $t_{SCLL}$  impacts the SCL low time  $t_{LOW}$ .
- When the SCL rising edge is internally 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 [FMPI2C master initialization](#) for more details.

**Caution:** Changing the timing configuration is not allowed when the FMPI2C is enabled.

The FMPI2C slave NOSTRETCH mode must also be configured before enabling the peripheral. Refer to [FMPI2C slave initialization](#) for more details.

**Caution:** Changing the NOSTRETCH configuration is not allowed when the FMPI2C is enabled.

**Figure 243. FMPI2C initialization flowchart**

### 23.4.5 Software reset

A software reset can be performed by clearing the PE bit in the FMPI2C\_CR1 register. In that case FMPI2C 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. FMPI2C\_CR2 register: START, STOP, NACK
2. FMPI2C\_ISR register: BUSY, TXE, TXIS, RXNE, ADDR, NACKF, TCR, TC, STOPF, BERR, ARLO, OVR

and in addition when the SMBus feature is supported:

1. FMPI2C\_CR2 register: PECBYTE
2. FMPI2C\_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.

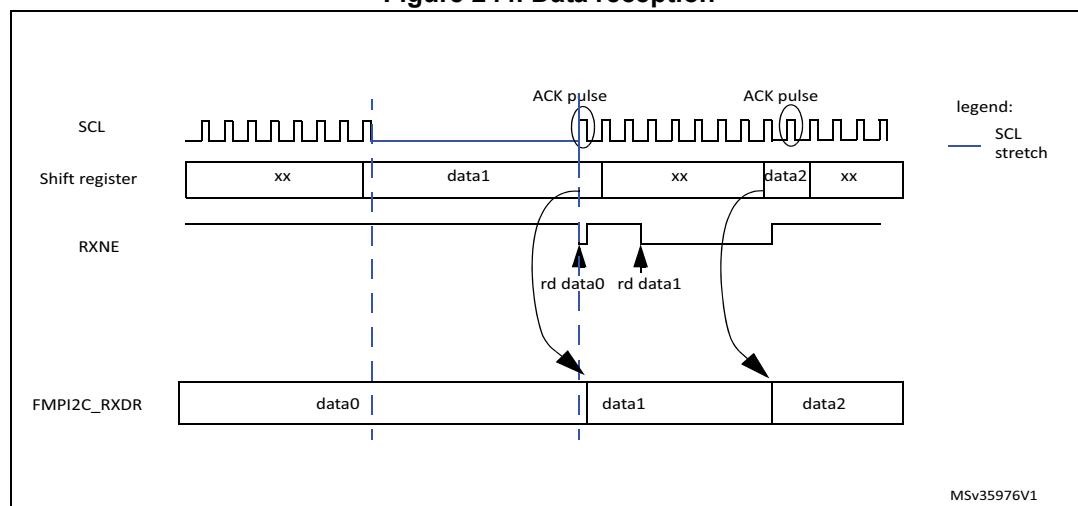
### 23.4.6 Data transfer

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 FMPI2C\_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 FMPI2C\_RXDR is read. The stretch is inserted between the 8th and 9th SCL pulse (before the Acknowledge pulse).

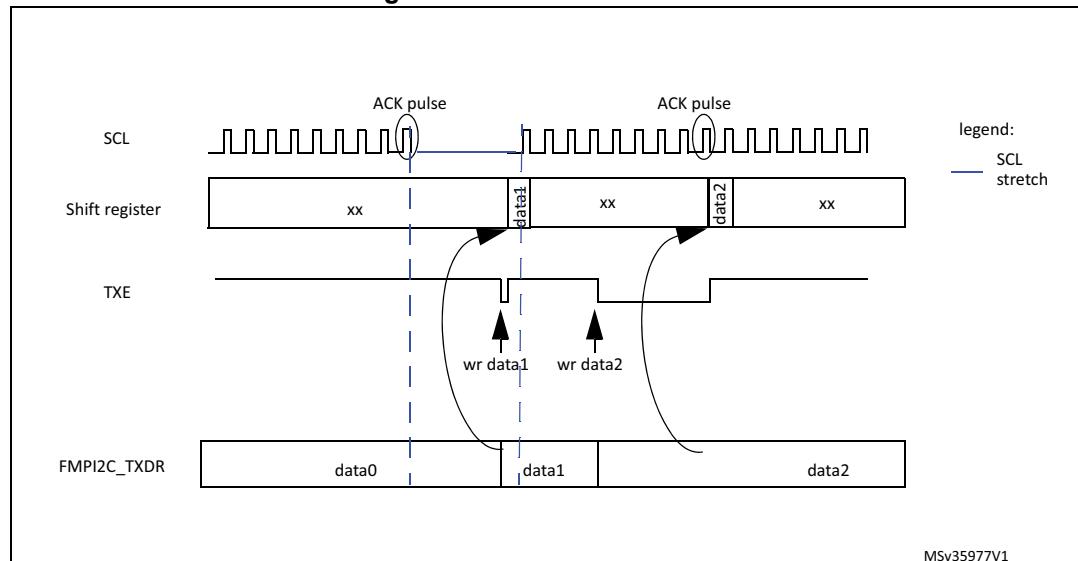
**Figure 244. Data reception**



## Transmission

If the FMPi2C\_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 FMPi2C\_TXDR, SCL line is stretched low until FMPi2C\_TXDR is written. The stretch is done after the 9th SCL pulse.

**Figure 245. Data transmission**



## Hardware transfer management

The FMPi2C 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 FMPi2C\_CR2 register.

The number of bytes to be transferred is programmed in the NBYTES[7:0] bit field in the FMPi2C\_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 FMPi2C\_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 FMPI2C\_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 FMPI2C\_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 FMPI2C\_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 129. FMPI2C configuration table**

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

### 23.4.7 FMPI2C slave mode

#### FMPI2C slave initialization

In order to work in slave mode, the user must enable at least one slave address. Two registers FMPI2C\_OAR1 and FMPI2C\_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 FMPI2C\_OAR1 register.  
OA1 is enabled by setting the OA1EN bit in the FMPI2C\_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 FMPI2C\_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 FMPI2C 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 FMPI2C\_OAR1 or FMPI2C\_OAR2 register with OA2MSK=0.

OA2 is enabled by setting the OA2EN bit in the FMPI2C\_OAR2 register.

- The General Call address is enabled by setting the GCEN bit in the FMPI2C\_CR1 register.

When the FMPI2C 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 FMPI2C must be configured with NOSTRETCH=1 in the FMPI2C\_CR1 register.

After receiving an ADDR interrupt, if several addresses are enabled the user must read the ADDCODE[6:0] bits in the FMPI2C\_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 FMPI2C 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 FMPI2C\_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 FMPI2C\_TXDR register.
- In reception when the FMPI2C\_RXDR register is not read yet and a new data reception is completed. This stretch is released when FMPI2C\_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 FMPI2C 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 FMPI2C\_CR1 register, the FMPI2C 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 FMPI2C\_TXDR register before the first SCL pulse corresponding to its transfer occurs. If not, an underrun occurs, the OVR flag is set in the FMPI2C\_ISR register and an interrupt is generated if the ERRIE bit is set in the FMPI2C\_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 FMPI2C\_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 FMPI2C\_ISR register and an interrupt is generated if the ERRIE bit is set in the FMPI2C\_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 FMP12C\_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 FMP12C\_RXDR register, and then decide to acknowledge it or not by configuring the ACK bit in the FMP12C\_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.

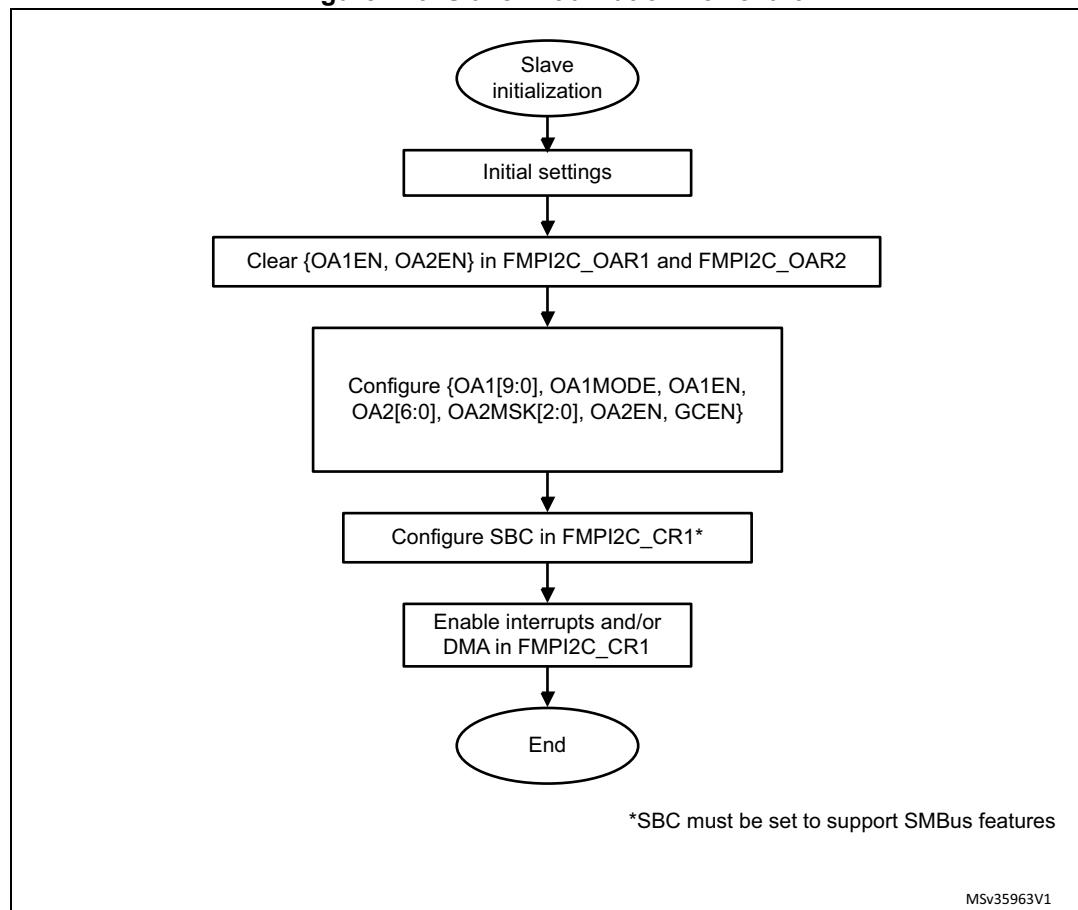
NBYTES can be loaded with a value greater than 0x1, and in this case, the reception flow is continuous during NBYTES data reception.

**Note:** *The SBC bit must be configured when the FMP12C 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 246. Slave initialization flowchart**



### Slave transmitter

A transmit interrupt status (TXIS) is generated when the FMP12C\_TXDR register becomes empty. An interrupt is generated if the TXIE bit is set in the FMP12C\_CR1 register.

The TXIS bit is cleared when the FMP12C\_TXDR register is written with the next data byte to be transmitted.

When a NACK is received, the NACKF bit is set in the FMP12C\_ISR register and an interrupt is generated if the NACKIE bit is set in the FMP12C\_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 FMP12C\_CR1 register, the STOPF flag is set in the FMP12C\_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 FMP12C\_TXDR register as the first data byte, or to flush the FMP12C\_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.

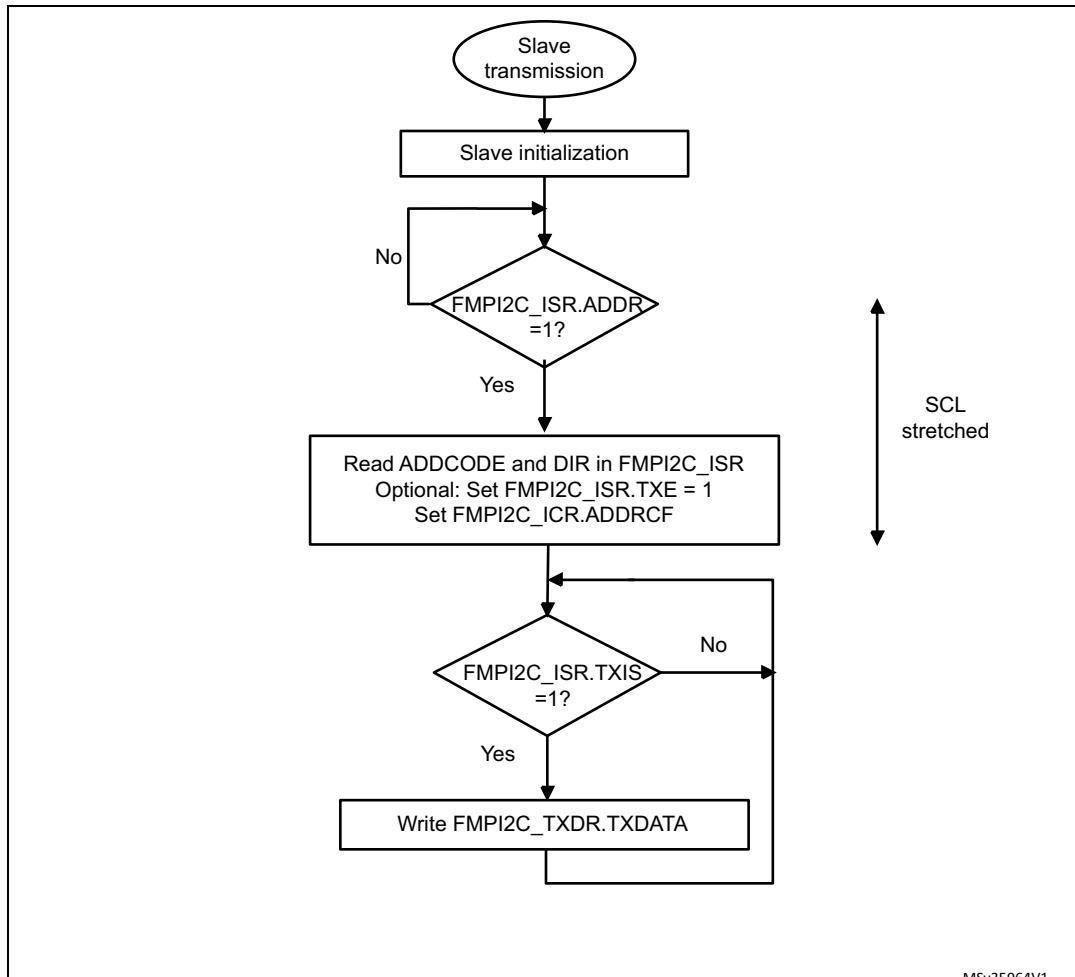
**Caution:** When NOSTRETCH=1, the SCL clock is not stretched while the ADDR flag is set, so the user cannot flush the FMP12C\_TXDR register content in the ADDR subroutine, in order to program the first data byte. The first data byte to be sent must be previously programmed in the FMP12C\_TXDR register:

- 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 FMP12C\_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.

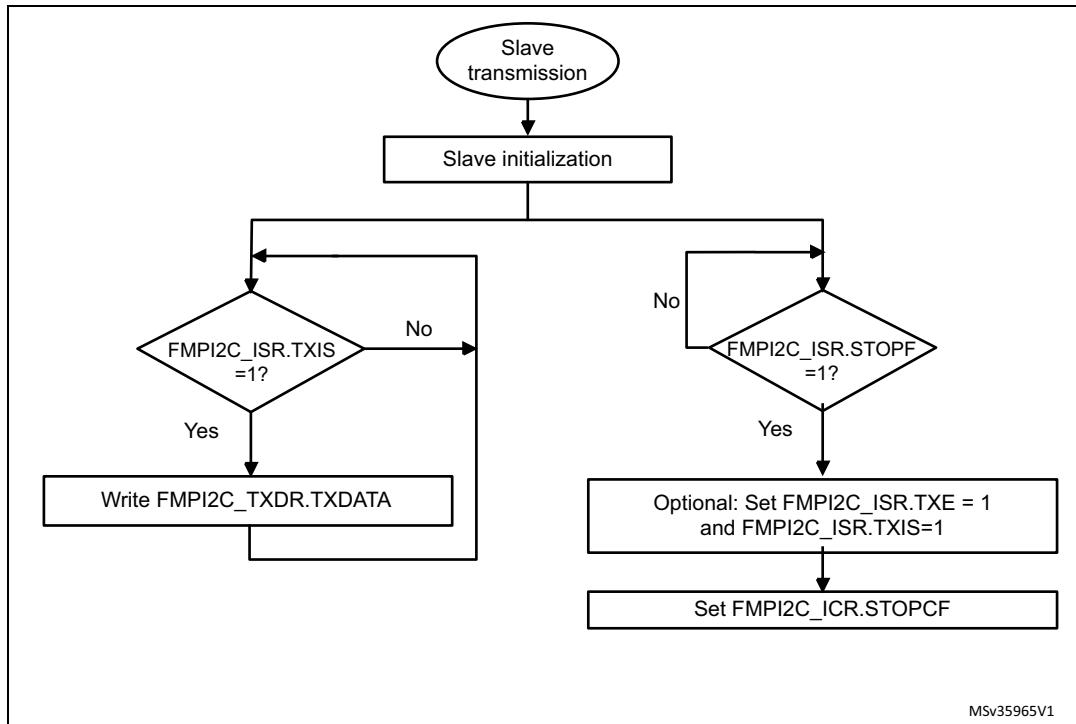
If STOPF is still set when the first data transmission starts, an underrun error will be generated (the OVR flag is set).

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 247. Transfer sequence flowchart for FMPI2C slave transmitter,  
NOSTRETCH=0

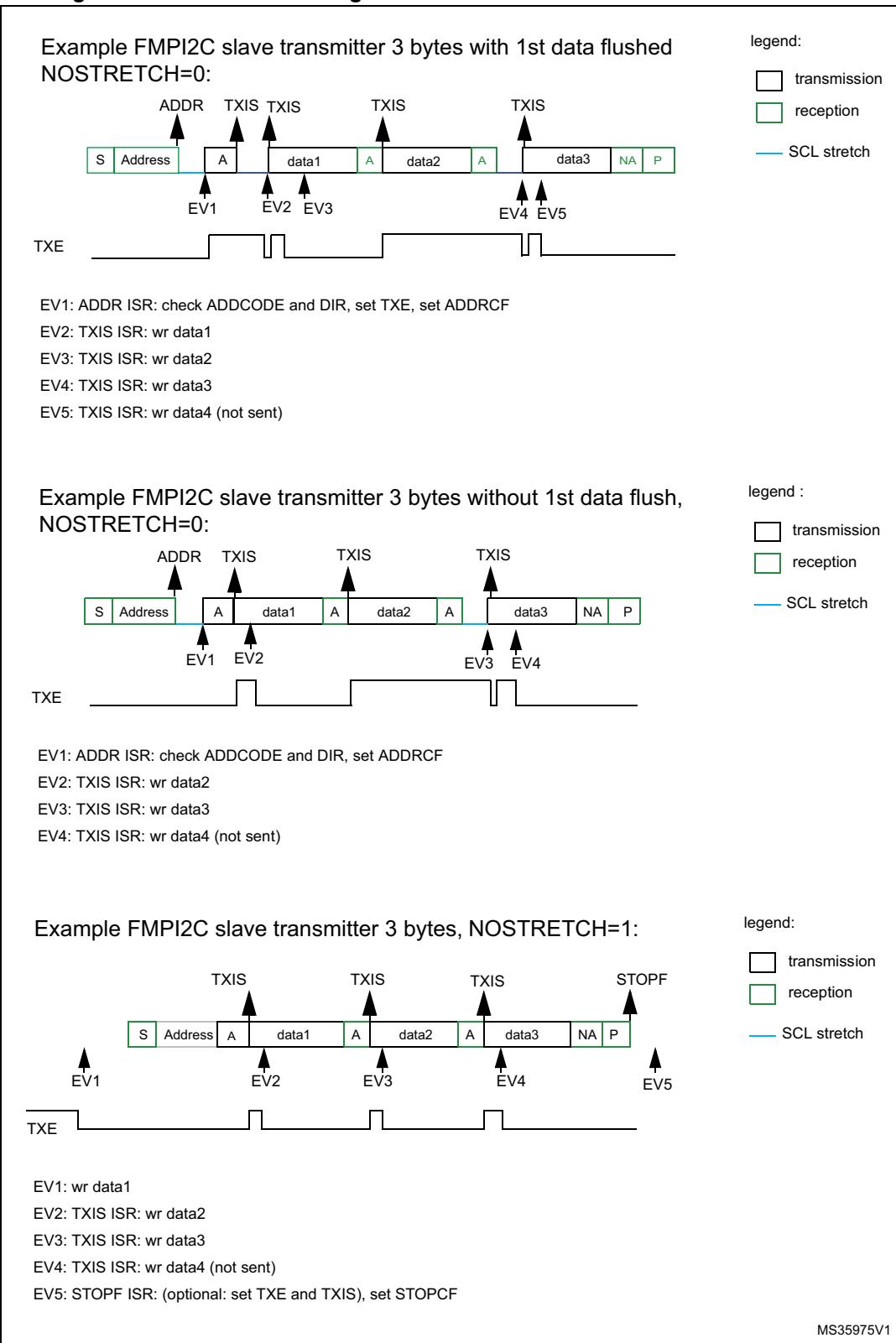


**Figure 248. Transfer sequence flowchart for FMPI2C slave transmitter,  
NOSTRETCH=1**



MSv35965V1

Figure 249. Transfer bus diagrams for FMPI2C slave transmitter



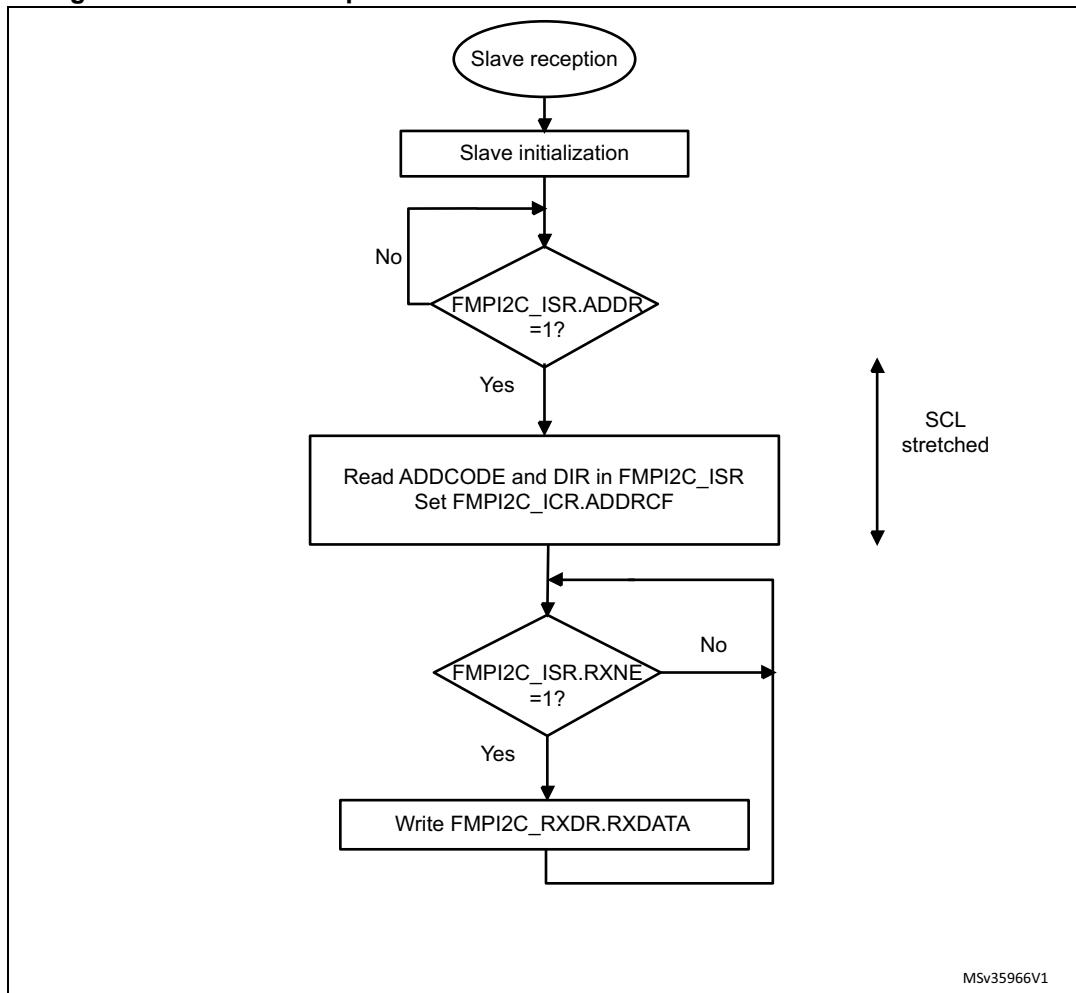
MS35975V1

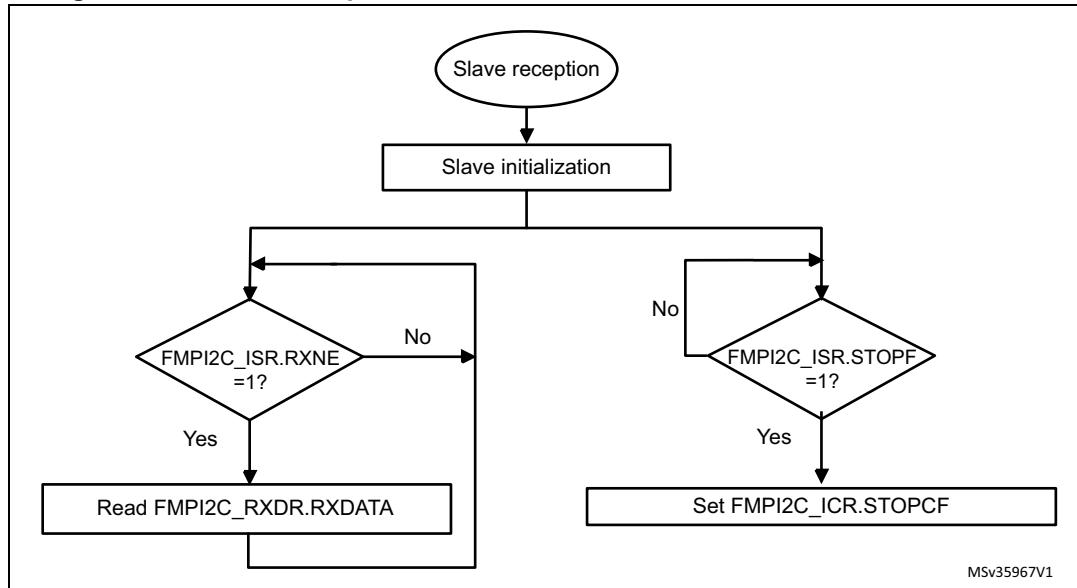
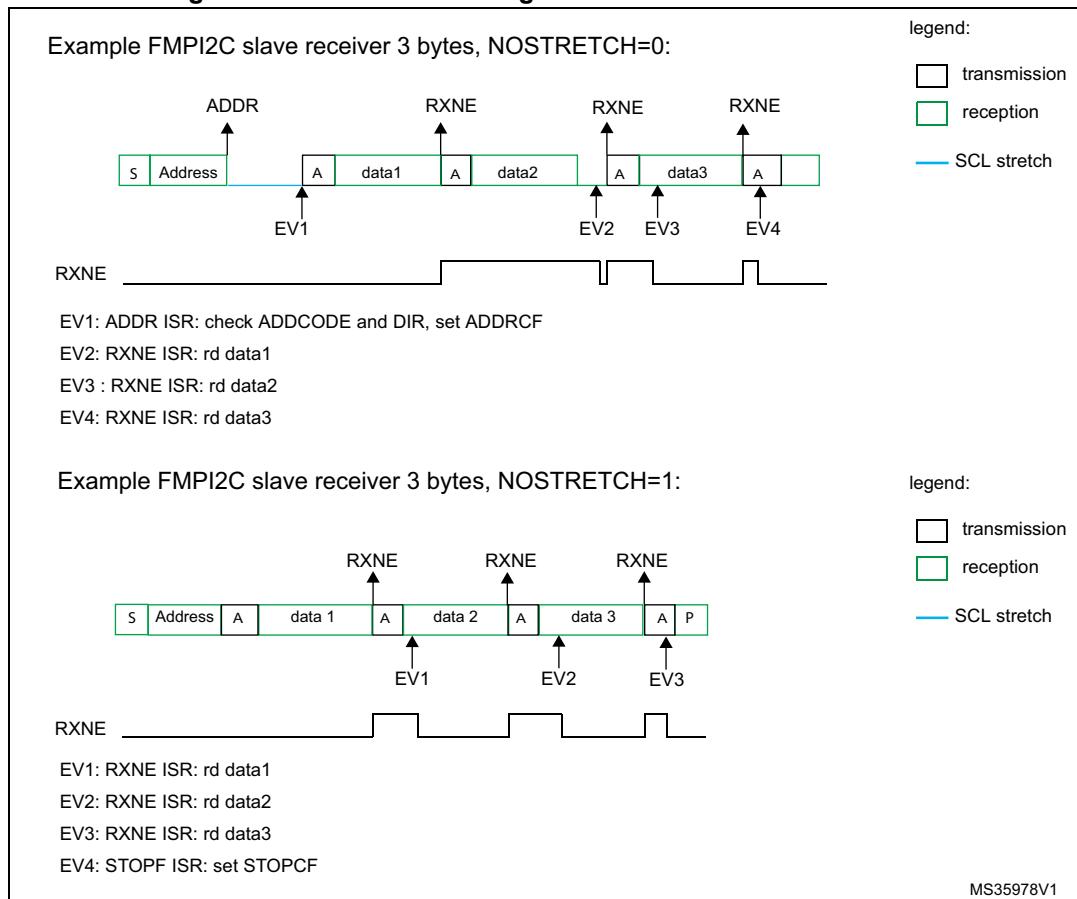
### Slave receiver

RXNE is set in FMPi2C\_ISR when the FMPi2C\_RXDR is full, and generates an interrupt if RXIE is set in FMPi2C\_CR1. RXNE is cleared when FMPi2C\_RXDR is read.

When a STOP is received and STOPIE is set in FMPi2C\_CR1, STOPF is set in FMPi2C\_ISR and an interrupt is generated.

**Figure 250. Transfer sequence flowchart for slave receiver with NOSTRETCH=0**



**Figure 251. Transfer sequence flowchart for slave receiver with NOSTRETCH=1****Figure 252. Transfer bus diagrams for FMPI2C slave receiver**

### 23.4.8 FMPI2C master mode

#### FMPI2C master initialization

Before enabling the peripheral, the FMPI2C master clock must be configured by setting the SCLH and SCLL bits in the FMPI2C\_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 FMPI2C 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 FMPI2C releases SCL to high level once the SCLL counter reaches the value programmed in the SCLL[7:0] bits in the FMPI2C\_TIMINGR register.

The FMPI2C 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 FMPI2C ties SCL to low level once the SCLH counter is reached reaches the value programmed in the SCLH[7:0] bits in the FMPI2C\_TIMINGR register.

Consequently the master clock period is:

$$t_{SCL} = t_{SYNC1} + t_{SYNC2} + \{[(SCLH+1) + (SCLL+1)] \times (PRESC+1) \times t_{I2CCLK}\}$$

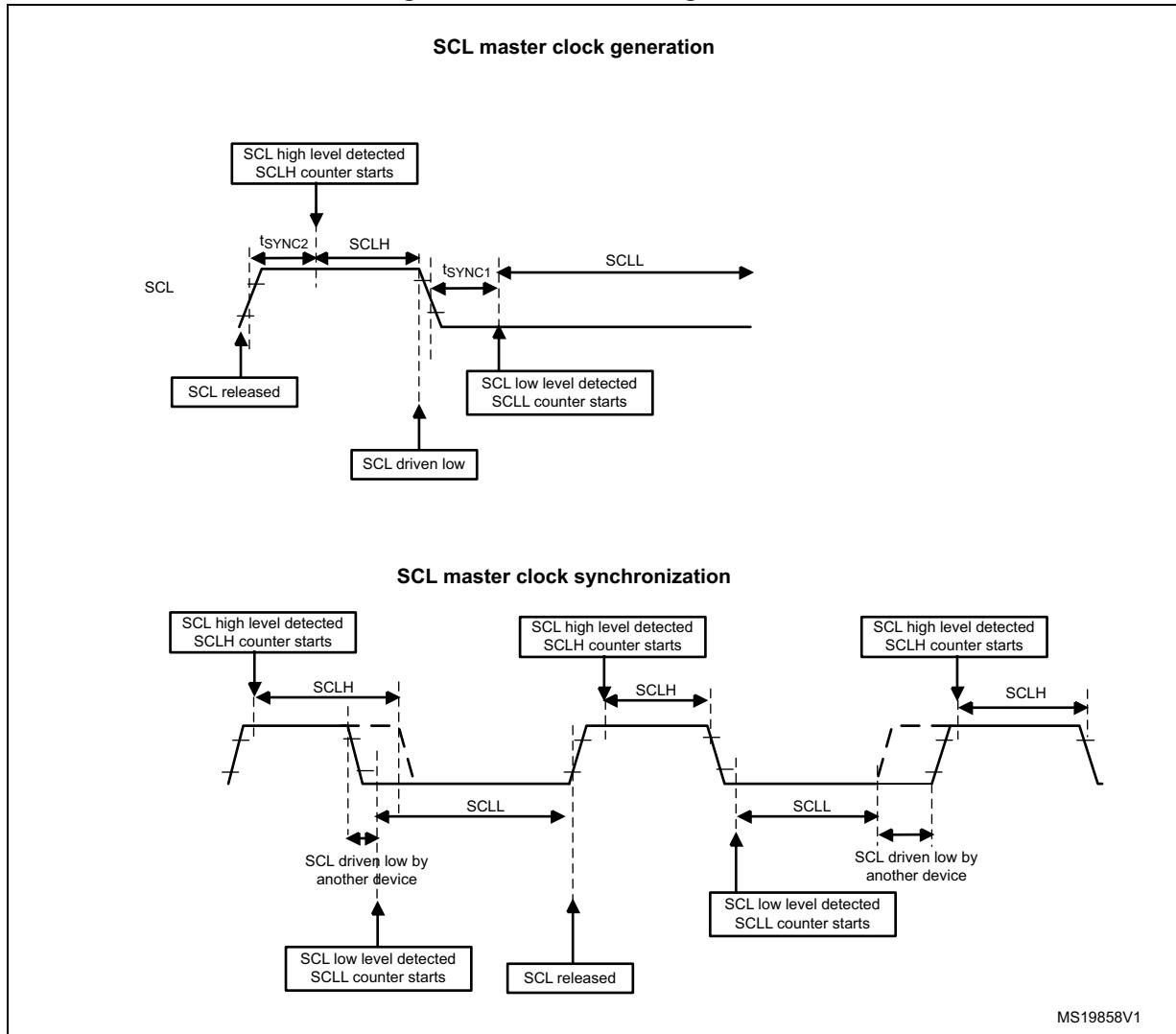
The duration of  $t_{SYNC1}$  depends on these parameters:

- 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 FMPI2CCLK clock (2 to 3 FMPI2CCLK 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 FMPI2CCLK clock (2 to 3 FMPI2CCLK periods)

Figure 253. Master clock generation



**Caution:** In order to be I<sup>2</sup>C or SMBus compliant, the master clock must respect the timings given below:

**Table 130. I<sup>2</sup>C-SMBUS specification clock timings**

Symbol	Parameter	Standard-mode (Sm)		Fast-mode (Fm)		Fast-mode Plus (Fm+)		SMBUS		Unit
		Min	Max	Min	Max	Min	Max	Min	Max	
f <sub>SCL</sub>	SCL clock frequency	-	100	-	400	-	1000	-	100	kHz
t <sub>HD:STA</sub>	Hold time (repeated) START condition	4.0	-	0.6	-	0.26	-	4.0	-	μs
t <sub>SU:STA</sub>	Set-up time for a repeated START condition	4.7	-	0.6	-	0.26	-	4.7	-	μs
t <sub>SU:STO</sub>	Set-up time for STOP condition	4.0	-	0.6	-	0.26	-	4.0	-	μs
t <sub>BUF</sub>	Bus free time between a STOP and START condition	4.7	-	1.3	-	0.5	-	4.7	-	μs
t <sub>LOW</sub>	Low period of the SCL clock	4.7	-	1.3	-	0.5	-	4.7	-	μs
t <sub>HIGH</sub>	Period of the SCL clock	4.0	-	0.6	-	0.26	-	4.0	50	μs
t <sub>r</sub>	Rise time of both SDA and SCL signals	-	1000	-	300	-	120	-	1000	ns
t <sub>f</sub>	Fall time of both SDA and SCL signals	-	300	-	300	-	120	-	300	ns

Note: SCLL is also used to generate the t<sub>BUF</sub> and t<sub>SU:STA</sub> timings.

SCLH is also used to generate the t<sub>HD:STA</sub> and t<sub>SU:STO</sub> timings.

Refer to [Section 23.4.9: FMPI2C\\_TIMINGR register configuration examples](#) for examples of FMPI2C\_TIMINGR settings vs. FMPI2CCLK 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 FMPI2C\_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 FMPI2C\_CR2 register. Changing all the above bits is not allowed when START bit is set.

Then the master automatically sends the START condition followed by the slave address as soon as it detects that the bus is free (BUSY = 0) and after a delay of t<sub>BUF</sub>.

In case of an arbitration loss, the master automatically switches back to slave mode and can acknowledge its own address if it is addressed as a slave.

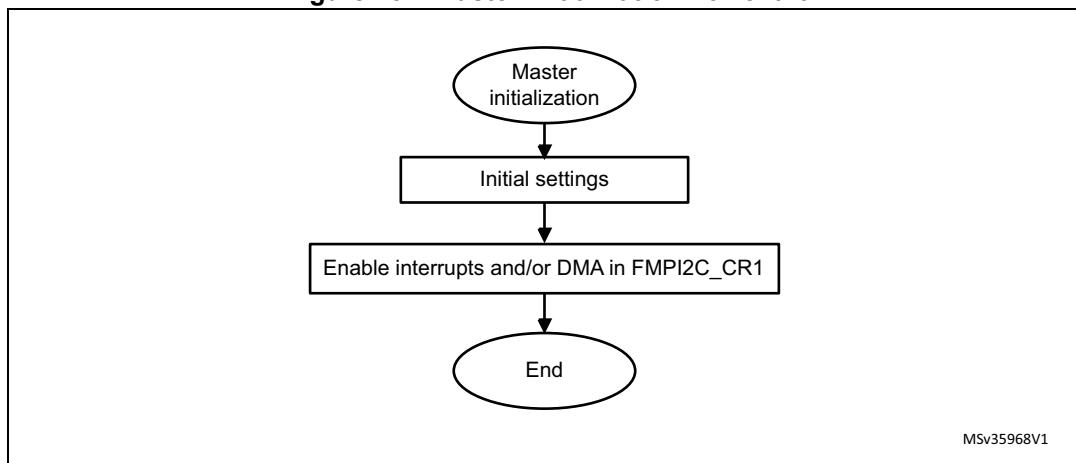
**Note:** The START bit is reset by hardware when the slave address has been sent on the bus, whatever the received acknowledge value. The START bit is also reset by hardware if an arbitration loss occurs.

In 10-bit addressing mode, when the Slave Address first 7 bits is NACKed by the slave, the master will re-launch 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 FMPI2C is addressed as a slave (ADDR=1) while the START bit is set, the FMPI2C 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.

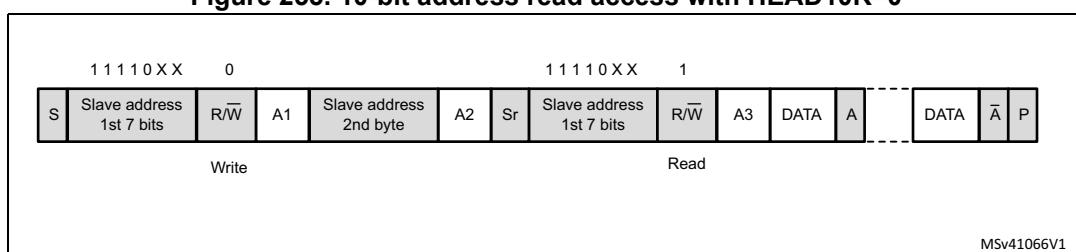
**Figure 254. Master initialization flowchart**



#### Initialization of a master receiver addressing a 10-bit address slave

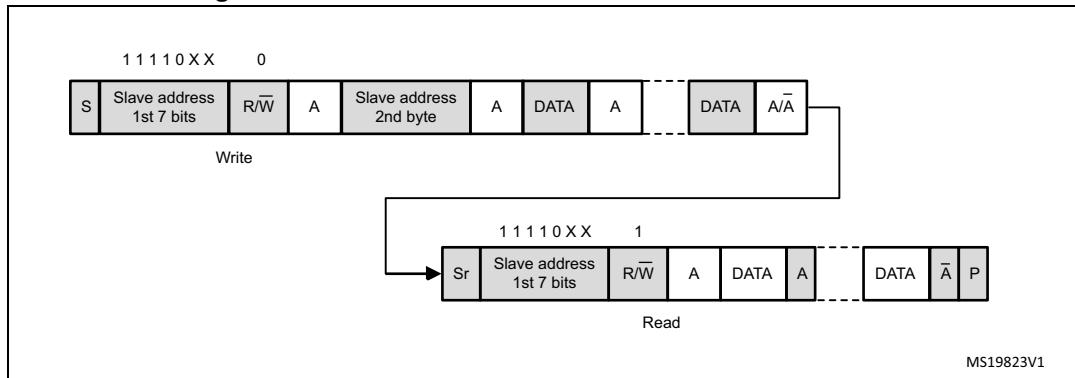
- If the slave address is in 10-bit format, the user can choose to send the complete read sequence by clearing the HEAD10R bit in the FMPI2C\_CR2 register. In this case the master automatically sends the following complete sequence after the START bit is set: (Re)Start + Slave address 10-bit header Write + Slave address 2nd byte + REStart + Slave address 10-bit header Read

**Figure 255. 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 256. 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.

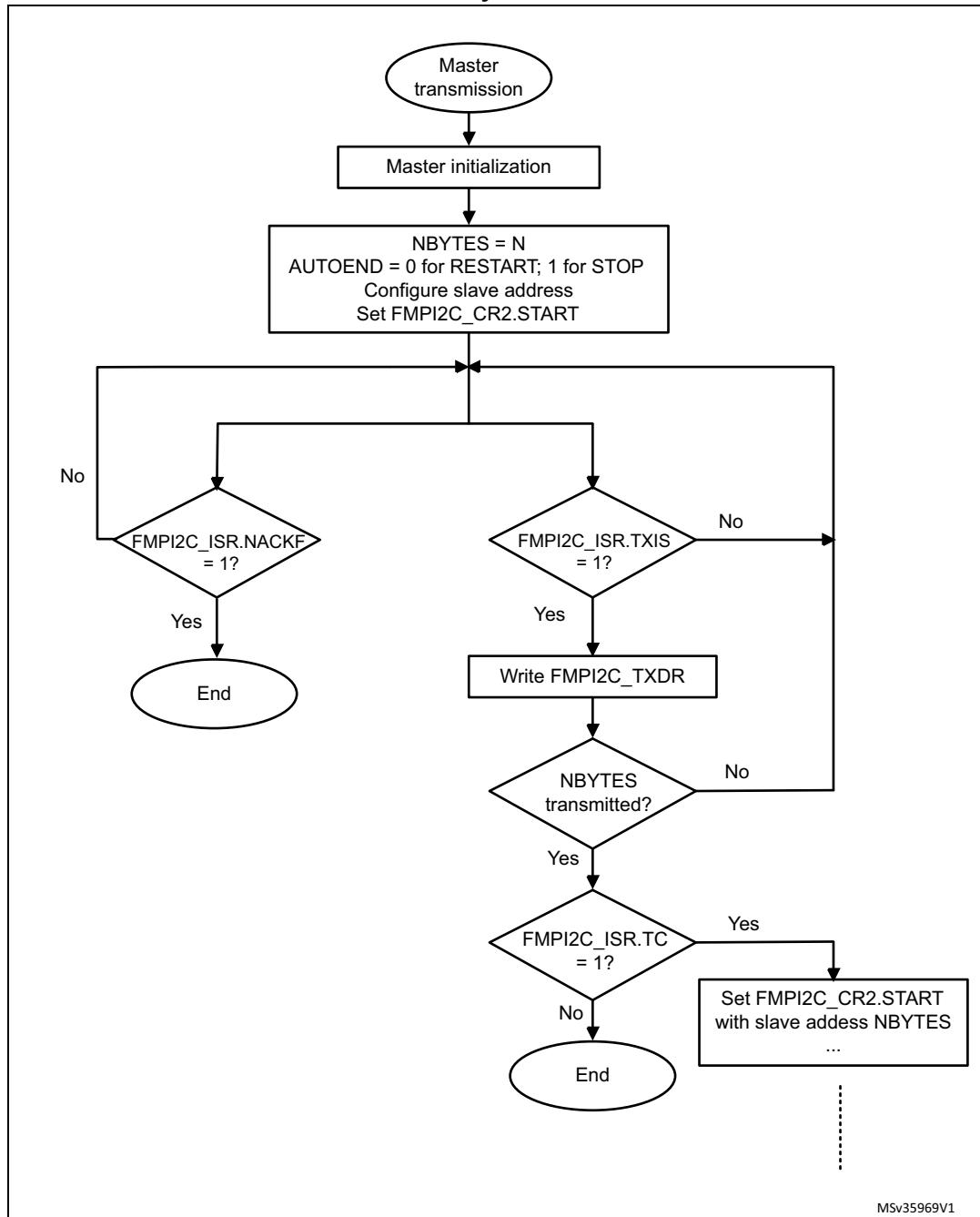
A TXIS event generates an interrupt if the TXIE bit is set in the FMP12C\_CR1 register. The flag is cleared when the FMP12C\_TXDR register is written with the next data byte to be transmitted.

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 FMP12C\_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.

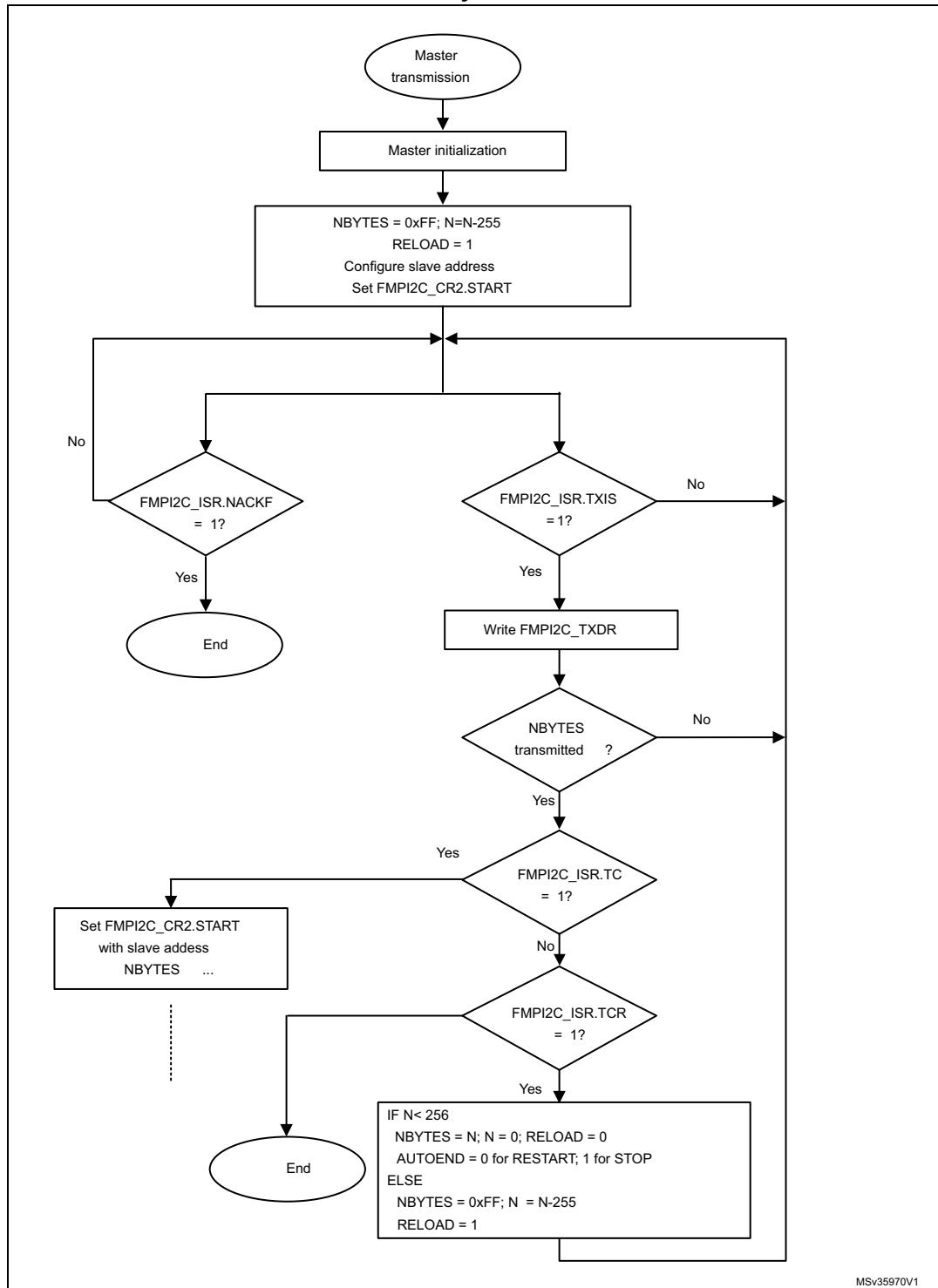
The TXIS flag is not set when a NACK is received.

- 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 FMP12C\_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 FMP12C\_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 FMP12C\_ISR register, and an interrupt is generated if the NACKIE bit is set.

**Figure 257. Transfer sequence flowchart for FMPi2C master transmitter for  $N \leq 255$  bytes**



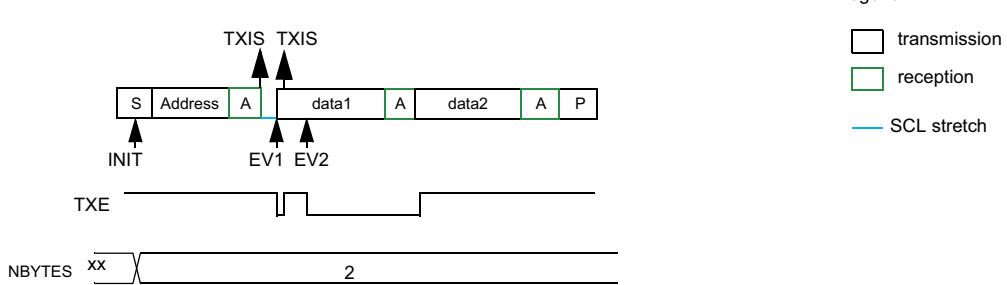
**Figure 258. Transfer sequence flowchart for FMPI2C master transmitter for N>255 bytes**



MSv35970V1

**Figure 259. Transfer bus diagrams for FMPI2C master transmitter**

Example FMPI2C master transmitter 2 bytes, automatic end mode (STOP)

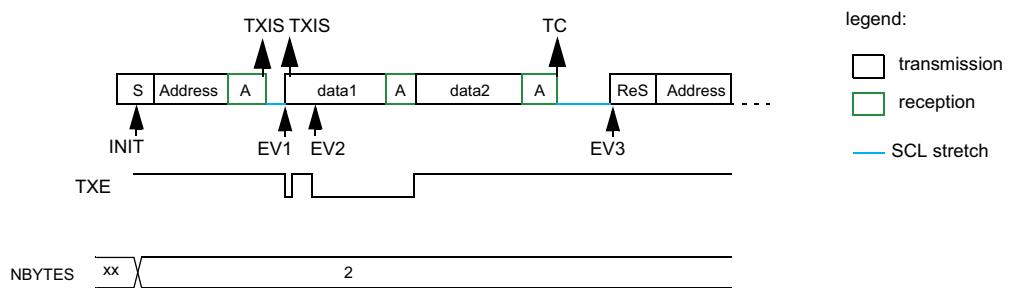


INIT: program Slave address, program NBYTES = 2, AUTOEND=1, set START

EV1: TXIS ISR: wr data1

EV2: TXIS ISR: wr data2

Example FMPI2C master transmitter 2 bytes, software end mode (RESTART)



INIT: program Slave address, program NBYTES = 2, AUTOEND=0, set START

EV1: TXIS ISR: wr data1

EV2: TXIS ISR: wr data2

EV3: TC ISR: program Slave address, program NBYTES = N, set START

MS35980V1

### 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 FMPI2C\_CR1 register. The flag is cleared when FMPI2C\_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 FMPI2C\_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 FMPI2C\_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 FMPI2C\_CR2 register. Setting the STOP bit clears the TC flag and the STOP condition is sent on the bus.

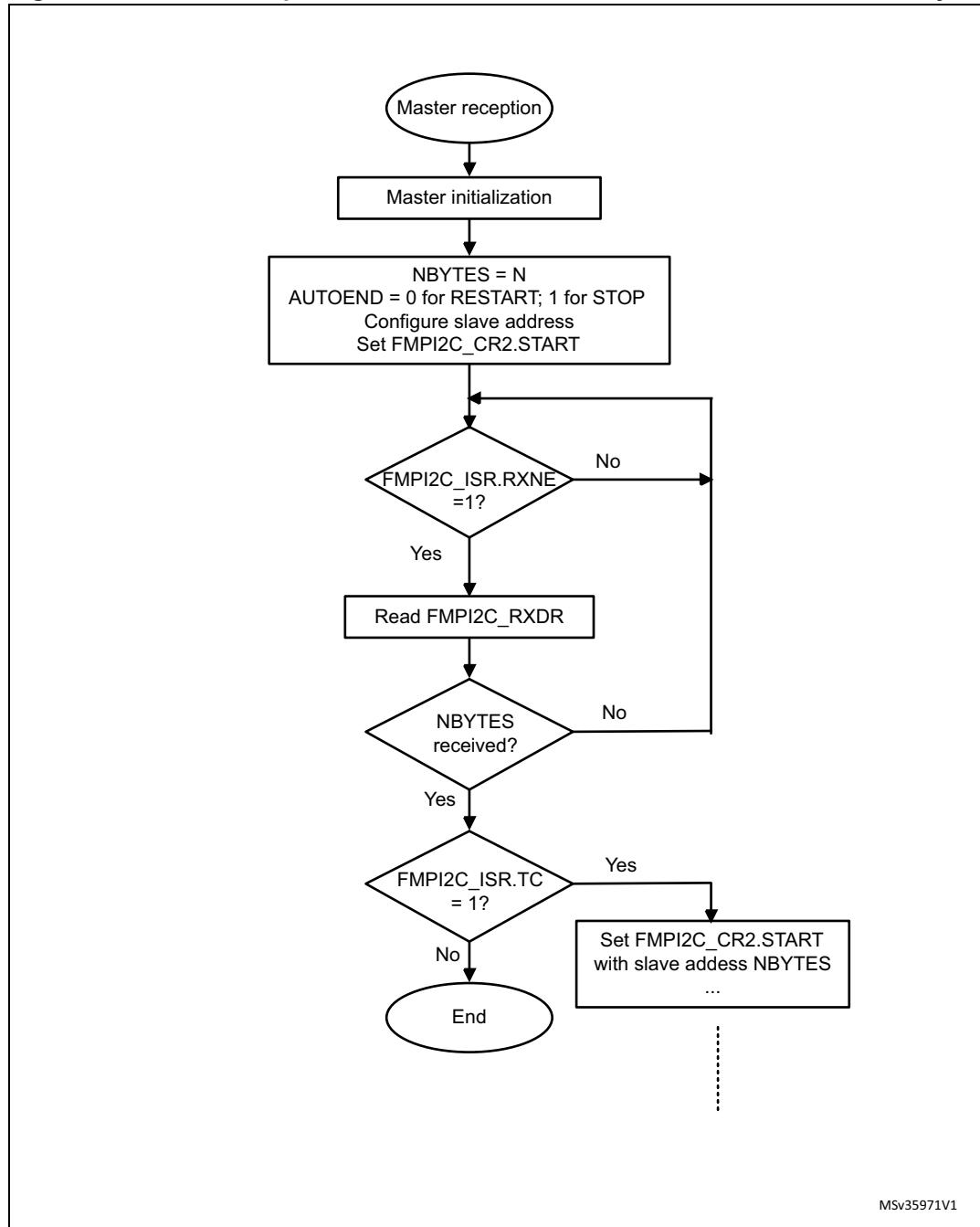
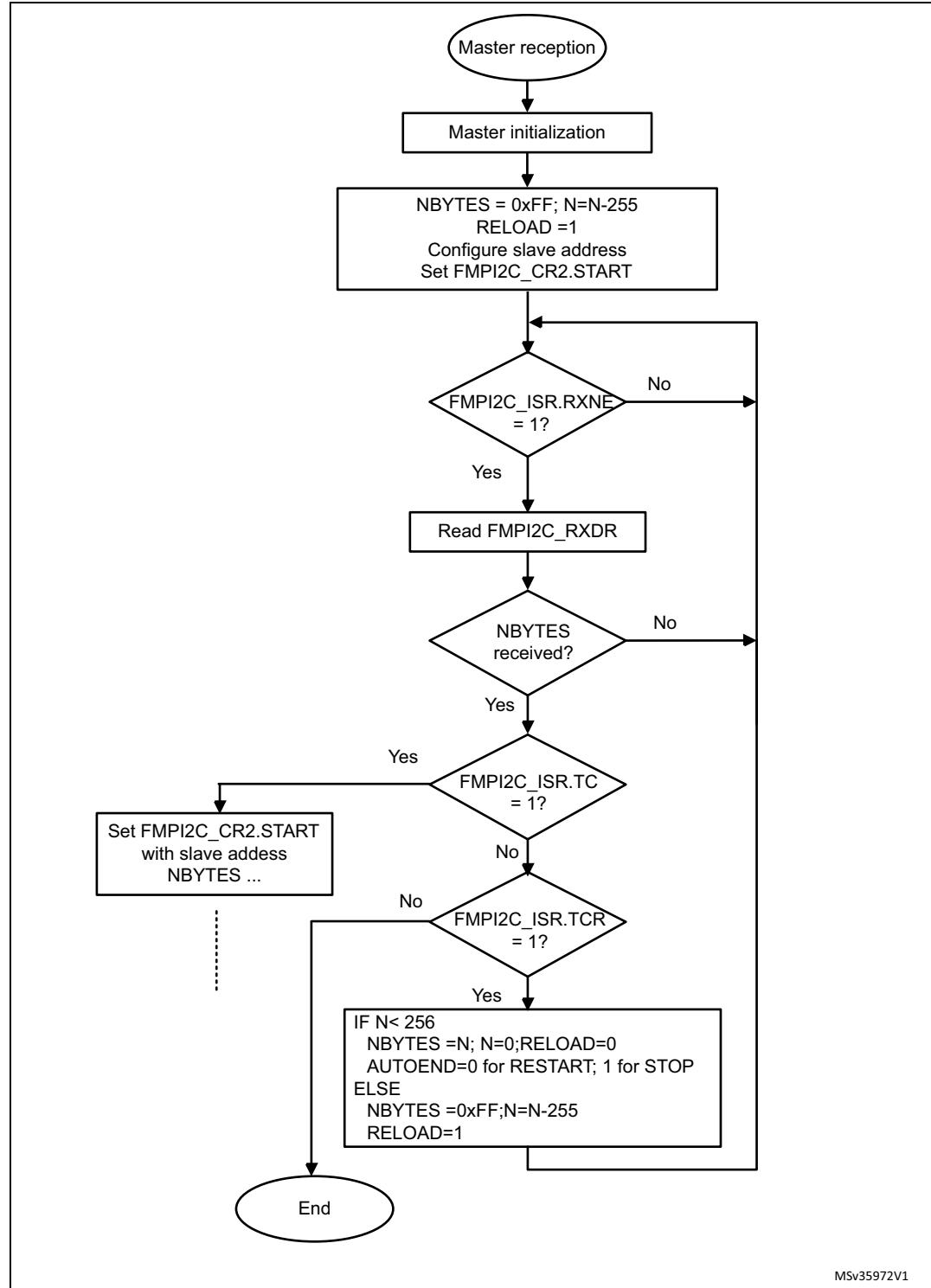
Figure 260. Transfer sequence flowchart for FMPI2C master receiver for  $N \leq 255$  bytes

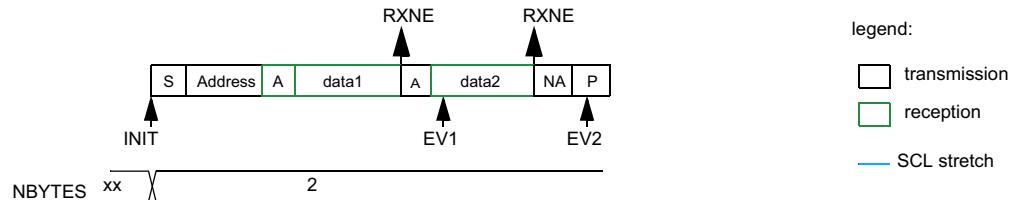
Figure 261. Transfer sequence flowchart for FMPi2C master receiver for N &gt; 255 bytes



MSv35972V1

Figure 262. Transfer bus diagrams for FMPI2C master receiver

Example FMPI2C master receiver 2 bytes, automatic end mode (STOP)

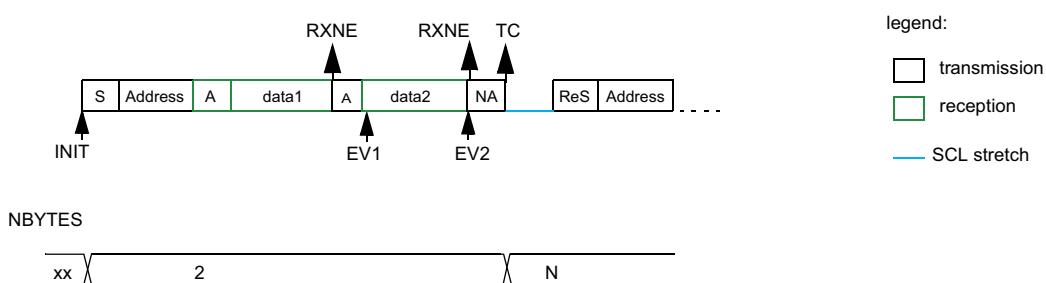


INIT: program Slave address, program NBYTES = 2, AUTOEND=1, set START

EV1: RXNE ISR: rd data1

EV2: RXNE ISR: rd data2

Example FMPI2C 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

MS35979V1

### 23.4.9 FMPI2C\_TIMINGR register configuration examples

The tables below provide examples of how to program the FMPI2C\_TIMINGR to obtain timings compliant with the I<sup>2</sup>C specification. In order to get more accurate configuration values, the STM32CubeMX tool (I<sup>2</sup>C Configuration window) should be used.

### 23.4.10 SMBus specific features

This section is relevant only when SMBus feature is supported. Refer to [Section 23.3: FMPI2C implementation](#).

#### Introduction

The System Management Bus (SMBus) is a two-wire interface through which various devices can communicate with each other and with the rest of the system. It is based on I<sup>2</sup>C principles of operation. SMBus provides a control bus for system and power management related tasks.

This peripheral is compatible with the SMBUS specification rev 2.0 (<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.

#### SMBUS is based on I<sup>2</sup>C specification rev 2.1.

#### 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 version 2.0 (<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 FMPI2C\_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 version 2.0 (<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 FMPI2C\_CR1 register. Refer to [Slave Byte Control mode on page 706](#) for more details.

### Host Notify protocol

This peripheral supports the Host Notify protocol by setting the SMBHEN bit in the FMPI2C\_CR1 register. In this case the host will acknowledge 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 will acknowledge 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 FMPI2C\_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 FMPI2C\_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 FMPI2C\_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

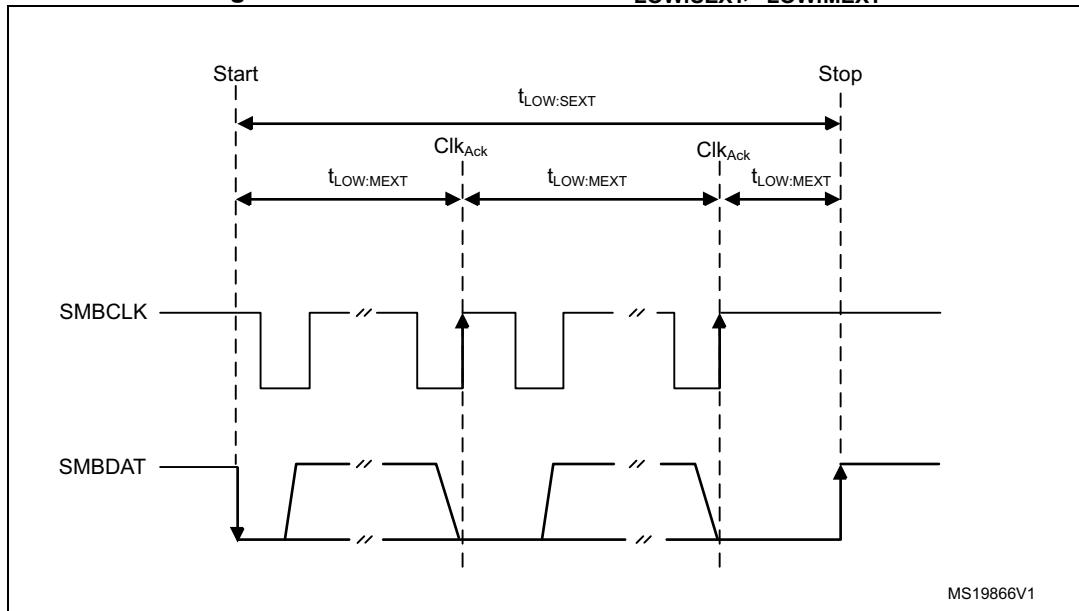
This peripheral embeds hardware timers in order to be compliant with the 3 timeouts defined in SMBus specification version 2.0.

**Table 131. 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 will also extend 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 will also extend 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 263. Timeout intervals for  $t_{LOW:SEXT}$ ,  $t_{LOW:MEXT}$**



MS19866V1

### 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 130: I2C-SMBUS specification clock timings](#))

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.

### 23.4.11 SMBus initialization

This section is relevant only when SMBus feature is supported. Refer to [Section 23.3: FMPI2C implementation](#).

In addition to FMPI2C 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 FMPI2C\_CR1 register. Refer to [Slave Byte Control mode on page 706](#) for more details.

#### Specific address (Slave mode)

The specific SMBus addresses should be enabled if needed. Refer to [Bus idle detection on page 728](#) for more details.

- The SMBus Device Default address (0b1100 001) is enabled by setting the SMBDEN bit in the FMPI2C\_CR1 register.
- The SMBus Host address (0b0001 000) is enabled by setting the SMBHEN bit in the FMPI2C\_CR1 register.
- The Alert Response Address (0b0001100) is enabled by setting the ALERTEN bit in the FMPI2C\_CR1 register.

#### Packet error checking

PEC calculation is enabled by setting the PECEN bit in the FMPI2C\_CR1 register. Then the PEC transfer is managed with the help of a hardware byte counter: NBYTES[7:0] in the FMPI2C\_CR2 register. The PECEN bit must be configured before enabling the FMPI2C.

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 PECBYTE bit is set and the RELOAD bit is cleared. If RELOAD is set, PECBYTE has no effect.

**Caution:** Changing the PECEN configuration is not allowed when the FMPI2C is enabled.

**Table 132. 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 FMPi2C\_TIMEOUTR register. The timers must be programmed in such a way that they detect a timeout before the maximum time given in the SMBus specification version 2.0.

- $t_{TIMEOUT}$  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 FMPi2C\_TIMEOUTR 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 FMPi2C\_ISR register.

Refer to [Table 133: Examples of TIMEOUTA settings for various FMPi2CCLK 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 FMPi2C\_TIMEOUTR 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 728](#) section, the TIMEOUT flag is set in the FMPi2C\_ISR register.

Refer to [Table 134: Examples of TIMEOUTB settings for various FMPi2CCLK 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 FMPi2C\_TIMEOUTR 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 FMPi2C\_ISR register.

Refer to [Table 135: Examples of TIMEOUTA settings for various FMP12CCLK frequencies \(max t<sub>IDLE</sub> = 50 µs\)](#)

**Caution:** Changing the TIMEOUTA and TIDLE configuration is not allowed when the TIMEOUTEN is set.

### 23.4.12 SMBus: FMP12C\_TIMEOUTTR register configuration examples

This section is relevant only when SMBus feature is supported. Refer to [Section 23.3: FMP12C implementation](#).

- Configuring the maximum duration of t<sub>TIMEOUT</sub> to 25 ms:

**Table 133. Examples of TIMEOUTA settings for various FMP12CCLK frequencies (max t<sub>TIMEOUT</sub> = 25 ms)**

f <sub>I2CCLK</sub>	TIMEOUTA[11:0] bits	TIDLE bit	TIMEOUTEN bit	t <sub>TIMEOUT</sub>
8 MHz	0x61	0	1	98 x 2048 x 125 ns = 25 ms
16 MHz	0xC3	0	1	196 x 2048 x 62.5 ns = 25 ms

- Configuring the maximum duration of t<sub>LOW:SEXT</sub> and t<sub>LOW:MEXT</sub> to 8 ms:

**Table 134. Examples of TIMEOUTB settings for various FMP12CCLK frequencies**

f <sub>I2CCLK</sub>	TIMEOUTB[11:0] bits	TEXTEN bit	t <sub>LOW:EXT</sub>
8 MHz	0x1F	1	32 x 2048 x 125 ns = 8 ms
16 MHz	0x3F	1	64 x 2048 x 62.5 ns = 8 ms

- Configuring the maximum duration of t<sub>IDLE</sub> to 50 µs

**Table 135. Examples of TIMEOUTA settings for various FMP12CCLK frequencies (max t<sub>IDLE</sub> = 50 µs)**

f <sub>I2CCLK</sub>	TIMEOUTA[11:0] bits	TIDLE bit	TIMEOUTEN bit	t <sub>TIDLE</sub>
8 MHz	0x63	1	1	100 x 4 x 125 ns = 50 µs
16 MHz	0xC7	1	1	200 x 4 x 62.5 ns = 50 µs

### 23.4.13 SMBus slave mode

This section is relevant only when SMBus feature is supported. Refer to [Section 23.3: FMPi2C implementation](#).

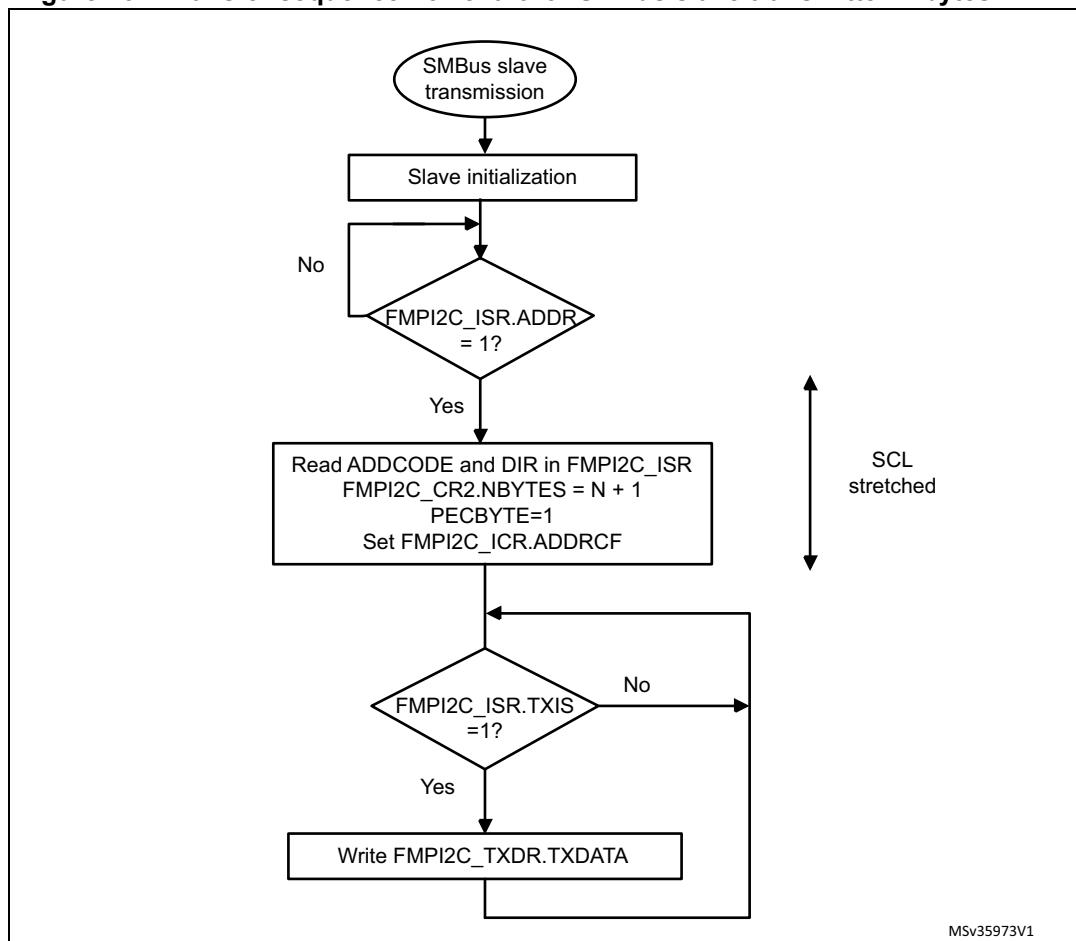
In addition to FMPi2C slave transfer management (refer to [Section 23.4.7: FMPi2C slave mode](#)) some additional software flowcharts are provided to support SMBus.

#### SMBus Slave transmitter

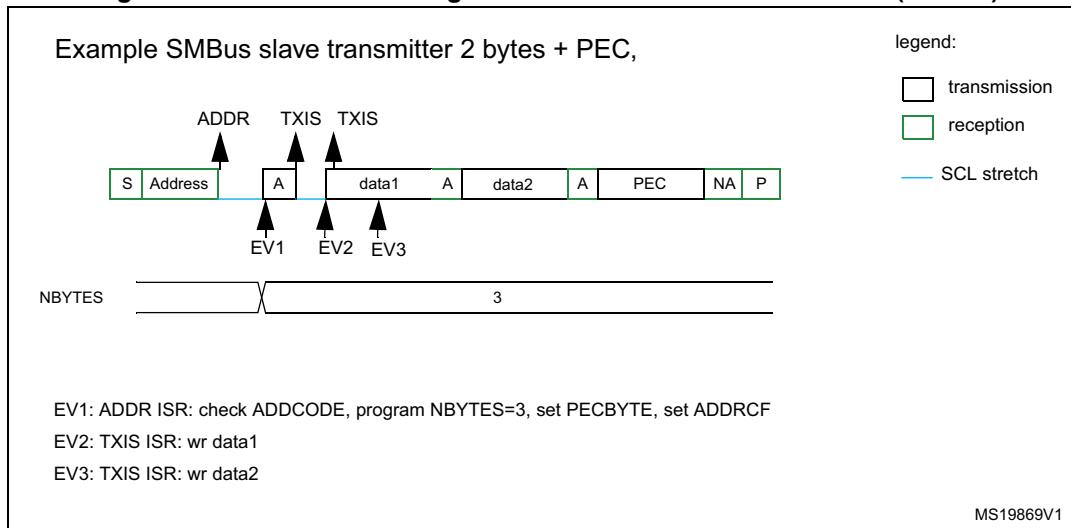
When the IP is used in SMBus, SBC must be programmed to '1' in order to allow the PEC transmission at the end of the programmed number of data bytes. When the PECBYTE bit is set, the number of bytes programmed in NBYTES[7:0] includes the PEC transmission. In that case the total number of TXIS interrupts will be NBYTES-1 and the content of the FMPi2C\_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 264. Transfer sequence flowchart for SMBus slave transmitter N bytes + PEC**



MSv35973V1

**Figure 265. Transfer bus diagrams for SMBus slave transmitter (SBC=1)**

### SMBus Slave receiver

When the FMPI2C 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 706](#) for more details.

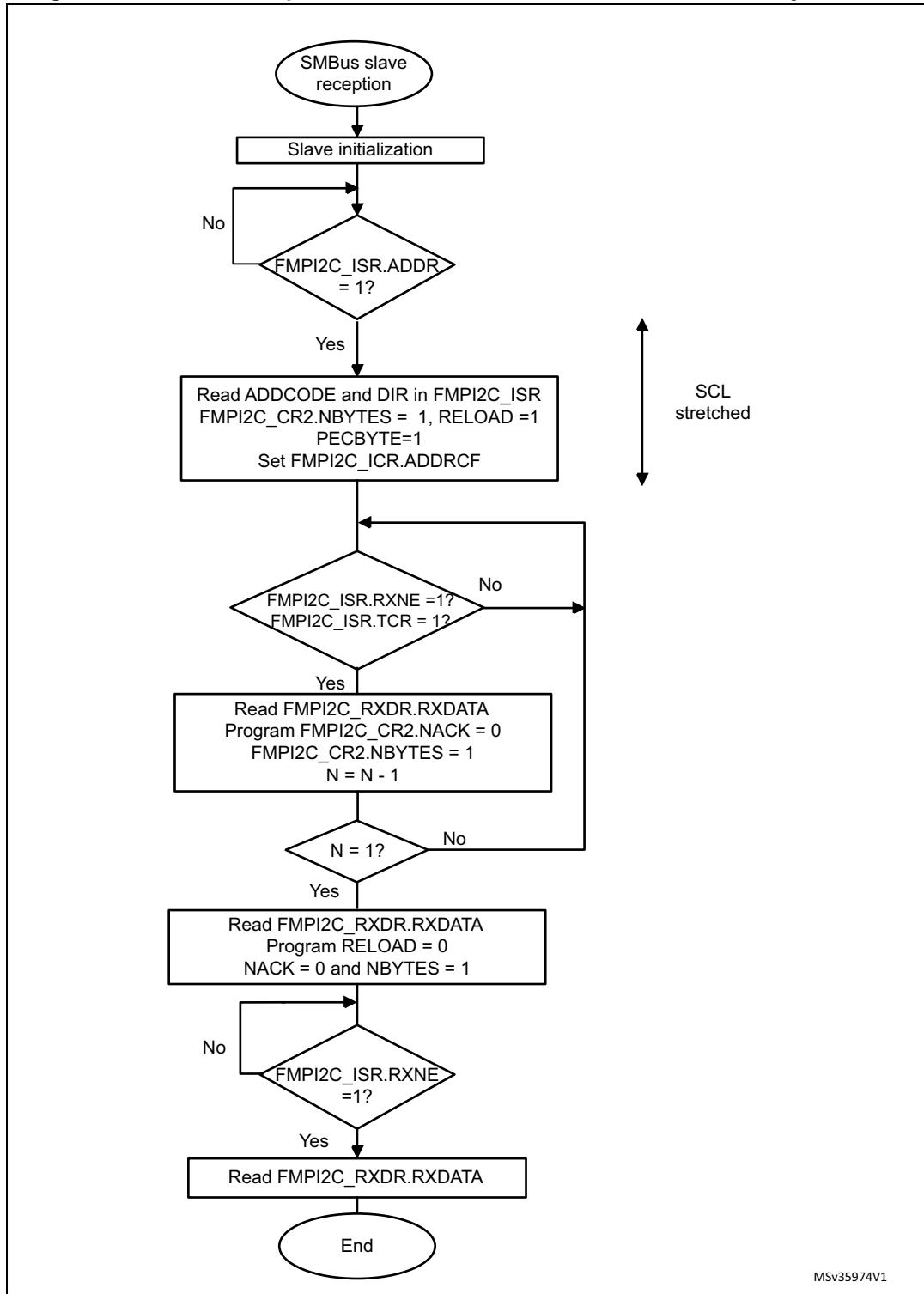
In order to check the PEC byte, the RELOAD bit must be cleared and the PECBYTE bit must be set. In this case, after NBYTES-1 data have been received, the next received byte is compared with the internal FMPI2C\_PECR register content. A NACK is automatically generated if the comparison does not match, 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 FMPI2C\_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 FMPI2C\_CR1 register.

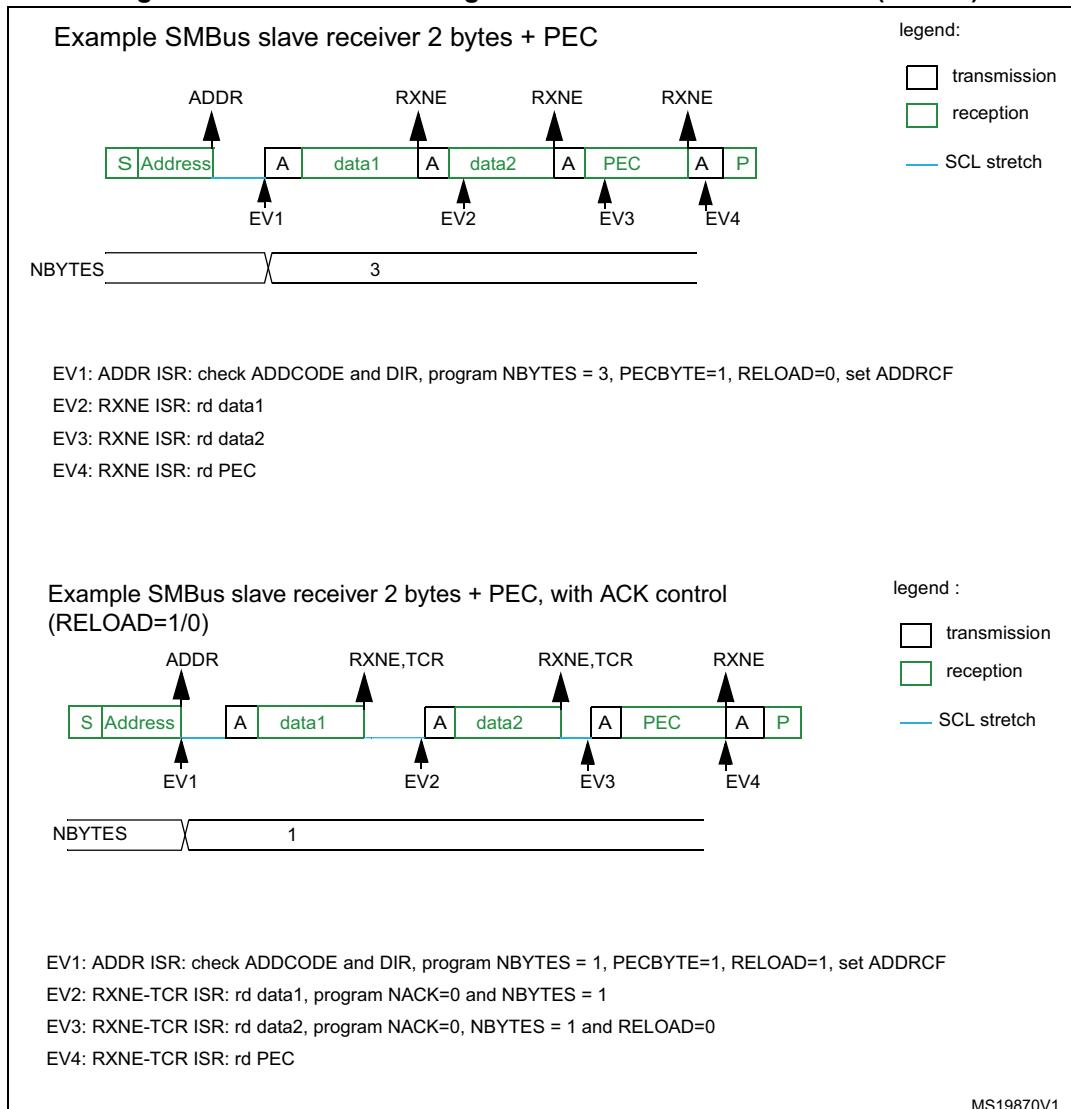
If no ACK software control is needed, the user can program PECBYTE=1 and, in the same write operation, program NBYTES with the number of bytes to be received in a continuous flow. After NBYTES-1 are received, the next received byte is checked as being the PEC.

**Caution:** The PECBYTE bit has no effect when the RELOAD bit is set.

Figure 266. Transfer sequence flowchart for SMBus slave receiver N Bytes + PEC



MSv35974V1

**Figure 267. Bus transfer diagrams for SMBus slave receiver (SBC=1)**

This section is relevant only when SMBus feature is supported. Refer to [Section 23.3: FMP12C implementation](#).

In addition to FMP12C master transfer management (refer to [Section 23.4.8: FMP12C master mode](#)) some additional software flowcharts are provided to support SMBus.

### SMBus Master transmitter

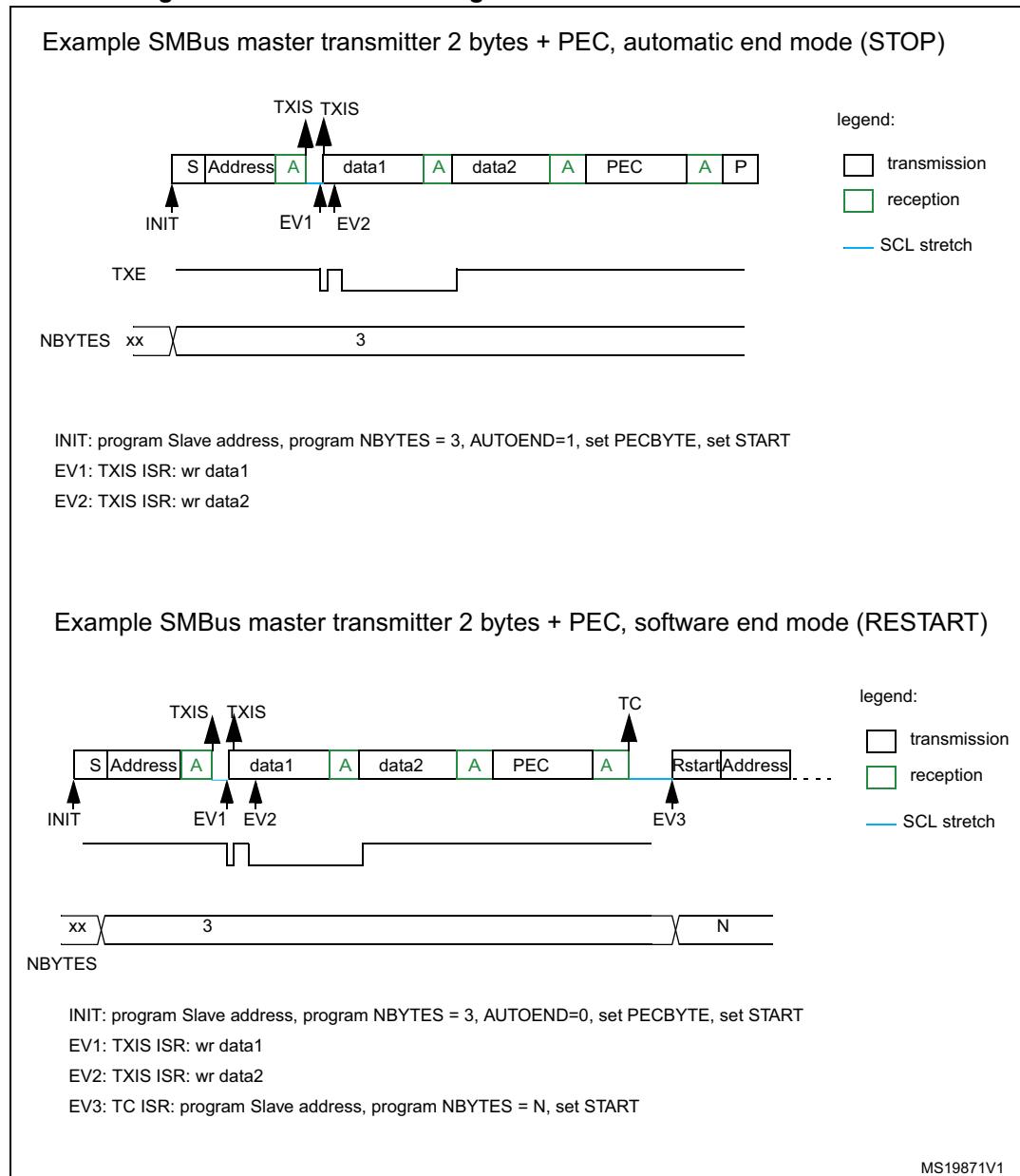
When the SMBus master wants to transmit the PEC, the PECBYTE bit must be set and the number of bytes must be programmed in the NBYTES[7:0] field, before setting the START bit. In this case the total number of TXIS interrupts will be NBYTES-1. So if the PECBYTE bit is set when NBYTES=0x1, the content of the FMP12C\_PECR register is automatically transmitted.

If the SMBus master wants to send a STOP condition after the PEC, automatic end mode should 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 FMPI2C\_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 268. Bus transfer diagrams for SMBus master transmitter**



MS19871V1

### SMBus Master receiver

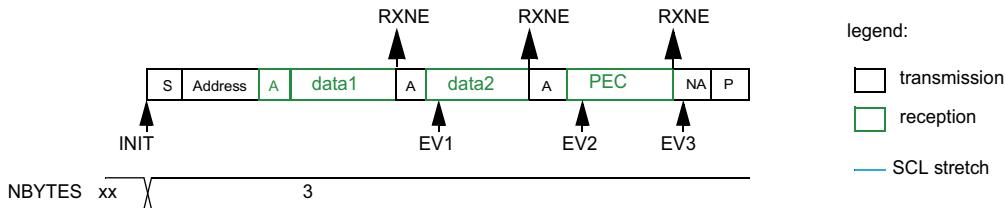
When the SMBus master wants to receive the PEC followed by a STOP at the end of the transfer, automatic end mode can be selected (AUTOEND=1). The PECBYTE bit must be set and the slave address must be programmed, before setting the START bit. In this case, after NBYTES-1 data have been received, the next received byte is automatically checked versus the FMP12C\_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 FMP12C\_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 269. Bus transfer diagrams for SMBus master receiver**

Example SMBus master receiver 2 bytes + PEC, automatic end mode (STOP)



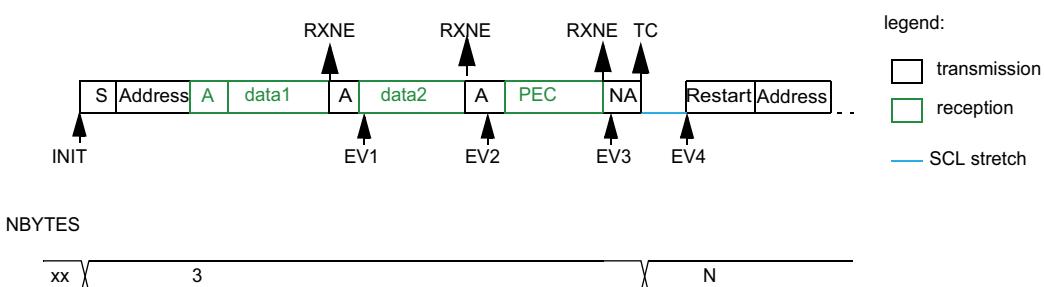
INIT: program Slave address, program NBYTES = 3, AUTOEND=1, set PECPBYTE, set START

EV1: RXNE ISR: rd data1

EV2: RXNE ISR: rd data2

EV3: RXNE ISR: rd PEC

Example SMBus master receiver 2 bytes + PEC, software end mode (RESTART)



INIT: program Slave address, program NBYTES = 3, AUTOEND=0, set PECPBYTE, set START

EV1: RXNE ISR: rd data1

EV2: RXNE ISR: rd data2

EV3: RXNE ISR: read PEC

EV4: TC ISR: program Slave address, program NBYTES = N, set START

MS19872V1

### 23.4.14 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 FMPI2C 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 FMP12C enters address recognition state like for a correct START condition.

When a bus error is detected, the BERR flag is set in the FMP12C\_ISR register, and an interrupt is generated if the ERRIE bit is set in the FMP12C\_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 FMP12C\_ISR register, and an interrupt is generated if the ERRIE bit is set in the FMP12C\_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 FMP12C\_TXDR register is sent if TXE=0, 0xFF if not.
  - When a new byte should be sent and the FMP12C\_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 FMP12C\_ISR register, and an interrupt is generated if the ERRIE bit is set in the FMP12C\_CR1 register.

### Packet Error Checking Error (PECERR)

This section is relevant only when the SMBus feature is supported. Refer to [Section 23.3: FMP12C implementation](#).

A PEC error is detected when the received PEC byte does not match with the FMP12C\_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 FMP12C\_ISR register, and an interrupt is generated if the ERRIE bit is set in the FMP12C\_CR1 register.

### Timeout Error (TIMEOUT)

This section is relevant only when the SMBus feature is supported. Refer to [Section 23.3: FMPi2C 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 FMPi2C\_ISR register, and an interrupt is generated if the ERRIE bit is set in the FMPi2C\_CR1 register.

### Alert (ALERT)

This section is relevant only when the SMBus feature is supported. Refer to [Section 23.3: FMPi2C implementation](#).

The ALERT flag is set when the FMPi2C 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 FMPi2C\_CR1 register.

## 23.4.15 DMA requests

### Transmission using DMA

DMA (Direct Memory Access) can be enabled for transmission by setting the TXDMAEN bit in the FMPi2C\_CR1 register. Data is loaded from an SRAM area configured using the DMA peripheral (see [Section 9: Direct memory access controller \(DMA\) on page 202](#)) to the FMPi2C\_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 717](#).

- 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 731](#) and [SMBus Master transmitter on page 734](#).

*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 FMPI2C\_CR1 register. Data is loaded from the FMPI2C\_RXDR register to an SRAM area configured using the DMA peripheral (refer to [Section 9: 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 23.3: FMPI2C implementation](#)): the PEC transfer is managed with the NBYTES counter. Refer to [SMBus Slave receiver on page 732](#) and [SMBus Master receiver on page 736](#).

*Note:* If DMA is used for reception, the RXIE bit does not need to be enabled.

#### 23.4.16 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.

### 23.5 FMPI2C low-power modes

Table 136. low-power modes

Mode	Description
Sleep	No effect FMPI2C interrupts cause the device to exit the Sleep mode.
Stop	The contents of FMPI2C registers are kept.
Standby	The FMPI2C peripheral is powered down and must be reinitialized after exiting Standby.

## 23.6 FMPI2C interrupts

The table below gives the list of FMPI2C interrupt requests.

**Table 137. FMPI2C Interrupt requests**

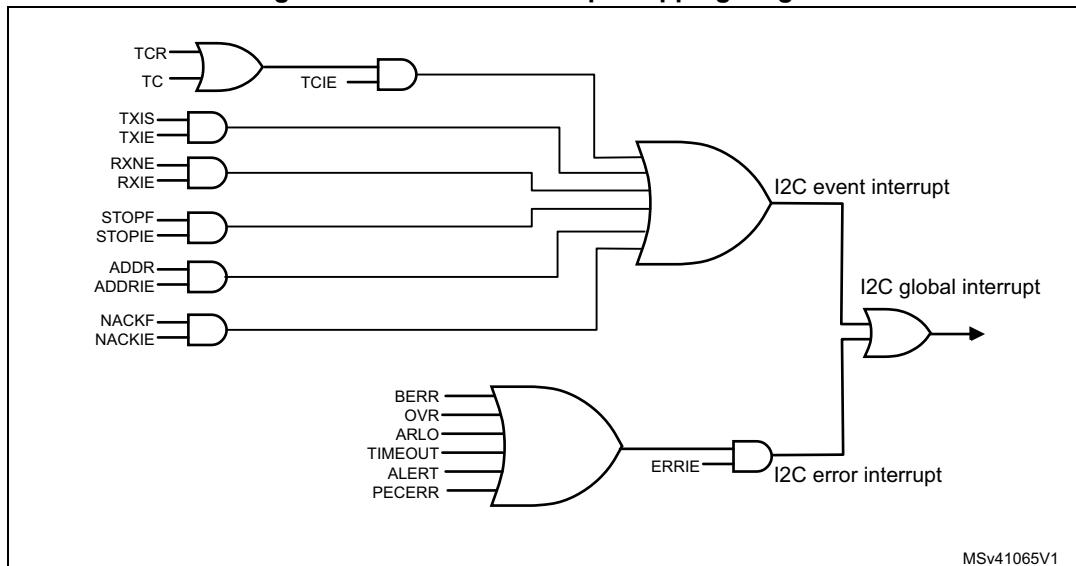
Interrupt event	Event flag	Event flag/Interrupt clearing method	Interrupt enable control bit
Receive buffer not empty	RXNE	Read FMPI2C_RXDR register	RXIE
Transmit buffer interrupt status	TXIS	Write FMPI2C_TXDR register	TXIE
Stop detection interrupt flag	STOPF	Write STOPCF=1	STOPIE
Transfer Complete Reload	TCR	Write FMPI2C_CR2 with NBYTES[7:0] ≠ 0	TCIE
Transfer complete	TC	Write START=1 or STOP=1	
Address matched	ADDR	Write ADDRCF=1	ADDRIE
NACK reception	NACKF	Write NACKCF=1	NACKIE
Bus error	BERR	Write BERRCF=1	ERRIE
Arbitration loss	ARLO	Write ARLOCF=1	
Overrun/Underrun	OVR	Write OVRCF=1	
PEC error	PECERR	Write PECERRCF=1	
Timeout/t <sub>L</sub> OW error	TIMEOUT	Write TIMEOUTCF=1	
SMBus Alert	ALERT	Write ALERTCF=1	

Depending on the product implementation, all these interrupts events can either share the same interrupt vector (FMPI2C global interrupt), or be grouped into 2 interrupt vectors (FMPI2C event interrupt and FMPI2C error interrupt). Refer to for details.

In order to enable the FMPI2C interrupts, the following sequence is required:

1. Configure and enable the FMPI2C IRQ channel in the NVIC.
2. Configure the FMPI2C to generate interrupts.

Figure 270. FMPI2C interrupt mapping diagram



## 23.7 FMPI2C registers

Refer to [Section 1.1 on page 51](#) for a list of abbreviations used in register descriptions.

The peripheral registers are accessed by words (32-bit).

### 23.7.1 Control register 1 (FMPI2C\_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  $2 \times \text{PCLK1} + 6 \times \text{FMPI2CCLK}$ .

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				ERRIE	TCIE	STOP IE	NACK IE	ADDR IE	RXIE	TXIE	PE
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 23.3: FMPI2C 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 23.3: FMPI2C 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 23.3: FMPI2C 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 23.3: FMPI2C 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 FMP12C 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 will filter 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 FMP12C 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 will 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 FMP12C 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.*

### 23.7.2 Control register 2 (FMP12C\_CR2)

Address offset: 0x04

Reset value: 0x0000 0000

Access: No wait states, except if a write access occurs while a write access to this register is ongoing. In this case, wait states are inserted in the second write access until the previous one is completed. The latency of the second write access can be up to  $2 \times \text{PCLK1} + 6 \times \text{FMP12CCLK}$ .

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	PEC BYTE	AUTO END	RE LOAD	NBYTES[7:0]							
					rs	rw	rw	rw							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NACK	STOP	START	HEAD 10R	ADD10	RD_	WRN	SADD[9:0]						rw		
rs	rs	rs	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 23.3: FMP12C 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 will follow).
- 1: The transfer is not completed after the NBYTES data transfer (NBYTES will be 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 FMPI2C\_ICR register.

0: No Start generation.

1: Restart/Start generation:

- If the FMPI2C 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 will generate 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 FMPI2C is in slave mode.*

*This bit has no effect when RELOAD is set. In 10-bit addressing mode, if a NACK is received on the first part of the address, the START bit is not cleared by hardware and the master will resend the address sequence, unless the START bit is cleared by software*

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.*

### 23.7.3 Own address 1 register (FMP12C\_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 FMP12CCLK.

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.	Res.	OA1 MODE	OA1[9:8]	OA1[7:1]						OA1[0]	
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.*

### 23.7.4 Own address 2 register (FMP12C\_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  $2 \times \text{PCLK1} + 6 \times \text{FMP12CCLK}$ .

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]						Res.		
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 FMP12C 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.

### 23.7.5 Timing register (FMP12C\_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			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SCLH[7:0]				SCLL[7:0]				rw				rw			

Bits 31:28 **PRESC[3:0]**: Timing prescaler

This field is used to prescale FMP12CCLK in order to generate the clock period  $t_{PRESC}$  used for data setup and hold counters (refer to [FMP12C timings on page 698](#)) and for SCL high and low level counters (refer to [FMP12C master initialization on page 713](#)).

$$t_{PRESC} = (PRESC+1) \times t_{I2CCLK}$$

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  $t_{SCLDEL}$ .

$$t_{SCLDEL} = (SCLDEL+1) \times t_{PRESC}$$

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  $t_{SDADEL}$ .

$$t_{SDADEL} = SDADEL \times t_{PRESC}$$

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.

$$t_{SCLH} = (SCLH+1) \times t_{PRESC}$$

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.

$$t_{SCLL} = (SCLL+1) \times t_{PRESC}$$

Note:  $t_{SCLL}$  is also used to generate  $t_{BUF}$  and  $t_{SU:STA}$  timings.

Note: This register must be configured when the FMP12C is disabled ( $PE = 0$ ).

Note: The STM32CubeMX tool calculates and provides the I2C\_TIMINGR content in the I2C Configuration window.

### 23.7.6 Timeout register (FMPI2C\_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  $2 \times \text{PCLK1} + 6 \times \text{FMPI2CCLK}$ .

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

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 FMPI2C interface, a timeout error is detected (TIMEOUT=1).

Bits 30:28 Reserved, must be kept at reset value.

Bits 27:16 **TIMEOUTB[11:0]**: Bus timeout B

This field is used to configure the cumulative clock extension timeout:

In master mode, the master cumulative clock low extend time ( $t_{\text{LOW:MEXT}}$ ) is detected

In slave mode, the slave cumulative clock low extend time ( $t_{\text{LOW:SEXT}}$ ) is detected

$$t_{\text{LOW:EXT}} = (\text{TIMEOUTB}+1) \times 2048 \times t_{\text{I2CCLK}}$$

*Note: These bits can be written only when TEXTEN=0.*

Bit 15 **TIMOUTEN**: Clock timeout enable

0: SCL timeout detection is disabled

1: SCL timeout detection is enabled: when SCL is low for more than  $t_{\text{TIMEOUT}}$  (TIDLE=0) or high for more than  $t_{\text{IDLE}}$  (TIDLE=1), a timeout error is detected (TIMEOUT=1).

Bits 14:13 Reserved, must be kept at reset value.

Bit 12 **TIDLE**: Idle clock timeout detection

0: TIMEOUTA is used to detect SCL low timeout

1: TIMEOUTA is used to detect both SCL and SDA high timeout (bus idle condition)

*Note: This bit can be written only when TIMOUTEN=0.*

Bits 11:0 **TIMEOUTA[11:0]**: Bus Timeout A

This field is used to configure:

- The SCL low timeout condition  $t_{\text{TIMEOUT}}$  when TIDLE=0

$$t_{\text{TIMEOUT}} = (\text{TIMEOUTA}+1) \times 2048 \times t_{\text{I2CCLK}}$$

- The bus idle condition (both SCL and SDA high) when TIDLE=1

$$t_{\text{IDLE}} = (\text{TIMEOUTA}+1) \times 4 \times t_{\text{I2CCLK}}$$

*Note: These bits can be written only when TIMOUTEN=0.*

**Note:** If the SMBus feature is not supported, this register is reserved and forced by hardware to “0x00000000”. Refer to [Section 23.3: FMPI2C implementation](#).

### 23.7.7 Interrupt and status register (FMP12C\_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
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 23.3: FMP12C implementation.*

Bit 12 TIMEOUT: Timeout or t<sub>LOW</sub> detection flag

This flag is set by hardware when a timeout or extended clock timeout occurred. It is cleared by software by setting the TIMEOUTCF bit.

*Note: This bit is cleared by hardware when PE=0.*

*If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'.*

*Refer to Section 23.3: FMP12C 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 23.3: FMP12C 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 *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 *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 FMP12C\_RXDR register, and is ready to be read. It is cleared when FMP12C\_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 FMP12C\_TXDR register is empty and the data to be transmitted must be written in the FMP12C\_TXDR register. It is cleared when the next data to be sent is written in the FMP12C\_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 FMP12C\_TXDR register is empty. It is cleared when the next data to be sent is written in the FMP12C\_TXDR register.

This bit can be written to '1' by software in order to flush the transmit data register FMP12C\_TXDR.

*Note: This bit is set by hardware when PE=0.*

### 23.7.8 Interrupt clear register (FMP12C\_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 FMP12C\_ISR register.

*Note: If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'. Refer to [Section 23.3: FMP12C implementation](#).*

Bit 12 **TIMOUTCF**: Timeout detection flag clear

Writing 1 to this bit clears the TIMEOUT flag in the FMP12C\_ISR register.

*Note: If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'. Refer to [Section 23.3: FMP12C implementation](#).*

Bit 11 **PECCF**: PEC Error flag clear

Writing 1 to this bit clears the PECERR flag in the FMP12C\_ISR register.

*Note: If the SMBus feature is not supported, this bit is reserved and forced by hardware to '0'. Refer to [Section 23.3: FMP12C implementation](#).*

- Bit 10 **OVRCF**: Overrun/Underrun flag clear  
Writing 1 to this bit clears the OVR flag in the FMP12C\_ISR register.
- Bit 9 **ARLOCF**: Arbitration Lost flag clear  
Writing 1 to this bit clears the ARLO flag in the FMP12C\_ISR register.
- Bit 8 **BERRCF**: Bus error flag clear  
Writing 1 to this bit clears the BERRF flag in the FMP12C\_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 FMP12C\_ISR register.
- Bit 4 **NACKCF**: Not Acknowledge flag clear  
Writing 1 to this bit clears the ACKF flag in FMP12C\_ISR register.
- Bit 3 **ADDRCF**: Address matched flag clear  
Writing 1 to this bit clears the ADDR flag in the FMP12C\_ISR register. Writing 1 to this bit also clears the START bit in the FMP12C\_CR2 register.
- Bits 2:0 Reserved, must be kept at reset value.

### 23.7.9 PEC register (FMP12C\_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.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	PEC[7:0]														
															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 PESEN=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 23.3: FMP12C implementation](#).

### 23.7.10 Receive data register (FMPI2C\_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.	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.
RXDATA[7:0]												r			

Bits 31:8 Reserved, must be kept at reset value.

Bits 7:0 **RXDATA[7:0]** 8-bit receive data

Data byte received from the I<sup>2</sup>C bus.

### 23.7.11 Transmit data register (FMPI2C\_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.	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.
TXDATA[7:0]												rw			

Bits 31:8 Reserved, must be kept at reset value.

Bits 7:0 **TXDATA[7:0]** 8-bit transmit data

Data byte to be transmitted to the I<sup>2</sup>C bus.

*Note:* These bits can be written only when TXE=1.

### 23.7.12 FMPI2C register map

The table below provides the FMPI2C register map and reset values.

**Table 138. FMPI2C 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
0x0	<b>FMPI2C_CR1</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PECEN	ALERTEN	SMBDEN	SMBHEN	GCEN	NOSTRETCH	SBC	RXDMAEN	TXDMAEN	START	Rec.	ANOFF	ADD10	RD_WRN	ERRIE	TCIE	STOPIE	NACKIE	ADDRIE	RXIE	TXIE	PE	Res.	
		Reset value	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		
0x4	<b>FMPI2C_CR2</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	NBYTES[7:0]								SADD[9:0]								OA1[9:0]							
		Reset value	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	PECBYTE	0	AUTOEND	0	RELOAD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x8	<b>FMPI2C_OAR1</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SCLDEL[3:0]								SDADEL[3:0]								OA2[7:1]							
		Reset value	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			
0xC	<b>FMPI2C_OAR2</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TIMEOUTB[11:0]								SCLH[7:0]								SCLL[7:0]							
		Reset value	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			
0x10	<b>FMPI2C_TIMING_R</b>	PRESC[3:0]	0	0	0	0	0	0	0	SCLDEL[3:0]								SDADEL[3:0]								OA2[7:1]							
		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			
0x14	<b>FMPI2C_TIMEOUTR</b>	TEXTEN	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TIMEOUTB[11:0]								SCLH[7:0]								TIMEOUTA[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			
0x18	<b>FMPI2C_ISR</b>	ADDCODE[6:0]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ADDCODE[6:0]								DIR								TIMEOUTA[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			
0x1C	<b>FMPI2C_ICR</b>	ALERTCF	0	ALERTCF	0	TIMEOUT	0	TITLE	0	TIMEOUTB[11:0]								OA1MODE								OA2MSK[2:0]							
		Reset value	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			
0x20	<b>FMPI2C_PECR</b>	PEC[7:0]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PEC[7:0]								BERRCF								RXDATA[7:0]							
		Reset value	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			
0x24	<b>FMPI2C_RXDR</b>	0	0	0	0	0	0	0	0	PEC[7:0]								BERRCF								RXDATA[7:0]							
		Reset value	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			

**Table 138. FMPI2C register map and reset values (continued)**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x28	FMPI2C_TXDR	Res.	TXDATA[7:0]																														
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			

Refer to [Section 2.2.2 on page 56](#) for the register boundary addresses.

## 24 Inter-integrated circuit ( $I^2C$ ) interface

### 24.1 $I^2C$ introduction

$I^2C$  (inter-integrated circuit) bus Interface serves as an interface between the microcontroller and the serial  $I^2C$  bus. It provides multimaster capability, and controls all  $I^2C$  bus-specific sequencing, protocol, arbitration and timing. It supports the standard mode (Sm, up to 100 kHz) and Fm mode (Fm, up to 400 kHz).

It may be used for a variety of purposes, including CRC generation and verification, SMBus (system management bus) and PMBus (power management bus).

Depending on specific device implementation DMA capability can be available for reduced CPU overload.

## 24.2 I<sup>2</sup>C main features

- Parallel-bus/I<sup>2</sup>C protocol converter
- Multimaster capability: the same interface can act as Master or Slave
- I<sup>2</sup>C Master features:
  - Clock generation
  - Start and Stop generation
- I<sup>2</sup>C Slave features:
  - Programmable I<sup>2</sup>C Address detection
  - Dual Addressing Capability to acknowledge 2 slave addresses
  - Stop bit detection
- Generation and detection of 7-bit/10-bit addressing and General Call
- Supports different communication speeds:
  - Standard Speed (up to 100 kHz)
  - Fast Speed (up to 400 kHz)
- Analog noise filter
- Programmable digital noise filter
- Status flags:
  - Transmitter/Receiver mode flag
  - End-of-Byte transmission flag
  - I<sup>2</sup>C busy flag
- Error flags:
  - Arbitration lost condition for master mode
  - Acknowledgment failure after address/ data transmission
  - Detection of misplaced start or stop condition
  - Overrun/Underrun if clock stretching is disabled
- 2 Interrupt vectors:
  - 1 Interrupt for successful address/ data communication
  - 1 Interrupt for error condition
- Optional clock stretching
- 1-byte buffer with DMA capability
- Configurable PEC (packet error checking) generation or verification:
  - PEC value can be transmitted as last byte in Tx mode
  - PEC error checking for last received byte
- SMBus 2.0 Compatibility:
  - 25 ms clock low timeout delay
  - 10 ms master cumulative clock low extend time
  - 25 ms slave cumulative clock low extend time
  - Hardware PEC generation/verification with ACK control
  - Address Resolution Protocol (ARP) supported
- PMBus Compatibility

**Note:** Some of the above features may not be available in certain products. The user should refer to the product data sheet, to identify the specific features supported by the  $I^2C$  interface implementation.

## 24.3 $I^2C$ 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  $I^2C$  bus by a data pin (SDA) and by a clock pin (SCL). It can be connected with a standard (up to 100 kHz) or fast (up to 400 kHz)  $I^2C$  bus.

### 24.3.1 Mode selection

The interface can operate in one of the four following modes:

- Slave transmitter
- Slave receiver
- Master transmitter
- Master receiver

By default, it operates in slave mode. The interface automatically switches from slave to master, after it generates a START condition and from master to slave, if an arbitration loss or a Stop generation occurs, allowing multimaster capability.

#### Communication flow

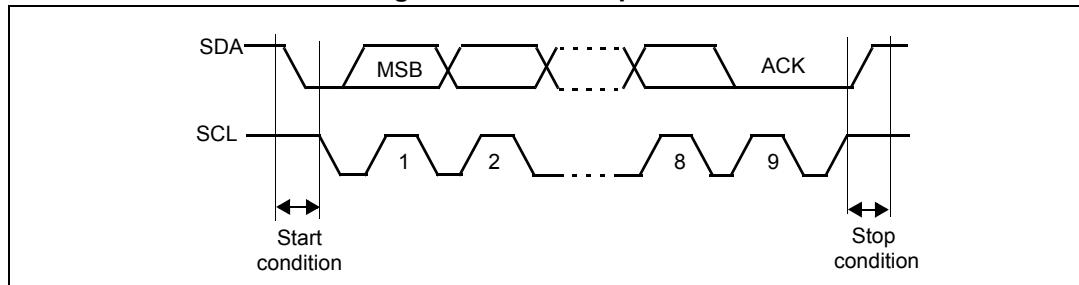
In Master mode, the  $I^2C$  interface initiates a data transfer and generates the clock signal. A serial data transfer always begins with a start condition and ends with a stop condition. Both start and stop conditions are generated in master mode by software.

In Slave mode, the interface is capable of recognizing its own addresses (7 or 10-bit), and the General Call address. The General Call address detection may be enabled or disabled by software.

Data and addresses are transferred as 8-bit bytes, MSB first. The first byte(s) following the start condition contain the address (one in 7-bit mode, two in 10-bit mode). The address is always transmitted in Master mode.

A 9th clock pulse follows the 8 clock cycles of a byte transfer, during which the receiver must send an acknowledge bit to the transmitter. Refer to [Figure 271](#).

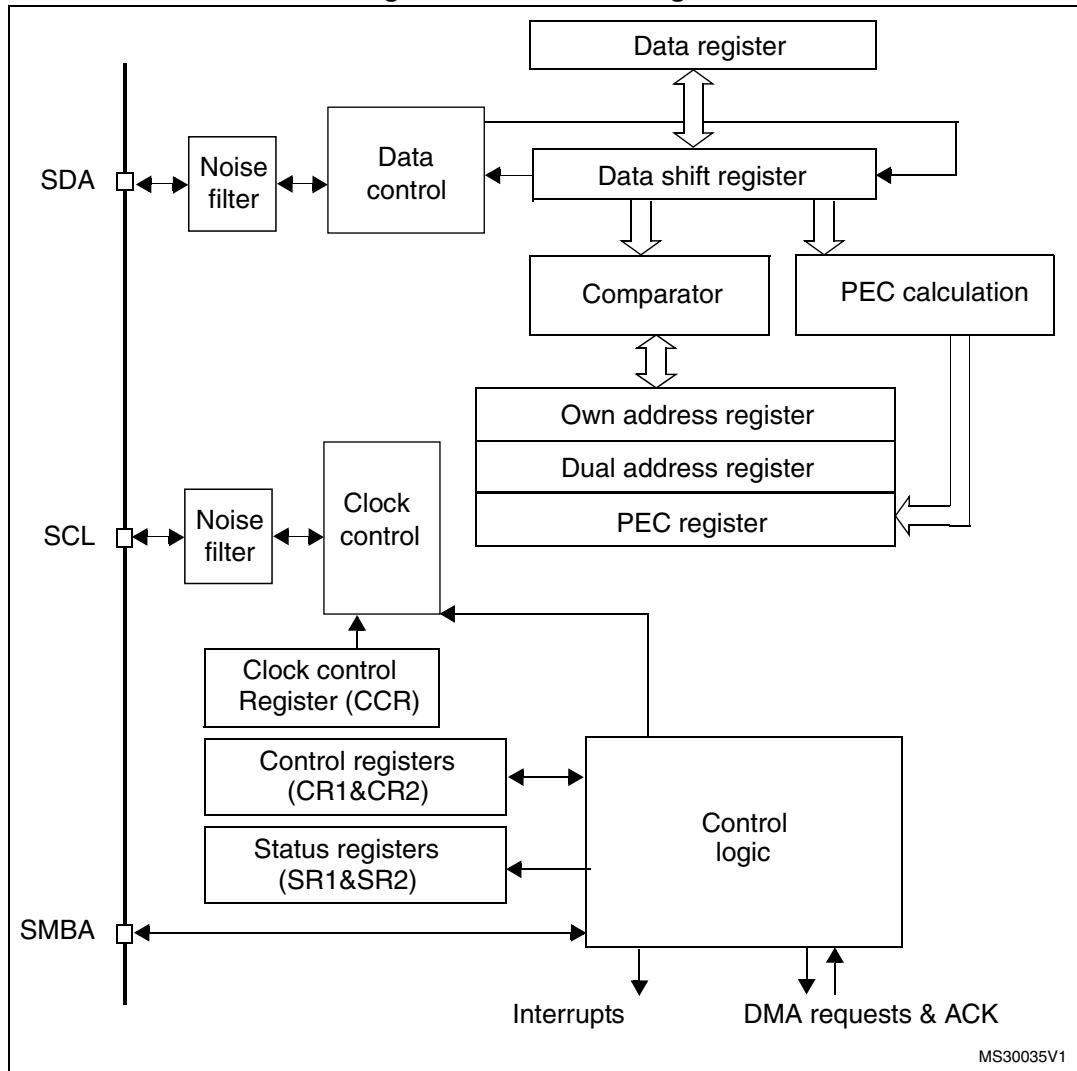
**Figure 271.  $I^2C$  bus protocol**



Acknowledge may be enabled or disabled by software. The  $I^2C$  interface addresses (dual addressing 7-bit/ 10-bit and/or general call address) can be selected by software.

The block diagram of the I<sup>2</sup>C interface is shown in [Figure 272](#).

**Figure 272. I<sup>2</sup>C block diagram**



1. SMBA is an optional signal in SMBus mode. This signal is not applicable if SMBus is disabled.

### 24.3.2 I<sup>2</sup>C slave mode

By default the I<sup>2</sup>C interface operates in Slave mode. To switch from default Slave mode to Master mode a Start condition generation is needed.

The peripheral input clock must be programmed in the I2C\_CR2 register in order to generate correct timings. The peripheral input clock frequency must be at least:

- 2 MHz in Sm mode
- 4 MHz in Fm mode

As soon as a start condition is detected, the address is received from the SDA line and sent to the shift register. Then it is compared with the address of the interface (OAR1) and with OAR2 (if ENDUAL=1) or the General Call address (if ENGC = 1).

**Note:** In 10-bit addressing mode, the comparison includes the header sequence (11110xx0), where xx denotes the two most significant bits of the address.

**Header or address not matched:** the interface ignores it and waits for another Start condition.

**Header matched** (10-bit mode only): the interface generates an acknowledge pulse if the ACK bit is set and waits for the 8-bit slave address.

**Address matched:** the interface generates in sequence:

- An acknowledge pulse if the ACK bit is set
- The ADDR bit is set by hardware and an interrupt is generated if the ITEVFEN bit is set.
- If ENDUAL=1, the software has to read the DUALF bit to check which slave address has been acknowledged.

In 10-bit mode, after receiving the address sequence the slave is always in Receiver mode. It will enter Transmitter mode on receiving a repeated Start condition followed by the header sequence with matching address bits and the least significant bit set (11110xx1).

The TRA bit indicates whether the slave is in Receiver or Transmitter mode.

### Slave transmitter

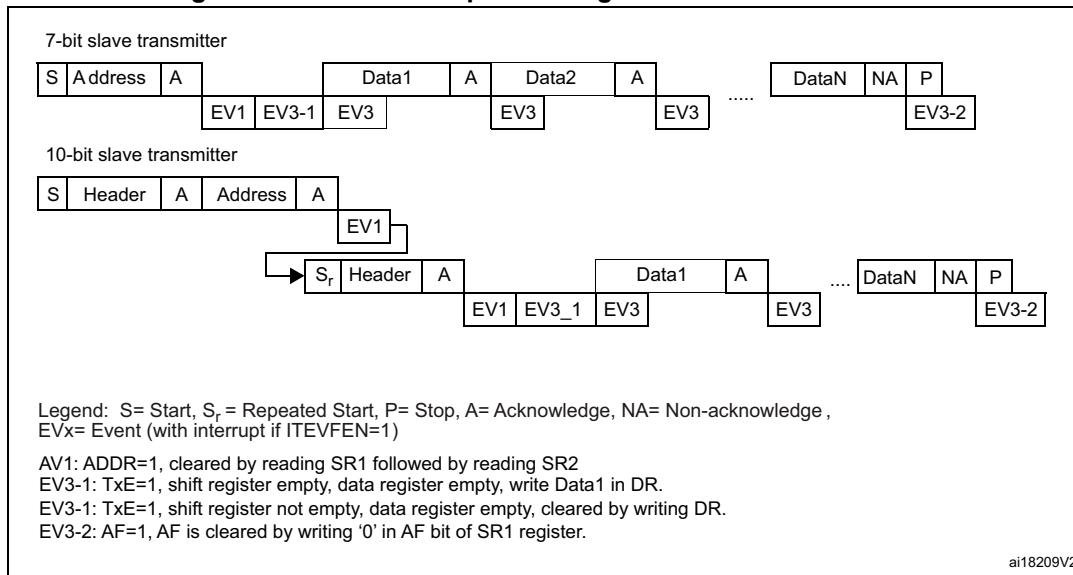
Following the address reception and after clearing ADDR, the slave sends bytes from the DR register to the SDA line via the internal shift register.

The slave stretches SCL low until ADDR is cleared and DR filled with the data to be sent (see [Figure 273 Transfer sequencing EV1 EV3](#)).

When the acknowledge pulse is received:

- The TxE bit is set by hardware with an interrupt if the ITEVFEN and the ITBUFEN bits are set.

If TxE is set and some data were not written in the I2C\_DR register before the end of the next data transmission, the BTF bit is set and the interface waits until BTF is cleared by a read to I2C\_SR1 followed by a write to the I2C\_DR register, stretching SCL low.

**Figure 273. Transfer sequence diagram for slave transmitter**

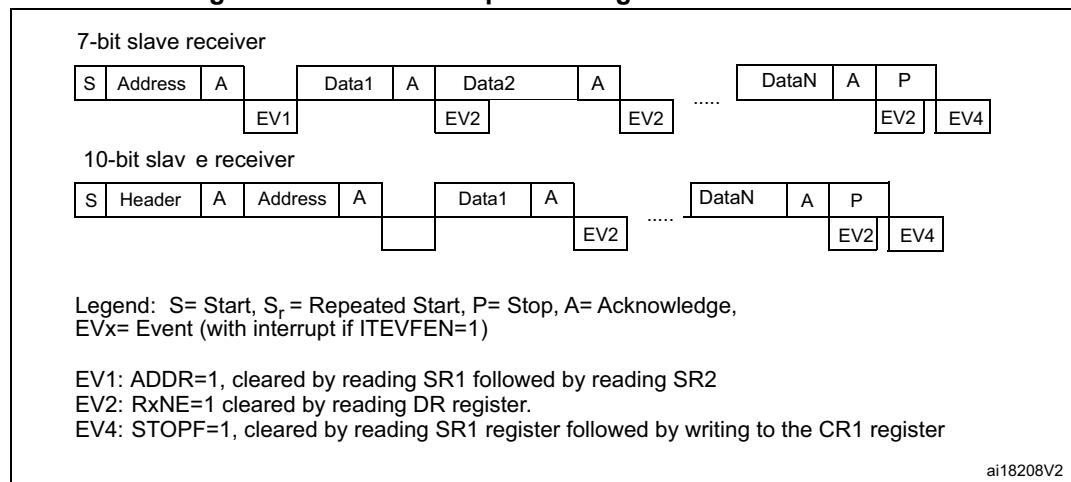
1. The EV1 and EV3\_1 events stretch SCL low until the end of the corresponding software sequence.
2. The EV3 event stretches SCL low if the software sequence is not completed before the end of the next byte transmission

### Slave receiver

Following the address reception and after clearing ADDR, the slave receives bytes from the SDA line into the DR register via the internal shift register. After each byte the interface generates in sequence:

- An acknowledge pulse if the ACK bit is set
- The RxNE bit is set by hardware and an interrupt is generated if the ITEVFEN and ITBUFEN bit is set.

If RxNE is set and the data in the DR register is not read before the end of the next data reception, the BTF bit is set and the interface waits until BTF is cleared by a read from the I2C\_DR register, stretching SCL low (see [Figure 274 Transfer sequencing](#)).

**Figure 274. Transfer sequence diagram for slave receiver**

1. The EV1 event stretches SCL low until the end of the corresponding software sequence.
2. The EV2 event stretches SCL low if the software sequence is not completed before the end of the next byte reception.
3. After checking the SR1 register content, the user should perform the complete clearing sequence for each flag found set.  
Thus, for ADDR and STOPF flags, the following sequence is required inside the I2C interrupt routine:  
READ SR1  
if (ADDR == 1) {READ SR1; READ SR2}  
if (STOPF == 1) {READ SR1; WRITE CR1}  
The purpose is to make sure that both ADDR and STOPF flags are cleared if both are found set.

### Closing slave communication

After the last data byte is transferred a Stop Condition is generated by the master. The interface detects this condition and sets:

- The STOPF bit and generates an interrupt if the ITEVFEN bit is set.

The STOPF bit is cleared by a read of the SR1 register followed by a write to the CR1 register (see [Figure 274: Transfer sequence diagram for slave receiver](#) EV4).

### 24.3.3 I<sup>2</sup>C master mode

In Master mode, the I<sup>2</sup>C interface initiates a data transfer and generates the clock signal. A serial data transfer always begins with a Start condition and ends with a Stop condition. Master mode is selected as soon as the Start condition is generated on the bus with a START bit.

The following is the required sequence in master mode.

- Program the peripheral input clock in I2C\_CR2 Register in order to generate correct timings
- Configure the clock control registers
- Configure the rise time register
- Program the I2C\_CR1 register to enable the peripheral
- Set the START bit in the I2C\_CR1 register to generate a Start condition

The peripheral input clock frequency must be at least:

- 2 MHz in Sm mode
- 4 MHz in Fm mode

## SCL master clock generation

The CCR bits are used to generate the high and low level of the SCL clock, starting from the generation of the rising and falling edge (respectively). As a slave may stretch the SCL line, the peripheral checks the SCL input from the bus at the end of the time programmed in TRISE bits after rising edge generation.

- If the SCL line is low, it means that a slave is stretching the bus, and the high level counter stops until the SCL line is detected high. This allows to guarantee the minimum HIGH period of the SCL clock parameter.
- If the SCL line is high, the high level counter keeps on counting.

Indeed, the feedback loop from the SCL rising edge generation by the peripheral to the SCL rising edge detection by the peripheral takes time even if no slave stretches the clock. This loopback duration is linked to the SCL rising time (impacting SCL VIH input detection), plus delay due to the noise filter present on the SCL input path, plus delay due to internal SCL input synchronization with APB clock. The maximum time used by the feedback loop is programmed in the TRISE bits, so that the SCL frequency remains stable whatever the SCL rising time.

## Start condition

Setting the START bit causes the interface to generate a Start condition and to switch to Master mode (MSL bit set) when the BUSY bit is cleared.

*Note:* In master mode, setting the START bit causes the interface to generate a ReStart condition at the end of the current byte transfer.

Once the Start condition is sent:

- The SB bit is set by hardware and an interrupt is generated if the ITEVFEN bit is set.

Then the master waits for a read of the SR1 register followed by a write in the DR register with the Slave address (see [Figure 275](#) and [Figure 276](#) Transfer sequencing EV5).

## Slave address transmission

Then the slave address is sent to the SDA line via the internal shift register.

- In 10-bit addressing mode, sending the header sequence causes the following event:
  - The ADD10 bit is set by hardware and an interrupt is generated if the ITEVFEN bit is set.

Then the master waits for a read of the SR1 register followed by a write in the DR register with the second address byte (see [Figure 275](#) and [Figure 276](#) Transfer sequencing).

- The ADDR bit is set by hardware and an interrupt is generated if the ITEVFEN bit is set.

Then the master waits for a read of the SR1 register followed by a read of the SR2 register (see [Figure 275](#) and [Figure 276](#) Transfer sequencing).

- In 7-bit addressing mode, one address byte is sent.

As soon as the address byte is sent,

- The ADDR bit is set by hardware and an interrupt is generated if the ITEVFEN bit is set.

Then the master waits for a read of the SR1 register followed by a read of the SR2 register (see [Figure 275](#) and [Figure 276](#) Transfer sequencing).

The master can decide to enter Transmitter or Receiver mode depending on the LSB of the slave address sent.

- In 7-bit addressing mode,
  - To enter Transmitter mode, a master sends the slave address with LSB reset.
  - To enter Receiver mode, a master sends the slave address with LSB set.
- In 10-bit addressing mode,
  - To enter Transmitter mode, a master sends the header (11110xx0) and then the slave address, (where xx denotes the two most significant bits of the address).
  - To enter Receiver mode, a master sends the header (11110xx0) and then the slave address. Then it should send a repeated Start condition followed by the header (11110xx1), (where xx denotes the two most significant bits of the address).

The TRA bit indicates whether the master is in Receiver or Transmitter mode.

### Master transmitter

Following the address transmission and after clearing ADDR, the master sends bytes from the DR register to the SDA line via the internal shift register.

The master waits until the first data byte is written into I2C\_DR (see [Figure 275 Transfer sequencing EV8\\_1](#)).

When the acknowledge pulse is received, the TxE bit is set by hardware and an interrupt is generated if the ITEVFEN and ITBUFEN bits are set.

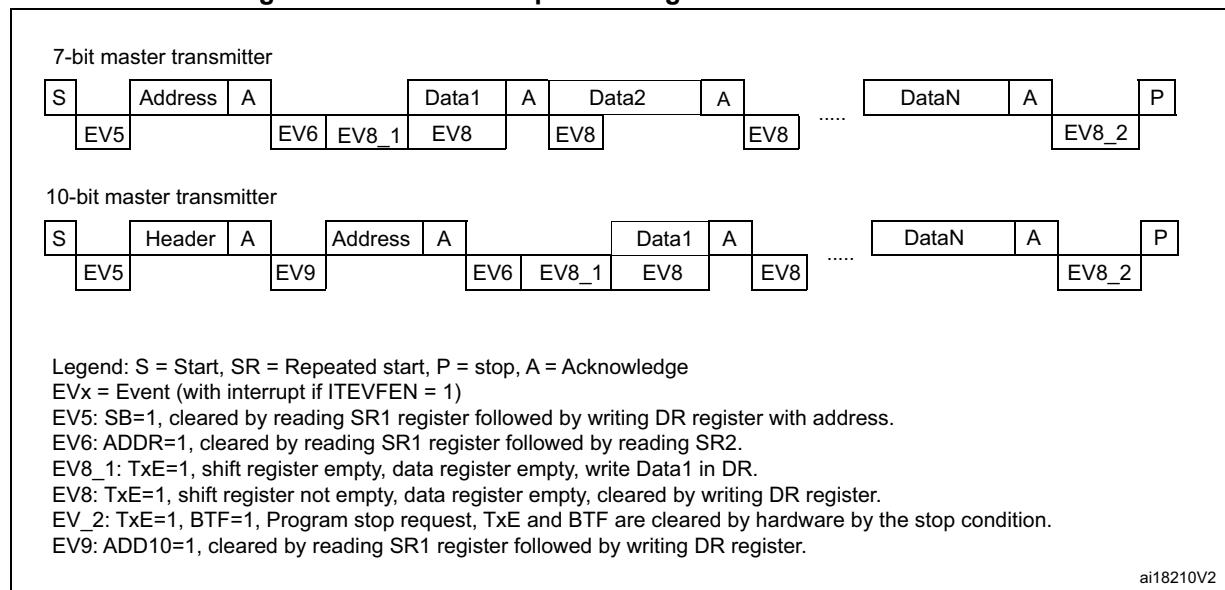
If TxE is set and a data byte was not written in the DR register before the end of the last data transmission, BTF is set and the interface waits until BTF is cleared by a write to I2C\_DR, stretching SCL low.

### Closing the communication

After the last byte is written to the DR register, the STOP bit is set by software to generate a Stop condition (see [Figure 275 Transfer sequencing EV8\\_2](#)). The interface automatically goes back to slave mode (MSL bit cleared).

*Note:* *Stop condition should be programmed during EV8\_2 event, when either TxE or BTF is set.*

Figure 275. Transfer sequence diagram for master transmitter



1. The EV5, EV6, EV9, EV8\_1 and EV8\_2 events stretch SCL low until the end of the corresponding software sequence.
2. The EV8 event stretches SCL low if the software sequence is not complete before the end of the next byte transmission.

### Master receiver

Following the address transmission and after clearing ADDR, the I<sup>2</sup>C interface enters Master Receiver mode. In this mode the interface receives bytes from the SDA line into the DR register via the internal shift register. After each byte the interface generates in sequence:

1. An acknowledge pulse if the ACK bit is set
2. The RxNE bit is set and an interrupt is generated if the ITEVFEN and ITBUFEN bits are set (see [Figure 276 Transfer sequencing EV7](#)).

If the RxNE bit is set and the data in the DR register is not read before the end of the last data reception, the BTF bit is set by hardware and the interface waits until BTF is cleared by a read in the DR register, stretching SCL low.

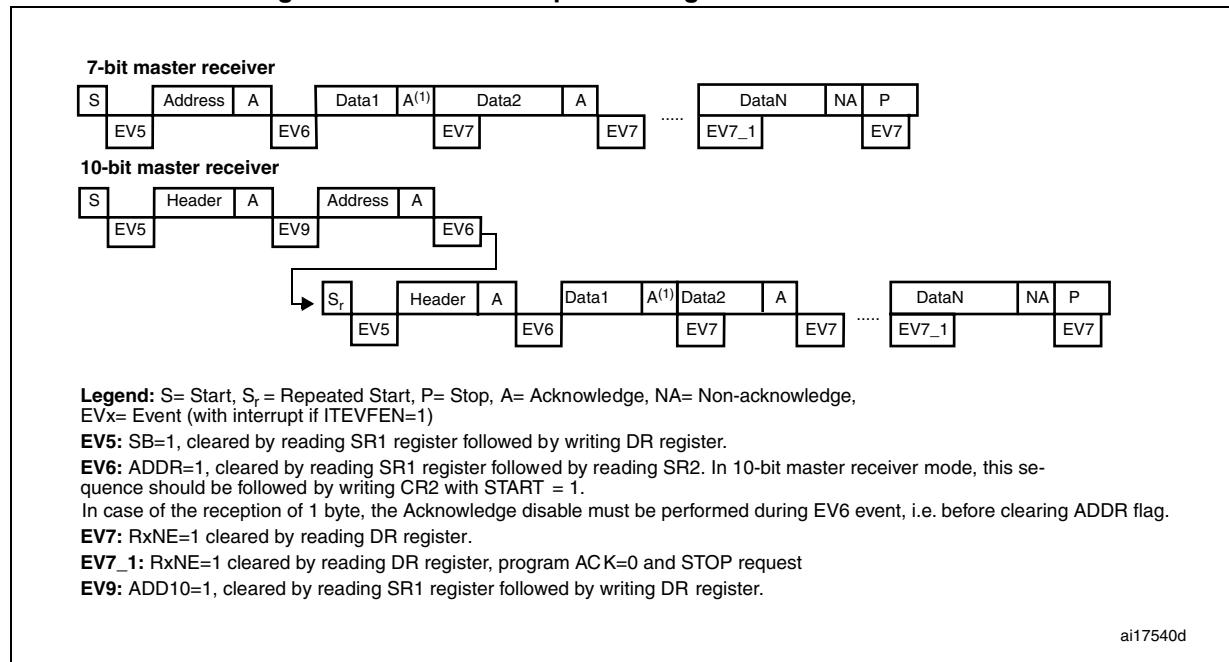
### Closing the communication

The master sends a NACK for the last byte received from the slave. After receiving this NACK, the slave releases the control of the SCL and SDA lines. Then the master can send a Stop/Restart condition.

1. To generate the nonacknowledge pulse after the last received data byte, the ACK bit must be cleared just after reading the second last data byte (after second last RxNE event).
2. In order to generate the Stop/Restart condition, software must set the STOP/START bit after reading the second last data byte (after the second last RxNE event).
3. In case a single byte has to be received, the Acknowledge disable is made during EV6 (before ADDR flag is cleared) and the STOP condition generation is made after EV6.

After the Stop condition generation, the interface goes automatically back to slave mode (MSL bit cleared).

Figure 276. Transfer sequence diagram for master receiver



ai17540d

1. If a single byte is received, it is NA.
2. The EV5, EV6 and EV9 events stretch SCL low until the end of the corresponding software sequence.
3. The EV7 event stretches SCL low if the software sequence is not completed before the end of the next byte reception.
4. The EV7\_1 software sequence must be completed before the ACK pulse of the current byte transfer.

The procedures described below are recommended if the EV7-1 software sequence is not completed before the ACK pulse of the current byte transfer.

These procedures must be followed to make sure:

- The ACK bit is set low on time before the end of the last data reception
- The STOP bit is set high after the last data reception without reception of supplementary data.

#### For 2-byte reception:

- Wait until ADDR = 1 (SCL stretched low until the ADDR flag is cleared)
- Set ACK low, set POS high
- Clear ADDR flag
- Wait until BTF = 1 (Data 1 in DR, Data2 in shift register, SCL stretched low until a data 1 is read)
- Set STOP high
- Read data 1 and 2

**For N > 2 -byte reception, from N-2 data reception**

- Wait until BTF = 1 (data N-2 in DR, data N-1 in shift register, SCL stretched low until data N-2 is read)
- Set ACK low
- Read data N-2
- Wait until BTF = 1 (data N-1 in DR, data N in shift register, SCL stretched low until a data N-1 is read)
- Set STOP high
- Read data N-1 and N

**24.3.4 Error conditions**

The following are the error conditions which may cause communication to fail.

**Bus error (BERR)**

This error occurs when the  $I^2C$  interface detects an external Stop or Start condition during an address or a data transfer. In this case:

- the BERR bit is set and an interrupt is generated if the ITERREN bit is set
- in Slave mode: data are discarded and the lines are released by hardware:
  - in case of a misplaced Start, the slave considers it is a restart and waits for an address, or a Stop condition
  - in case of a misplaced Stop, the slave behaves like for a Stop condition and the lines are released by hardware
- In Master mode: the lines are not released and the state of the current transmission is not affected. It is up to the software to abort or not the current transmission

**Acknowledge failure (AF)**

This error occurs when the interface detects a nonacknowledge bit. In this case:

- the AF bit is set and an interrupt is generated if the ITERREN bit is set
- a transmitter which receives a NACK must reset the communication:
  - If Slave: lines are released by hardware
  - If Master: a Stop or repeated Start condition must be generated by software

**Arbitration lost (ARLO)**

This error occurs when the  $I^2C$  interface detects an arbitration lost condition. In this case,

- the ARLO bit is set by hardware (and an interrupt is generated if the ITERREN bit is set)
- the  $I^2C$  Interface goes automatically back to slave mode (the MSL bit is cleared). When the  $I^2C$  loses the arbitration, it is not able to acknowledge its slave address in the same transfer, but it can acknowledge it after a repeated Start from the winning master.
- lines are released by hardware

### Overrun/underrun error (OVR)

An overrun error can occur in slave mode when clock stretching is disabled and the I<sup>2</sup>C interface is receiving data. The interface has received a byte (RxNE=1) and the data in DR has not been read, before the next byte is received by the interface. In this case,

- The last received byte is lost.
- In case of Overrun error, software should clear the RxNE bit and the transmitter should re-transmit the last received byte.

Underrun error can occur in slave mode when clock stretching is disabled and the I<sup>2</sup>C interface is transmitting data. The interface has not updated the DR with the next byte (TxEN=1), before the clock comes for the next byte. In this case,

- The same byte in the DR register will be sent again
- The user should make sure that data received on the receiver side during an underrun error are discarded and that the next bytes are written within the clock low time specified in the I<sup>2</sup>C bus standard.

For the first byte to be transmitted, the DR must be written after ADDR is cleared and before the first SCL rising edge. If not possible, the receiver must discard the first data.

### 24.3.5 Programmable noise filter

In Fm mode, the I<sup>2</sup>C standard requires that spikes are suppressed to a length of 50 ns on SDA and SCL lines.

An analog noise filter is implemented in the SDA and SCL I/Os. This filter is enabled by default and can be disabled by setting the ANOFF bit in the I2C\_FLTR register.

A digital noise filter can be enabled by configuring the DNF[3:0] bits to a non-zero value. This suppresses the spikes on SDA and SCL inputs with a length of up to  $DNF[3:0] * T_{PCLK1}$ .

Enabling the digital noise filter increases the SDA hold time by  $(DNF[3:0] + 1) * T_{PCLK}$ .

To be compliant with the maximum hold time of the I<sup>2</sup>C-bus specification version 2.1 (Thd:dat), the DNF bits must be programmed using the constraints shown in [Table 139](#), and assuming that the analog filter is disabled.

**Note:** *DNF[3:0] must only be configured when the I<sup>2</sup>C is disabled (PE = 0). If the analog filter is also enabled, the digital filter is added to the analog filter.*

**Table 139. Maximum DNF[3:0] value to be compliant with Thd:dat(max)**

PCLK1 frequency	Maximum DNF value	
	Sm mode	Fm mode
$2 \leq F_{PCLK1} \leq 5$	2	0
$5 < F_{PCLK1} \leq 10$	12	0
$10 < F_{PCLK1} \leq 20$	15	1
$20 < F_{PCLK1} \leq 30$	15	7
$30 < F_{PCLK1} \leq 40$	15	13
$40 < F_{PCLK1} \leq 50$	15	15

**Note:** For each frequency range, the constraint is given based on the worst case which is the minimum frequency of the range. Greater DNF values can be used if the system can support maximum hold time violation.

### 24.3.6 SDA/SCL line control

- If clock stretching is enabled:
  - Transmitter mode: If  $TxE=1$  and  $BTF=1$ : the interface holds the clock line low before transmission to wait for the microcontroller to write the byte in the Data Register (both buffer and shift register are empty).
  - Receiver mode: If  $RxNE=1$  and  $BTF=1$ : the interface holds the clock line low after reception to wait for the microcontroller to read the byte in the Data Register (both buffer and shift register are full).
- If clock stretching is disabled in Slave mode:
  - Overrun Error in case of  $RxNE=1$  and no read of DR has been done before the next byte is received. The last received byte is lost.
  - Underrun Error in case  $TxE=1$  and no write into DR has been done before the next byte must be transmitted. The same byte will be sent again.
  - Write Collision not managed.

### 24.3.7 SMBus

#### Introduction

The System Management Bus (SMBus) is a two-wire interface through which various devices can communicate with each other and with the rest of the system. It is based on  $I^2C$  principles of operation. SMBus provides a control bus for system and power management related tasks. A system may use SMBus to pass messages to and from devices instead of toggling individual control lines.

The System Management Bus Specification refers to three types of devices. A *slave* is a device that is receiving or responding to a command. A *master* is a device that issues commands, generates the clocks, and terminates the transfer. A *host* is a specialized master that provides the main interface to the system's CPU. A host must be a master-slave and must support the SMBus host notify protocol. Only one host is allowed in a system.

#### Similarities between SMBus and $I^2C$

- 2 wire bus protocol (1 Clk, 1 Data) + SMBus Alert line optional
- Master-slave communication, Master provides clock
- Multi master capability
- SMBus data format similar to  $I^2C$  7-bit addressing format ([Figure 271](#)).

#### Differences between SMBus and $I^2C$

The following table describes the differences between SMBus and  $I^2C$ .

**Table 140. SMBus vs. I<sup>2</sup>C**

SMBus	I <sup>2</sup> C
Max. speed 100 kHz	Max. speed 400 kHz
Min. clock speed 10 kHz	No minimum clock speed
35 ms clock low timeout	No timeout
Logic levels are fixed	Logic levels are V <sub>DD</sub> dependent
Different address types (reserved, dynamic etc.)	7-bit, 10-bit and general call slave address types
Different bus protocols (quick command, process call etc.)	No bus protocols

### SMBus application usage

With System Management Bus, a device can provide manufacturer information, tell the system what its model/part number is, save its state for a suspend event, report different types of errors, accept control parameters, and return its status. SMBus provides a control bus for system and power management related tasks.

### Device identification

Any device that exists on the System Management Bus as a slave has a unique address called the Slave Address. For the list of reserved slave addresses, refer to the SMBus specification version. 2.0 (<http://smbus.org/>).

### Bus protocols

The SMBus specification supports up to 9 bus protocols. For more details of these protocols and SMBus address types, refer to SMBus specification version. 2.0. These protocols should be implemented by the user software.

### Address resolution protocol (ARP)

SMBus slave address conflicts can be resolved by dynamically assigning a new unique address to each slave device. The Address Resolution Protocol (ARP) has the following attributes:

- Address assignment uses the standard SMBus physical layer arbitration mechanism
- Assigned addresses remain constant while device power is applied; address retention through device power loss is also allowed
- No additional SMBus packet overhead is incurred after address assignment. (i.e. subsequent accesses to assigned slave addresses have the same overhead as accesses to fixed address devices.)
- Any SMBus master can enumerate the bus

### Unique device identifier (UDID)

In order to provide a mechanism to isolate each device for the purpose of address assignment, each device must implement a unique device identifier (UDID).

For the details on 128 bit UDID and more information on ARP, refer to SMBus specification version 2.0.

### SMBus alert mode

SMBus Alert is an optional signal with an interrupt line for devices that want to trade their ability to master for a pin. SMBA is a wired-AND signal just as the SCL and SDA signals are. SMBA is used in conjunction with the SMBus General Call Address. Messages invoked with the SMBus are 2 bytes long.

A slave-only device can signal the host through SMBA that it wants to talk by setting ALERT bit in I2C\_CR1 register. The host processes the interrupt and simultaneously accesses all SMBA devices through the *Alert Response Address* (known as ARA having a value 0001 100X). Only the device(s) which pulled SMBA low will acknowledge the Alert Response Address. This status is identified using SMBALERT Status flag in I2C\_SR1 register. The host performs a modified Receive Byte operation. The 7 bit device address provided by the slave transmit device is placed in the 7 most significant bits of the byte. The eighth bit can be a zero or one.

If more than one device pulls SMBA low, the highest priority (lowest address) device will win communication rights via standard arbitration during the slave address transfer. After acknowledging the slave address the device must disengage its SMBA pull-down. If the host still sees SMBA low when the message transfer is complete, it knows to read the ARA again.

A host which does not implement the SMBA signal may periodically access the ARA.

For more details on SMBus Alert mode, refer to SMBus specification version 2.0 (<http://smbus.org/>).

### Timeout error

There are differences in the timing specifications between  $I^2C$  and SMBus. SMBus defines a clock low timeout, TIMEOUT of 35 ms. Also SMBus specifies TLOW: SEXT as the cumulative clock low extend time for a slave device. SMBus specifies TLOW: MEXT as the cumulative clock low extend time for a master device. For more details on these timeouts, refer to SMBus specification version 2.0.

The status flag Timeout or Tlow Error in I2C\_SR1 shows the status of this feature.

### How to use the interface in SMBus mode

To switch from  $I^2C$  mode to SMBus mode, the following sequence should be performed.

- Set the SMBus bit in the I2C\_CR1 register
- Configure the SMBTYPE and ENARP bits in the I2C\_CR1 register as required for the application

If you want to configure the device as a master, follow the Start condition generation procedure in [Section 24.3.3: I<sup>2</sup>C master mode](#). Otherwise, follow the sequence in [Section 24.3.2: I<sup>2</sup>C slave mode](#).

The application has to control the various SMBus protocols by software.

- SMB Device Default Address acknowledged if ENARP=1 and SMBTYPE=0
- SMB Host Header acknowledged if ENARP=1 and SMBTYPE=1
- SMB Alert Response Address acknowledged if SMBALERT=1

### 24.3.8 DMA requests

DMA requests (when enabled) are generated only for data transfer. DMA requests are generated by Data Register becoming empty in transmission and Data Register becoming full in reception. The DMA must be initialized and enabled before the I<sup>2</sup>C data transfer. The DMAEN bit must be set in the I2C\_CR2 register before the ADDR event. In master mode or in slave mode when clock stretching is enabled, the DMAEN bit can also be set during the ADDR event, before clearing the ADDR flag. The DMA request must be served before the end of the current byte transfer. When the number of data transfers which has been programmed for the corresponding DMA stream is reached, the DMA controller sends an End of Transfer EOT signal to the I<sup>2</sup>C interface and generates a Transfer Complete interrupt if enabled:

- Master transmitter: In the interrupt routine after the EOT interrupt, disable DMA requests then wait for a BTF event before programming the Stop condition.
- Master receiver
  - When the number of bytes to be received is equal to or greater than two, the DMA controller sends a hardware signal, EOT\_1, corresponding to the last but one data byte (number\_of\_bytes – 1). If, in the I2C\_CR2 register, the LAST bit is set, I<sup>2</sup>C automatically sends a NACK after the next byte following EOT\_1. The user can generate a Stop condition in the DMA Transfer Complete interrupt routine if enabled.
  - When a single byte must be received: the NACK must be programmed during EV6 event, i.e. program ACK=0 when ADDR=1, before clearing ADDR flag. Then the user can program the STOP condition either after clearing ADDR flag, or in the DMA Transfer Complete interrupt routine.

### Transmission using DMA

DMA mode can be enabled for transmission by setting the DMAEN bit in the I2C\_CR2 register. Data will be loaded from a Memory area configured using the DMA peripheral (refer to the DMA specification) to the I2C\_DR register whenever the TxE bit is set. To map a DMA stream x for I<sup>2</sup>C transmission (where x is the stream number), perform the following sequence:

1. Set the I2C\_DR register address in the DMA\_SxPAR register. The data will be moved to this address from the memory after each TxE event.
2. Set the memory address in the DMA\_SxMA0R register (and in DMA\_SxMA1R register in the case of a double buffer mode). The data will be loaded into I2C\_DR from this memory after each TxE event.
3. Configure the total number of bytes to be transferred in the DMA\_SxNDTR register. After each TxE event, this value will be decremented.
4. Configure the DMA stream priority using the PL[0:1] bits in the DMA\_SxCR register
5. Set the DIR bit in the DMA\_SxCR register and configure interrupts after half transfer or full transfer depending on application requirements.
6. Activate the stream by setting the EN bit in the DMA\_SxCR register.

When the number of data transfers which has been programmed in the DMA Controller registers is reached, the DMA controller sends an End of Transfer EOT/ EOT\_1 signal to the I<sup>2</sup>C interface and the DMA generates an interrupt, if enabled, on the DMA stream interrupt vector.

*Note:* Do not enable the ITBUFEN bit in the I2C\_CR2 register if DMA is used for transmission.

### Reception using DMA

DMA mode can be enabled for reception by setting the DMAEN bit in the I2C\_CR2 register. Data will be loaded from the I2C\_DR register to a Memory area configured using the DMA peripheral (refer to the DMA specification) whenever a data byte is received. To map a DMA stream x for I<sup>2</sup>C reception (where x is the stream number), perform the following sequence:

1. Set the I2C\_DR register address in DMA\_SxPAR register. The data will be moved from this address to the memory after each RxNE event.
2. Set the memory address in the DMA\_SxMA0R register (and in DMA\_SxMA1R register in the case of a double buffer mode). The data will be loaded from the I2C\_DR register to this memory area after each RxNE event.
3. Configure the total number of bytes to be transferred in the DMA\_SxNDTR register. After each RxNE event, this value will be decremented.
4. Configure the stream priority using the PL[0:1] bits in the DMA\_SxCR register
5. Reset the DIR bit and configure interrupts in the DMA\_SxCR register after half transfer or full transfer depending on application requirements.
6. Activate the stream by setting the EN bit in the DMA\_SxCR register.

When the number of data transfers which has been programmed in the DMA Controller registers is reached, the DMA controller sends an End of Transfer EOT/ EOT\_1 signal to the I<sup>2</sup>C interface and DMA generates an interrupt, if enabled, on the DMA stream interrupt vector.

*Note:* *Do not enable the ITBUFEN bit in the I2C\_CR2 register if DMA is used for reception.*

### 24.3.9 Packet error checking

A PEC calculator has been implemented to improve the reliability of communication. The PEC is calculated by using the  $C(x) = x^8 + x^2 + x + 1$  CRC-8 polynomial serially on each bit.

- PEC calculation is enabled by setting the ENPEC bit in the I2C\_CR1 register. PEC is a CRC-8 calculated on all message bytes including addresses and R/W bits.
  - In transmission: set the PEC transfer bit in the I2C\_CR1 register after the TxE event corresponding to the last byte. The PEC will be transferred after the last transmitted byte.
  - In reception: set the PEC bit in the I2C\_CR1 register after the RxNE event corresponding to the last byte so that the receiver sends a NACK if the next received byte is not equal to the internally calculated PEC. In case of Master-Receiver, a NACK must follow the PEC whatever the check result. The PEC must

be set before the ACK of the CRC reception in slave mode. It must be set when the ACK is set low in master mode.

- A PECERR error flag/interrupt is also available in the I2C\_SR1 register.
- If DMA and PEC calculation are both enabled:-

  - In transmission: when the I<sup>2</sup>C interface receives an EOT signal from the DMA controller, it automatically sends a PEC after the last byte.
  - In reception: when the I<sup>2</sup>C interface receives an EOT\_1 signal from the DMA controller, it will automatically consider the next byte as a PEC and will check it. A DMA request is generated after PEC reception.

- To allow intermediate PEC transfers, a control bit is available in the I2C\_CR2 register (LAST bit) to determine if it is really the last DMA transfer or not. If it is the last DMA request for a master receiver, a NACK is automatically sent after the last received byte.
- PEC calculation is corrupted by an arbitration loss.

## 24.4 I<sup>2</sup>C interrupts

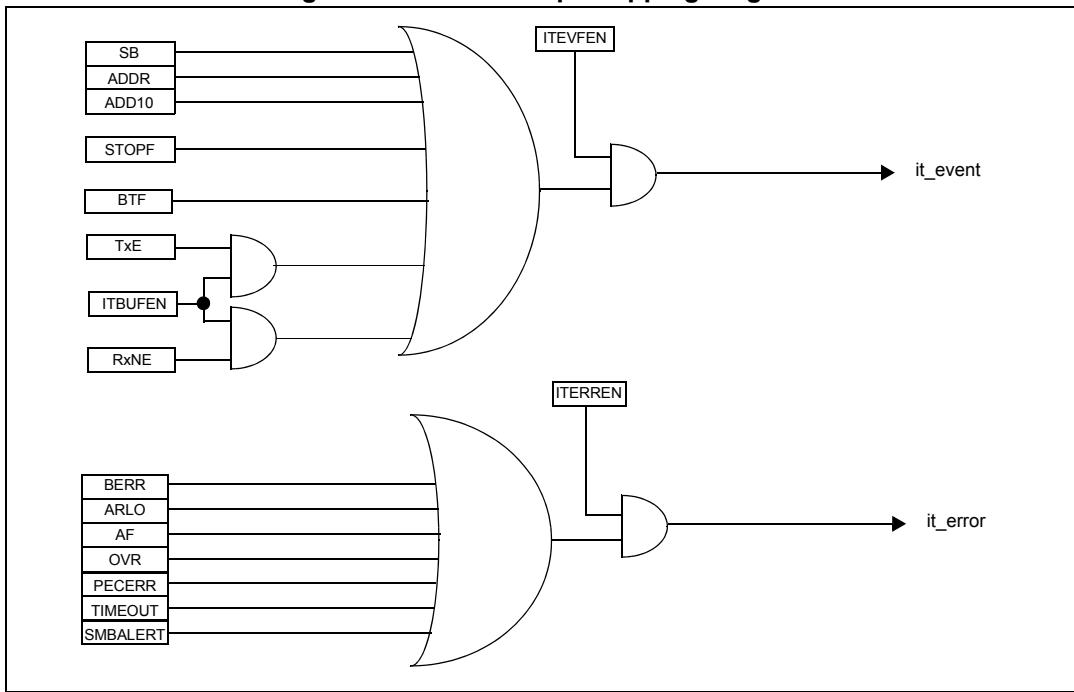
The table below gives the list of I<sup>2</sup>C interrupt requests.

**Table 141. I<sup>2</sup>C Interrupt requests**

Interrupt event	Event flag	Enable control bit
Start bit sent (Master)	SB	ITEVFEN
Address sent (Master) or Address matched (Slave)	ADDR	
10-bit header sent (Master)	ADD10	
Stop received (Slave)	STOPF	
Data byte transfer finished	BTF	
Receive buffer not empty	RxNE	ITEVFEN and ITBUFEN
Transmit buffer empty	TxE	
Bus error	BERR	ITERREN
Arbitration loss (Master)	ARLO	
Acknowledge failure	AF	
Overrun/Underrun	OVR	
PEC error	PECERR	
Timeout/Tlow error	TIMEOUT	
SMBus Alert	SMBALERT	

**Note:** SB, ADDR, ADD10, STOPF, BTF, RxNE and TxE are logically ORed on the same interrupt channel.

BERR, ARLO, AF, OVR, PECERR, TIMEOUT and SMBALERT are logically ORed on the same interrupt channel.

**Figure 277.  $I^2C$  interrupt mapping diagram**

## 24.5 I<sup>2</sup>C debug mode

When the microcontroller enters the debug mode (Cortex®-M4 with FPU core halted), the SMBUS timeout either continues to work normally or stops, depending on the DBG\_I2Cx\_SMBUS\_TIMEOUT configuration bits in the DBG module. For more details, refer to [Section 33.16.2: Debug support for timers, watchdog, bxCAN and I2C](#).

## 24.6 I<sup>2</sup>C registers

Refer to [Section 1.1 on page 51](#) for a list of abbreviations used in register descriptions.

The peripheral registers have to be accessed by half-words (16 bits) or words (32 bits).

### 24.6.1 I<sup>2</sup>C Control register 1 (I2C\_CR1)

Address offset: 0x00

Reset value: 0x0000

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SW RST	Res.	ALERT	PEC	POS	ACK	STOP	START	NO STRET CH	ENGC	ENPEC	ENARP	SMB TYPE	Res.	SM BUS	PE	
rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 **SWRST**: Software reset

When set, the I<sup>2</sup>C is under reset state. Before resetting this bit, make sure the I<sup>2</sup>C lines are released and the bus is free.

0: I<sup>2</sup>C Peripheral not under reset

1: I<sup>2</sup>C Peripheral under reset state

*Note:* This bit can be used to reinitialize the peripheral after an error or a locked state. As an example, if the BUSY bit is set and remains locked due to a glitch on the bus, the SWRST bit can be used to exit from this state.

Bit 14 Reserved, must be kept at reset value

Bit 13 **ALERT**: SMBus alert

This bit is set and cleared by software, and cleared by hardware when PE=0.

0: Releases SMBA pin high. Alert Response Address Header followed by NACK.

1: Drives SMBA pin low. Alert Response Address Header followed by ACK.

Bit 12 **PEC**: Packet error checking

This bit is set and cleared by software, and cleared by hardware when PEC is transferred or by a START or Stop condition or when PE=0.

0: No PEC transfer

1: PEC transfer (in Tx or Rx mode)

*Note:* PEC calculation is corrupted by an arbitration loss.

**Bit 11 POS:** Acknowledge/PEC Position (for data reception)

This bit is set and cleared by software and cleared by hardware when PE=0.

0: ACK bit controls the (N)ACK of the current byte being received in the shift register. The PEC bit indicates that current byte in shift register is a PEC.

1: ACK bit controls the (N)ACK of the next byte which will be received in the shift register. The PEC bit indicates that the next byte in the shift register is a PEC

*Note: The POS bit must be used only in 2-byte reception configuration in master mode. It must be configured before data reception starts, as described in the 2-byte reception procedure recommended in Master receiver.*

**Bit 10 ACK:** Acknowledge enable

This bit is set and cleared by software and cleared by hardware when PE=0.

0: No acknowledge returned

1: Acknowledge returned after a byte is received (matched address or data)

**Bit 9 STOP:** Stop generation

The bit is set and cleared by software, cleared by hardware when a Stop condition is detected, set by hardware when a timeout error is detected.

In Master Mode:

0: No Stop generation.

1: Stop generation after the current byte transfer or after the current Start condition is sent.

In Slave mode:

0: No Stop generation.

1: Release the SCL and SDA lines after the current byte transfer.

**Bit 8 START:** Start generation

This bit is set and cleared by software and cleared by hardware when start is sent or PE=0.

In Master Mode:

0: No Start generation

1: Repeated start generation

In Slave mode:

0: No Start generation

1: Start generation when the bus is free

**Bit 7 NOSTRETCH:** Clock stretching disable (Slave mode)

This bit is used to disable clock stretching in slave mode when ADDR or BTF flag is set, until it is reset by software.

0: Clock stretching enabled

1: Clock stretching disabled

**Bit 6 ENGC:** General call enable

0: General call disabled. Address 00h is NACKed.

1: General call enabled. Address 00h is ACKed.

**Bit 5 ENPEC:** PEC enable

0: PEC calculation disabled

1: PEC calculation enabled

**Bit 4 ENARP:** ARP enable

0: ARP disable

1: ARP enable

SMBus Device default address recognized if SMBTYPE=0

SMBus Host address recognized if SMBTYPE=1

**Bit 3 SMBTYPE:** SMBus type

0: SMBus Device

1: SMBus Host

Bit 2 Reserved, must be kept at reset value

Bit 1 **SMBUS**: SMBus mode

0: I<sup>2</sup>C mode

1: SMBus mode

Bit 0 **PE**: Peripheral enable

0: Peripheral disable

1: Peripheral enable

*Note: If this bit is reset while a communication is on going, the peripheral is disabled at the end of the current communication, when back to IDLE state.*

*All bit resets due to PE=0 occur at the end of the communication.*

*In master mode, this bit must not be reset before the end of the communication.*

**Note:** When the STOP, START or PEC bit is set, the software must not perform any write access to I2C\_CR1 before this bit is cleared by hardware. Otherwise there is a risk of setting a second STOP, START or PEC request.

## 24.6.2 I<sup>2</sup>C Control register 2 (I2C\_CR2)

Address offset: 0x04

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	LAST	DMA EN	ITBUF EN	ITEVT EN	ITERR EN	Res.	Res.			FREQ[5:0]			
			rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw

Bits 15:13 Reserved, must be kept at reset value

Bit 12 **LAST**: DMA last transfer

0: Next DMA EOT is not the last transfer

1: Next DMA EOT is the last transfer

*Note: This bit is used in master receiver mode to permit the generation of a NACK on the last received data.*

Bit 11 **DMAEN**: DMA requests enable

0: DMA requests disabled

1: DMA request enabled when TxE=1 or RxNE =1

Bit 10 **ITBUFEN**: Buffer interrupt enable

0: TxE = 1 or RxNE = 1 does not generate any interrupt.

1: TxE = 1 or RxNE = 1 generates Event Interrupt (whatever the state of DMAEN)

Bit 9 **ITEVTEN**: Event interrupt enable

0: Event interrupt disabled

1: Event interrupt enabled

This interrupt is generated when:

- SB = 1 (Master)
- ADDR = 1 (Master/Slave)
- ADD10= 1 (Master)
- STOPF = 1 (Slave)
- BTF = 1 with no TxE or RxNE event
- TxE event to 1 if ITBUFEN = 1
- RxNE event to 1if ITBUFEN = 1

**ITERREN:** Error interrupt enable

- 0: Error interrupt disabled
- 1: Error interrupt enabled

This interrupt is generated when:

- BERR = 1
- ARLO = 1
- AF = 1
- OVR = 1
- PECERR = 1
- TIMEOUT = 1
- SMBALERT = 1

Bits 7:6 Reserved, must be kept at reset value

Bits 5:0 **FREQ[5:0]:** Peripheral clock frequency

The FREQ bits must be configured with the APB clock frequency value (I2C peripheral connected to APB). The FREQ field is used by the peripheral to generate data setup and hold times compliant with the I2C specifications. The minimum allowed frequency is 2 MHz, the maximum frequency is limited by the maximum APB frequency (42 MHz) and cannot exceed 50 MHz (peripheral intrinsic maximum limit).

0b000000: Not allowed

0b000001: Not allowed

0b000010: 2 MHz

...

0b110010: 50 MHz

Higher than 0b101010: Not allowed

### 24.6.3 I<sup>2</sup>C Own address register 1 (I2C\_OAR1)

Address offset: 0x08

Reset value: 0x0000

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD MODE	Res.	Res.	Res.	Res.	Res.	ADD[9:8]		ADD[7:1]								ADD0
	rw					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 **ADDMODE**: Addressing mode (slave mode)

0: 7-bit slave address (10-bit address not acknowledged)

1: 10-bit slave address (7-bit address not acknowledged)

Bit 14 Should always be kept at 1 by software.

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

Bits 9:8 **ADD[9:8]**: Interface address

7-bit addressing mode: don't care

10-bit addressing mode: bits 9:8 of address

Bits 7:1 **ADD[7:1]**: Interface address

bits 7:1 of address

Bit 0 **ADD0**: Interface address

7-bit addressing mode: don't care

10-bit addressing mode: bit 0 of address

### 24.6.4 I<sup>2</sup>C Own address register 2 (I2C\_OAR2)

Address offset: 0x0C

Reset value: 0x0000

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res.	ADD2[7:1]								EN DUAL								
									rw	rw	rw	rw	rw	rw	rw	rw	

Bits 15:8 Reserved, must be kept at reset value

Bits 7:1 **ADD2[7:1]**: Interface address

bits 7:1 of address in dual addressing mode

Bit 0 **ENDUAL**: Dual addressing mode enable

0: Only OAR1 is recognized in 7-bit addressing mode

1: Both OAR1 and OAR2 are recognized in 7-bit addressing mode

### 24.6.5 I<sup>2</sup>C Data register (I2C\_DR)

Address offset: 0x10

Reset value: 0x0000

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

Bits 15:8 Reserved, must be kept at reset value

Bits 7:0 **DR[7:0]** 8-bit data register

Byte received or to be transmitted to the bus.

- Transmitter mode: Byte transmission starts automatically when a byte is written in the DR register. A continuous transmit stream can be maintained if the next data to be transmitted is put in DR once the transmission is started (TxE=1)
- Receiver mode: Received byte is copied into DR (RxNE=1). A continuous transmit stream can be maintained if DR is read before the next data byte is received (RxNE=1).

*Note: In slave mode, the address is not copied into DR.*

*Write collision is not managed (DR can be written if TxE=0).*

*If an ARLO event occurs on ACK pulse, the received byte is not copied into DR and so cannot be read.*

### 24.6.6 I<sup>2</sup>C Status register 1 (I2C\_SR1)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIMEO UT	Res.	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res.	STOPF	ADD10	BTF	ADDR	SB
rc_w0	rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	r	r		r	r	r	r	r

Bit 15 **SMBALERT**: SMBus alert

In SMBus host mode:

0: no SMBALERT

1: SMBALERT event occurred on pin

In SMBus slave mode:

0: no SMBALERT response address header

1: SMBALERT response address header to SMBALERT LOW received

- Cleared by software writing 0, or by hardware when PE=0.

Bit 14 **TIMEOUT**: Timeout or Tlow error

0: No timeout error

1: SCL remained LOW for 25 ms (Timeout)

or

Master cumulative clock low extend time more than 10 ms (Tlow:mext)

or

Slave cumulative clock low extend time more than 25 ms (Tlow:sext)

- When set in slave mode: slave resets the communication and lines are released by hardware
- When set in master mode: Stop condition sent by hardware
- Cleared by software writing 0, or by hardware when PE=0.

*Note: This functionality is available only in SMBus mode.*

## Bit 13 Reserved, must be kept at reset value

Bit 12 **PECERR**: PEC Error in reception

0: no PEC error: receiver returns ACK after PEC reception (if ACK=1)

1: PEC error: receiver returns NACK after PEC reception (whatever ACK)

- Cleared by software writing 0, or by hardware when PE=0.
- Note: When the received CRC is wrong, PECERR is not set in slave mode if the PEC control bit is not set before the end of the CRC reception. Nevertheless, reading the PEC value determines whether the received CRC is right or wrong.

Bit 11 **OVR**: Overrun/Underrun

0: No overrun/underrun

1: Overrun or underrun

- Set by hardware in slave mode when NOSTRETCH=1 and:
- In reception when a new byte is received (including ACK pulse) and the DR register has not been read yet. New received byte is lost.
- In transmission when a new byte should be sent and the DR register has not been written yet. The same byte is sent twice.
- Cleared by software writing 0, or by hardware when PE=0.

*Note: If the DR write occurs very close to SCL rising edge, the sent data is unspecified and a hold timing error occurs*

Bit 10 **AF**: Acknowledge failure

0: No acknowledge failure

1: Acknowledge failure

- Set by hardware when no acknowledge is returned.

- Cleared by software writing 0, or by hardware when PE=0.

Bit 9 **ARLO**: Arbitration lost (master mode)

0: No Arbitration Lost detected

1: Arbitration Lost detected

Set by hardware when the interface loses the arbitration of the bus to another master

– Cleared by software writing 0, or by hardware when PE=0.

After an ARLO event the interface switches back automatically to Slave mode (MSL=0).

*Note: In SMBUS, the arbitration on the data in slave mode occurs only during the data phase, or the acknowledge transmission (not on the address acknowledge).*

Bit 8 **BERR**: Bus error

0: No misplaced Start or Stop condition

1: Misplaced Start or Stop condition

– Set by hardware when the interface detects an SDA rising or falling edge while SCL is high, occurring in a non-valid position during a byte transfer.

– Cleared by software writing 0, or by hardware when PE=0.

Bit 7 **TxE**: Data register empty (transmitters)

0: Data register not empty

1: Data register empty

– Set when DR is empty in transmission. TxE is not set during address phase.

– Cleared by software writing to the DR register or by hardware after a start or a stop condition or when PE=0.

TxE is not set if either a NACK is received, or if next byte to be transmitted is PEC (PEC=1)

*Note: TxE is not cleared by writing the first data being transmitted, or by writing data when BTF is set, as in both cases the data register is still empty.*

Bit 6 **RxNE**: Data register not empty (receivers)

0: Data register empty

1: Data register not empty

– Set when data register is not empty in receiver mode. RxNE is not set during address phase.

– Cleared by software reading or writing the DR register or by hardware when PE=0.

RxNE is not set in case of ARLO event.

*Note: RxNE is not cleared by reading data when BTF is set, as the data register is still full.*

Bit 5 Reserved, must be kept at reset value

Bit 4 **STOPF**: Stop detection (slave mode)

0: No Stop condition detected

1: Stop condition detected

– Set by hardware when a Stop condition is detected on the bus by the slave after an acknowledge (if ACK=1).

– Cleared by software reading the SR1 register followed by a write in the CR1 register, or by hardware when PE=0

*Note: The STOPF bit is not set after a NACK reception.*

*It is recommended to perform the complete clearing sequence (READ SR1 then WRITE CR1) after the STOPF is set. Refer to Figure 274: Transfer sequence diagram for slave receiver on page 765.*

Bit 3 **ADD10**: 10-bit header sent (Master mode)

- 0: No ADD10 event occurred.
- 1: Master has sent first address byte (header).
- Set by hardware when the master has sent the first byte in 10-bit address mode.
- Cleared by software reading the SR1 register followed by a write in the DR register of the second address byte, or by hardware when PE=0.

*Note: ADD10 bit is not set after a NACK reception*

Bit 2 **BTF**: Byte transfer finished

- 0: Data byte transfer not done
- 1: Data byte transfer succeeded
- Set by hardware when NOSTRETCH=0 and:
- In reception when a new byte is received (including ACK pulse) and DR has not been read yet (RxNE=1).
- In transmission when a new byte should be sent and DR has not been written yet (TxEN=1).
- Cleared by software by either a read or write in the DR register or by hardware after a start or a stop condition in transmission or when PE=0.

*Note: The BTF bit is not set after a NACK reception*

*The BTF bit is not set if next byte to be transmitted is the PEC (TRA=1 in I2C\_SR2 register and PEC=1 in I2C\_CR1 register)*

Bit 1 **ADDR**: Address sent (master mode)/matched (slave mode)

This bit is cleared by software reading SR1 register followed reading SR2, or by hardware when PE=0.

Address matched (Slave)

- 0: Address mismatched or not received.
- 1: Received address matched.

- Set by hardware as soon as the received slave address matched with the OAR registers content or a general call or a SMBus Device Default Address or SMBus Host or SMBus Alert is recognized. (when enabled depending on configuration).

*Note: In slave mode, it is recommended to perform the complete clearing sequence (READ SR1 then READ SR2) after ADDR is set. Refer to [Figure 274: Transfer sequence diagram for slave receiver on page 765](#).*

Address sent (Master)

- 0: No end of address transmission
- 1: End of address transmission
- For 10-bit addressing, the bit is set after the ACK of the 2nd byte.
- For 7-bit addressing, the bit is set after the ACK of the byte.

*Note: ADDR is not set after a NACK reception*

Bit 0 **SB**: Start bit (Master mode)

- 0: No Start condition
- 1: Start condition generated.
- Set when a Start condition generated.
- Cleared by software by reading the SR1 register followed by writing the DR register, or by hardware when PE=0

### 24.6.7 I<sup>2</sup>C Status register 2 (I<sup>2</sup>C\_SR2)

Address offset: 0x18

Reset value: 0x0000

**Note:** *Reading I<sup>2</sup>C\_SR2 after reading I<sup>2</sup>C\_SR1 clears the ADDR flag, even if the ADDR flag was set after reading I<sup>2</sup>C\_SR1. Consequently, I<sup>2</sup>C\_SR2 must be read only when ADDR is found set in I<sup>2</sup>C\_SR1 or when the STOPF bit is cleared.*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PEC[7:0]								DUALF	SMB HOST	SMB DEFALT	GEN CALL	Res.	TRA	BUSY	MSL
r	r	r	r	r	r	r	r	r	r	r	r		r	r	r

Bits 15:8 **PEC[7:0]** Packet error checking register

This register contains the internal PEC when ENPEC=1.

Bit 7 **DUALF**: Dual flag (Slave mode)

0: Received address matched with OAR1

1: Received address matched with OAR2

– Cleared by hardware after a Stop condition or repeated Start condition, or when PE=0.

Bit 6 **SMBHOST**: SMBus host header (Slave mode)

0: No SMBus Host address

1: SMBus Host address received when SMBTYPE=1 and ENARP=1.

– Cleared by hardware after a Stop condition or repeated Start condition, or when PE=0.

Bit 5 **SMBDEFAULT**: SMBus device default address (Slave mode)

0: No SMBus Device Default address

1: SMBus Device Default address received when ENARP=1

– Cleared by hardware after a Stop condition or repeated Start condition, or when PE=0.

Bit 4 **GENCALL**: General call address (Slave mode)

0: No General Call

1: General Call Address received when ENGC=1

– Cleared by hardware after a Stop condition or repeated Start condition, or when PE=0.

Bit 3 Reserved, must be kept at reset value

Bit 2 **TRA**: Transmitter/receiver

- 0: Data bytes received
- 1: Data bytes transmitted

This bit is set depending on the R/W bit of the address byte, at the end of total address phase.

It is also cleared by hardware after detection of Stop condition (STOPF=1), repeated Start condition, loss of bus arbitration (ARLO=1), or when PE=0.

Bit 1 **BUSY**: Bus busy

- 0: No communication on the bus
- 1: Communication ongoing on the bus
- Set by hardware on detection of SDA or SCL low
- Cleared by hardware on detection of a Stop condition.

It indicates a communication in progress on the bus. This information is still updated when the interface is disabled (PE=0).

Bit 0 **MSL**: Master/slave

- 0: Slave Mode
- 1: Master Mode
- Set by hardware as soon as the interface is in Master mode (SB=1).
- Cleared by hardware after detecting a Stop condition on the bus or a loss of arbitration (ARLO=1), or by hardware when PE=0.

**Note:** *Reading I<sup>2</sup>C\_SR2 after reading I<sup>2</sup>C\_SR1 clears the ADDR flag, even if the ADDR flag was set after reading I<sup>2</sup>C\_SR1. Consequently, I<sup>2</sup>C\_SR2 must be read only when ADDR is found set in I<sup>2</sup>C\_SR1 or when the STOPF bit is cleared.*

#### 24.6.8 I<sup>2</sup>C Clock control register (I<sup>2</sup>C\_CCR)

Address offset: 0x1C

Reset value: 0x0000

**Note:** *f<sub>PCLK1</sub> must be at least 2 MHz to achieve Sm mode I<sup>2</sup>C frequencies. It must be at least 4 MHz to achieve Fm mode I<sup>2</sup>C frequencies. It must be a multiple of 10MHz to reach the 400 kHz maximum I<sup>2</sup>C Fm mode clock.*

*The CCR register must be configured only when the I<sup>2</sup>C is disabled (PE = 0).*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
F/S	DUTY	Res.	Res.	CCR[11:0]													
rw	rw			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit 15 **F/S**: I<sup>2</sup>C master mode selection

- 0: Sm mode I<sup>2</sup>C
- 1: Fm mode I<sup>2</sup>C

Bit 14 **DUTY**: Fm mode duty cycle  
 0: Fm mode  $t_{low}/t_{high} = 2$   
 1: Fm mode  $t_{low}/t_{high} = 16/9$  (see CCR)

Bits 13:12 Reserved, must be kept at reset value

Bits 11:0 **CCR[11:0]**: Clock control register in Fm/Sm mode (Master mode)  
 Controls the SCL clock in master mode.

Sm mode or SMBus:

$$T_{high} = CCR * T_{PCLK1}$$

$$T_{low} = CCR * T_{PCLK1}$$

Fm mode:

If DUTY = 0:  
 $T_{high} = CCR * T_{PCLK1}$   
 $T_{low} = 2 * CCR * T_{PCLK1}$

If DUTY = 1: (to reach 400 kHz)  
 $T_{high} = 9 * CCR * T_{PCLK1}$   
 $T_{low} = 16 * CCR * T_{PCLK1}$

For instance: in Sm mode, to generate a 100 kHz SCL frequency:  
 If FREQR = 08,  $T_{PCLK1} = 125$  ns so CCR must be programmed with 0x28  
 (0x28 => 40d x 125 ns = 5000 ns.)

*Note: The minimum allowed value is 0x04, except in FAST DUTY mode where the minimum allowed value is 0x01*

$t_{high} = t_r(SCL) + t_w(SCLH)$ . See device datasheet for the definitions of parameters.  
 $t_{low} = t_f(SCL) + t_w(SCLL)$ . See device datasheet for the definitions of parameters.  
*I<sup>2</sup>C communication speed, f<sub>SCL</sub> ~ 1/(t<sub>high</sub> + t<sub>low</sub>). The real frequency may differ due to the analog noise filter input delay.*

*The CCR register must be configured only when the I<sup>2</sup>C is disabled (PE = 0).*

## 24.6.9 I<sup>2</sup>C TRISE register (I2C\_TRISE)

Address offset: 0x20

Reset value: 0x0002

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

Bits 15:6 Reserved, must be kept at reset value

Bits 5:0 **TRISE[5:0]**: Maximum rise time in Fm/Sm mode (Master mode)

These bits should provide the maximum duration of the SCL feedback loop in master mode.  
 The purpose is to keep a stable SCL frequency whatever the SCL rising edge duration.

These bits must be programmed with the maximum SCL rise time given in the I<sup>2</sup>C bus specification, incremented by 1.

For instance: in Sm mode, the maximum allowed SCL rise time is 1000 ns.

If, in the I2C\_CR2 register, the value of FREQR[5:0] bits is equal to 0x08 and  $T_{PCLK1} = 125$  ns therefore the TRISE[5:0] bits must be programmed with 09h.

(1000 ns / 125 ns = 8 + 1)

The filter value can also be added to TRISE[5:0].

If the result is not an integer, TRISE[5:0] must be programmed with the integer part, in order to respect the  $t_{HIGH}$  parameter.

*Note: TRISE[5:0] must be configured only when the I<sup>2</sup>C is disabled (PE = 0).*

### 24.6.10 I<sup>2</sup>C FLTR register (I2C\_FLTR)

Address offset: 0x24

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	ANOFF	DNF[3:0]													

Bits 15:5 Reserved, must be kept at reset value

Bit 4 **ANOFF**: Analog noise filter OFF

- 0: Analog noise filter enable
- 1: Analog noise filter disable

*Note:* ANOFF must be configured only when the I2C is disabled (PE = 0).

Bits 3:0 **DNF[3:0]**: Digital noise filter

These bits are used to configure the digital noise filter on SDA and SCL inputs. The digital filter will suppress the spikes with a length of up to DNF[3:0] \* TPCLK1.

0000: Digital noise filter disable

0001: Digital noise filter enabled and filtering capability up to 1\* TPCLK1.

...

1111: Digital noise filter enabled and filtering capability up to 15\* TPCLK1.

*Note:* DNF[3:0] must be configured only when the I2C is disabled (PE = 0). If the analog filter is also enabled, the digital filter is added to the analog filter.

### 24.6.11 I<sup>2</sup>C register map

The table below provides the I<sup>2</sup>C register map and reset values.

**Table 142. I<sup>2</sup>C 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	I2C_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.	Res.	Res.	Res.	Res.	Res.	Res.
0x04	I2C_CR2		Reset value	Res.																																				
0x08	I2C_OAR1		Reset value	Res.																																				
0x0C	I2C_OAR2		Reset value	Res.																																				
0x10	I2C_DR		Reset value	Res.																																				
0x14	I2C_SR1		Reset value	Res.																																				
0x18	I2C_SR2		Reset value	Res.																																				
0x1C	I2C_CCR		Reset value	Res.																																				
0x20	I2C_TRISE		Reset value	Res.																																				
0x24	I2C_FLTR		Reset value	Res.																																				

Refer to [Section 2.2.2](#) for the register boundary addresses.

## 25 Universal synchronous asynchronous receiver transmitter (USART)

### 25.1 USART introduction

The universal synchronous asynchronous receiver transmitter (USART) offers a flexible means of full-duplex data exchange with external equipment requiring an industry standard NRZ asynchronous serial data format. The USART offers a very wide range of baud rates using a fractional baud rate generator.

It supports synchronous one-way communication and half-duplex single wire communication. It also supports the LIN (local interconnection network), Smartcard Protocol and IrDA (infrared data association) SIR ENDEC specifications, and modem operations (CTS/RTS). It allows multiprocessor communication.

High speed data communication is possible by using the DMA for multibuffer configuration.

## 25.2 USART main features

- Full duplex, asynchronous communications
- NRZ standard format (Mark/Space)
- Configurable oversampling method by 16 or by 8 to give flexibility between speed and clock tolerance
- Fractional baud rate generator systems
  - Common programmable transmit and receive baud rate (refer to the datasheets for the value of the baud rate at the maximum APB frequency).
- Programmable data word length (8 or 9 bits)
- Configurable stop bits - support for 1 or 2 stop bits
- LIN Master Synchronous Break send capability and LIN slave break detection capability
  - 13-bit break generation and 10/11 bit break detection when USART is hardware configured for LIN
- Transmitter clock output for synchronous transmission
- IrDA SIR encoder decoder
  - Support for 3/16 bit duration for normal mode
- Smartcard emulation capability
  - The Smartcard interface supports the asynchronous protocol Smartcards as defined in the ISO 7816-3 standards
  - 0.5, 1.5 stop bits for Smartcard operation
- Single-wire half-duplex communication
- Configurable multibuffer communication using DMA (direct memory access)
  - Buffering of received/transmitted bytes in reserved SRAM using centralized DMA
- Separate enable bits for transmitter and receiver
- Transfer detection flags:
  - Receive buffer full
  - Transmit buffer empty
  - End of transmission flags
- Parity control:
  - Transmits parity bit
  - Checks parity of received data byte
- Four error detection flags:
  - Overrun error
  - Noise detection
  - Frame error
  - Parity error
- Ten interrupt sources with flags:
  - CTS changes
  - LIN break detection
  - Transmit data register empty
  - Transmission complete

- Receive data register full
- Idle line received
- Overrun error
- Framing error
- Noise error
- Parity error
- Multiprocessor communication - enter into mute mode if address match does not occur
- Wake up from mute mode (by idle line detection or address mark detection)
- Two receiver wakeup modes: Address bit (MSB, 9<sup>th</sup> bit), Idle line

## 25.3 USART implementation

This section describes the full set of features implemented in USART1. Refer to [Table 143: USART features](#) for the differences between USART instances.

**Table 143. USART features**

USART modes/features <sup>(1)</sup>	USART1, USART2, USART3, USART6	UART4, UART5
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
USART data length	8 or 9 bits	

1. X = supported.

## 25.4 USART functional description

The interface is externally connected to another device by three pins (see [Figure 278](#)). Any USART bidirectional communication requires a minimum of two pins: Receive Data In (RX) and Transmit Data Out (TX):

**RX:** Receive Data Input is the serial data input. Oversampling techniques are used for data recovery by discriminating between valid incoming data and noise.

**TX:** Transmit Data Output. When the transmitter is disabled, the output pin returns to its I/O port configuration. When the transmitter is enabled and nothing is to be transmitted, the TX pin is at high level. In single-wire and smartcard modes, this I/O is used to transmit and receive the data (at USART level, data are then received on SW\_RX).

Through these pins, serial data is transmitted and received in normal USART mode as frames comprising:

- An Idle Line prior to transmission or reception
- A start bit
- A data word (8 or 9 bits) least significant bit first
- 0.5, 1, 1.5, 2 Stop bits indicating that the frame is complete
- This interface uses a fractional baud rate generator - with a 12-bit mantissa and 4-bit fraction
- A status register (USART\_SR)
- Data Register (USART\_DR)
- A baud rate register (USART\_BRR) - 12-bit mantissa and 4-bit fraction.
- A Guardtime Register (USART\_GTPR) in case of Smartcard mode.

Refer to [Section 25.6: USART registers](#) for the definition of each bit.

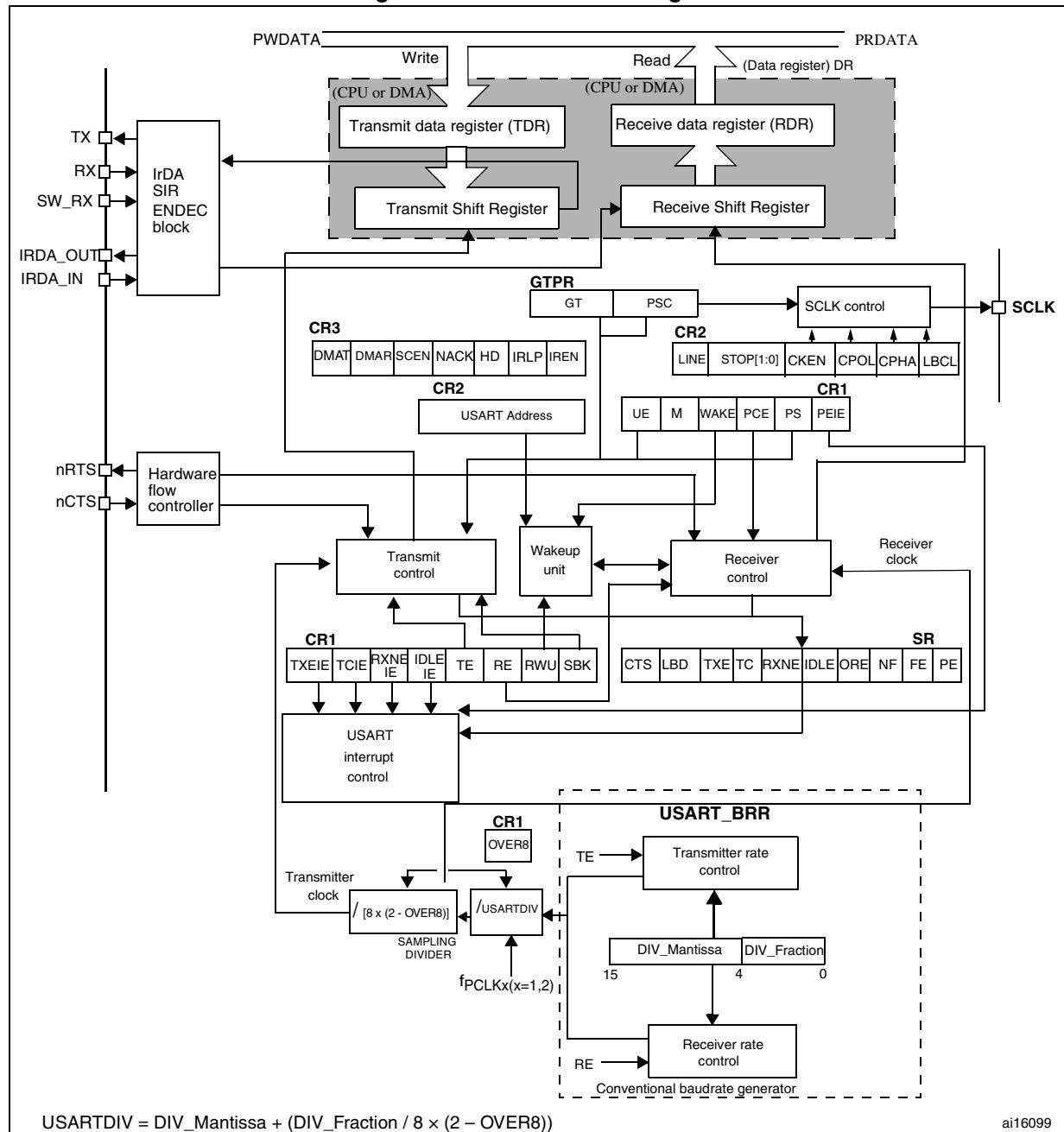
The following pin is required to interface in synchronous mode:

- **SCLK:** Transmitter clock output. This pin outputs the transmitter data clock for synchronous transmission corresponding to SPI master mode (no clock pulses on start bit and stop bit, and a software option to send a clock pulse on the last data bit). In parallel data can be received synchronously on RX. This can be used to control peripherals that have shift registers (e.g. LCD drivers). The clock phase and polarity are software programmable. In smartcard mode, SCLK can provide the clock to the smartcard.

The following pins are required in Hardware flow control mode:

- **nCTS:** Clear To Send blocks the data transmission at the end of the current transfer when high
- **nRTS:** Request to send indicates that the USART is ready to receive a data (when low).

Figure 278. USART block diagram



### 25.4.1 USART character description

Word length may be selected as being either 8 or 9 bits by programming the M bit in the USART\_CR1 register (see [Figure 279](#)).

The TX pin is in low state during the start bit. It is in high state during the stop bit.

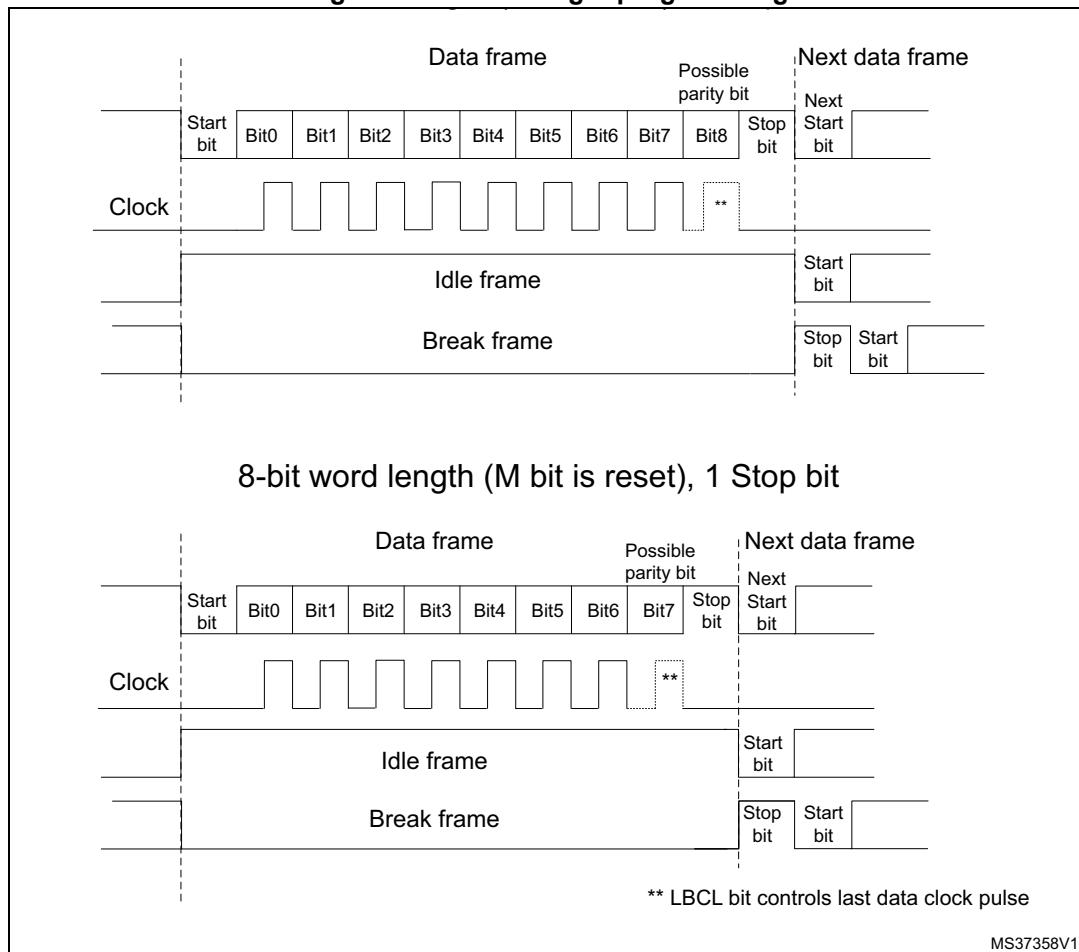
An **Idle character** is interpreted as an entire frame of “1”s followed by the start bit of the next frame which contains data (The number of “1”’s will include the number of stop bits).

A **Break character** is interpreted on receiving “0”s for a frame period. At the end of the break frame the transmitter inserts either 1 or 2 stop bits (logic “1” bit) to acknowledge the start bit.

Transmission and reception are driven by a common baud rate generator, the clock for each is generated when the enable bit is set respectively for the transmitter and receiver.

The details of each block is given below.

**Figure 279. Word length programming**



## 25.4.2 Transmitter

The transmitter can send data words of either 8 or 9 bits depending on the M bit status. When the transmit enable bit (TE) is set, the data in the transmit shift register is output on the TX pin and the corresponding clock pulses are output on the SCLK pin.

### Character transmission

During an USART transmission, data shifts out least significant bit first on the TX pin. In this mode, the USART\_DR register consists of a buffer (TDR) between the internal bus and the transmit shift register (see [Figure 278](#)).

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 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

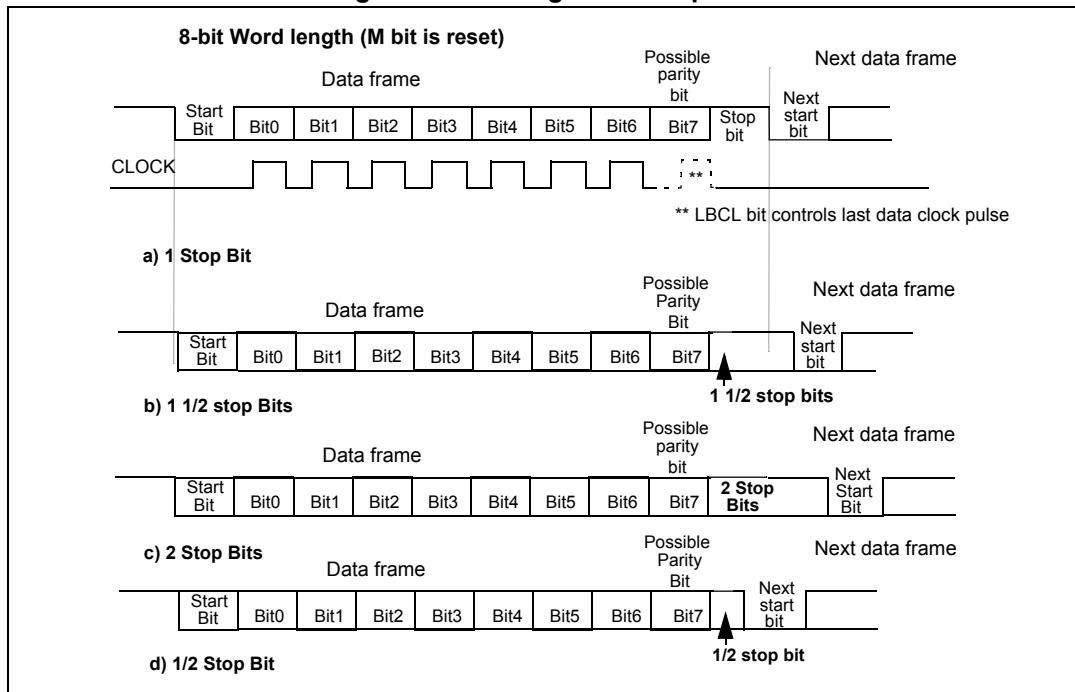
The number of stop bits to be transmitted with every character can be programmed in Control register 2, bits 13,12.

- **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.
- **0.5 stop bit:** To be used when receiving data in Smartcard mode.
- **1.5 stop bits:** To be used when transmitting and receiving data in Smartcard mode.

An idle frame transmission will include the stop bits.

A break transmission will be 10 low bits followed by the configured number of stop bits (when m = 0) and 11 low bits followed by the configured number of stop bits (when m = 1). It is not possible to transmit long breaks (break of length greater than 10/11 low bits).

Figure 280. Configurable stop bits



Procedure:

1. Enable the USART by writing the UE bit in USART\_CR1 register to 1.
2. Program the M bit in USART\_CR1 to define the word length.
3. Program the number of stop bits in USART\_CR2.
4. Select DMA enable (DMAT) in USART\_CR3 if Multi buffer Communication is to take place. Configure the DMA register as explained in multibuffer communication.
5. Select the desired baud rate using the USART\_BRR register.
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\_DR 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\_DR 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 data register.

The TXE bit is set by hardware and it indicates:

- The data has been moved from TDR to the shift register and the data transmission has started.
- The TDR register is empty.
- The next data can be written in the USART\_DR register without overwriting the previous data.

This flag generates an interrupt if the TXEIE bit is set.

When a transmission is taking place, a write instruction to the USART\_DR register stores the data in the TDR register and which is copied in the shift register at the end of the current transmission.

When no transmission is taking place, a write instruction to the USART\_DR register places the data directly in the shift register, the data transmission starts, and the TXE bit is immediately set.

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.

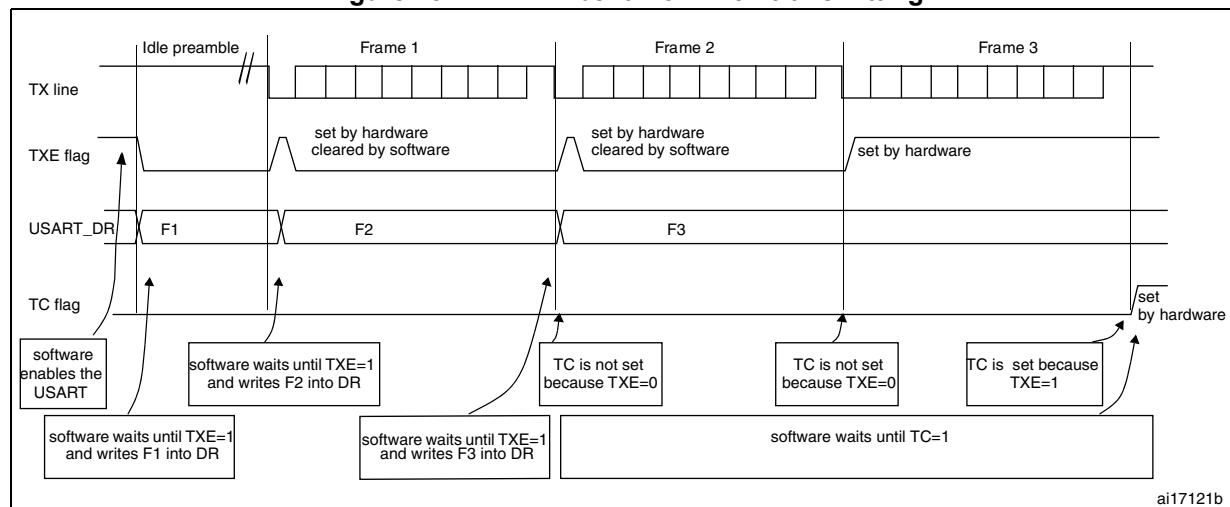
After writing the last data into the USART\_DR register, it is mandatory to wait for TC=1 before disabling the USART or causing the microcontroller to enter the low power mode (see [Figure 281: TC/TXE behavior when transmitting](#)).

The TC bit is cleared by the following software sequence:

1. A read from the USART\_SR register
2. A write to the USART\_DR register

**Note:** *The TC bit can also be cleared by writing a '0' to it. This clearing sequence is recommended only for Multibuffer communication.*

**Figure 281. TC/TXE behavior when transmitting**



### Break characters

Setting the SBK bit transmits a break character. The break frame length depends on the M bit (see [Figure 279](#)).

If the SBK bit is set to '1' a break character is sent on the TX line after completing the current character transmission. This bit is reset by hardware when the break character is completed (during the stop bit of the break character). The USART inserts a logic 1 bit at the end of the last break frame to guarantee the recognition of the start bit of the next frame.

**Note:** *If the software resets the SBK bit before the commencement of break transmission, the break character will not be transmitted. For two consecutive breaks, the SBK bit should be set after the stop bit of the previous break.*

## Idle characters

Setting the TE bit drives the USART to send an idle frame before the first data frame.

### 25.4.3 Receiver

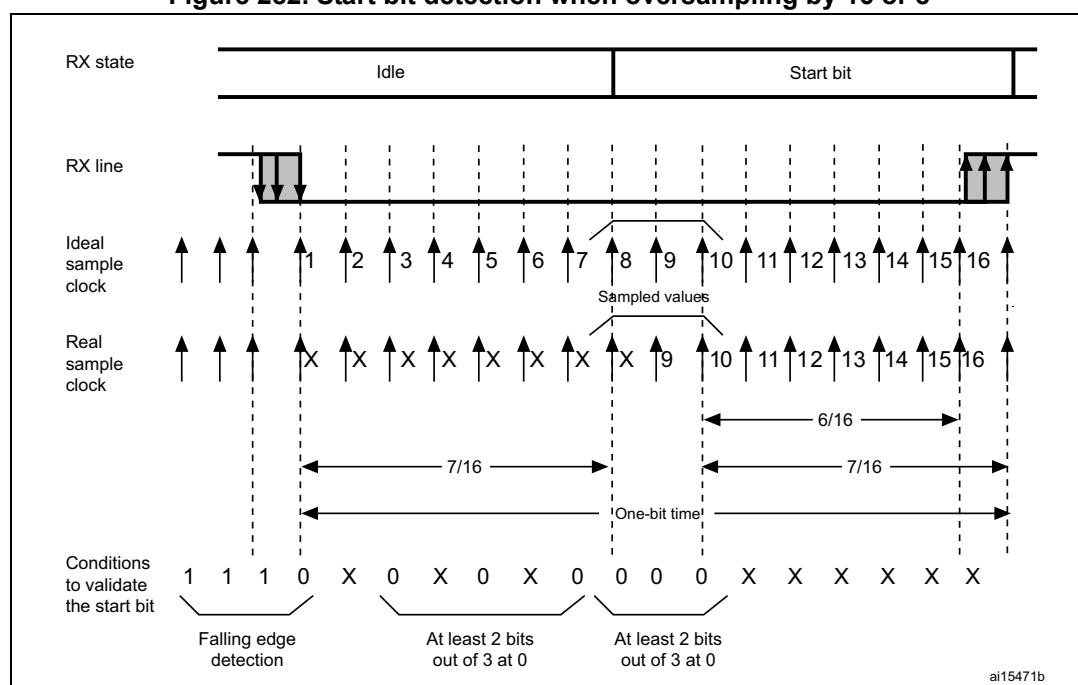
The USART can receive data words of either 8 or 9 bits depending on the M bit in the USART\_CR1 register.

#### 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 0 X 0 0 0 0.

**Figure 282. 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 NE noise flag is set if, for both samplings, at least 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). If this condition is not met, the start detection aborts and the receiver returns to the idle state (no flag is set).

If, 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, the start bit is validated but the NE noise flag bit is set.

## Character reception

During an USART reception, data shifts in least significant bit first through the RX pin. In this mode, the USART\_DR register consists of a buffer (RDR) between the internal bus and the received shift register.

Procedure:

1. Enable the USART by writing the UE bit in USART\_CR1 register to 1.
2. Program the M bit in USART\_CR1 to define the word length.
3. Program the number of stop bits in USART\_CR2.
4. Select DMA enable (DMAR) in USART\_CR3 if multibuffer communication is to take place. Configure the DMA register as explained in multibuffer communication. STEP 3
5. Select the desired baud rate using the baud rate register USART\_BRR
6. Set the RE bit USART\_CR1. This enables the receiver which begins searching for a start bit.

When a character is received

- The RXNE bit is set. It indicates that the content of the shift register is transferred to the RDR. In other words, data has been received and can be read (as well as its associated error flags).
- An interrupt is generated if the RXNEIE bit is set.
- The error flags can be set if a frame error, noise or an overrun error has been detected during reception.
- In multibuffer, RXNE is set after every byte received and is cleared by the DMA read to the Data Register.
- In single buffer mode, clearing the RXNE bit is performed by a software read to the USART\_DR register. The RXNE flag can also be cleared by writing a zero to it. The RXNE bit must be cleared before the end of the reception of the next character to avoid an overrun error.

Note:

*The RE bit should not be reset while receiving data. If the RE bit is disabled during reception, the reception of the current byte will be aborted.*

## Break character

When a break character is received, the USART handles it as a framing error.

## Idle character

When an idle frame is detected, there is the same procedure as a data received character plus an interrupt if the IDLEIE bit is set.

## Overrun error

An overrun error occurs when a character is received when RXNE has not been reset. Data can not be transferred from the shift register to the RDR register until the RXNE bit is cleared.

The RXNE flag is set after every byte received. An overrun error occurs if RXNE flag is set when the next data is received or the previous DMA request has not been serviced. When an overrun error occurs:

- The ORE bit is set.
- The RDR content will not be lost. The previous data is available when a read to USART\_DR is performed.
- The shift register will be overwritten. After that point, any data received during overrun is lost.
- An interrupt is generated if either the RXNEIE bit is set or both the EIE and DMAR bits are set.
- The ORE bit is reset by a read to the USART\_SR register followed by a USART\_DR register read operation.

**Note:** *The ORE bit, when set, indicates that at least 1 data has been lost. There are two possibilities:*

- if RXNE=1, then the last valid data is stored in the receive register RDR and can be read,
- if RXNE=0, then it means that the last valid data has already been read and thus there is nothing to be read in the RDR. This case can occur when the last valid data is read in the RDR at the same time as the new (and lost) data is received. It may also occur when the new data is received during the reading sequence (between the USART\_SR register read access and the USART\_DR read access).

### Selecting the proper oversampling method

The receiver implements different user-configurable oversampling techniques (except in synchronous mode) for data recovery by discriminating between valid incoming data and noise.

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 283](#) and [Figure 284](#)).

Depending on the application:

- select oversampling by 8 (OVER8=1) to achieve higher speed (up to  $f_{PCLK}/8$ ). In this case the maximum receiver tolerance to clock deviation is reduced (refer to [Section 25.4.5: USART receiver tolerance to clock deviation](#))
- 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  $f_{PCLK}/16$

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 144](#)) 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 tolerance to clock deviations (see [Section 25.4.5: USART receiver tolerance to clock deviation](#)). In this case the NF bit will never be set.

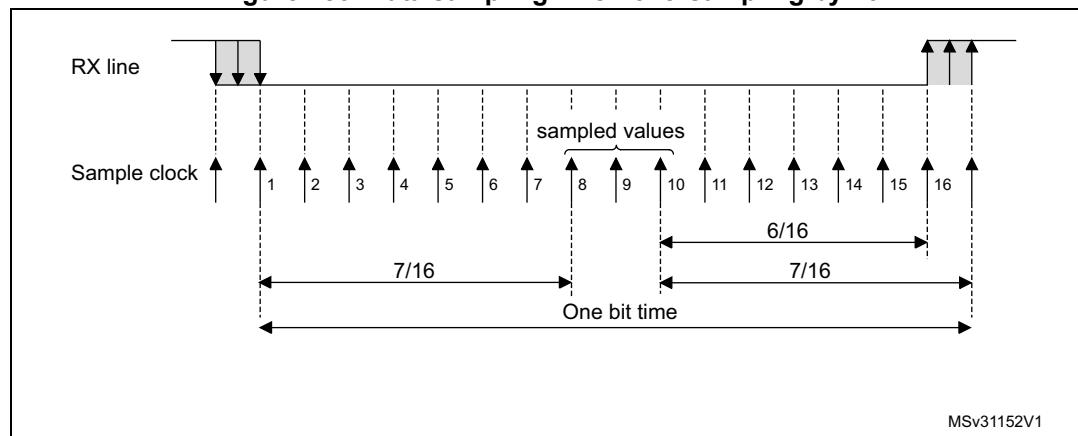
When noise is detected in a frame:

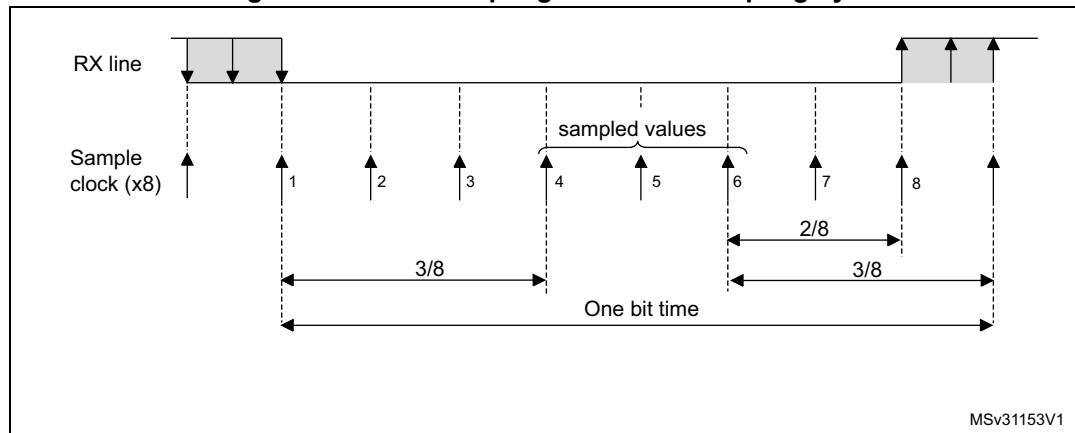
- The NF bit is set at the rising edge of the RXNE bit.
- The invalid data is transferred from the Shift register to the USART\_DR register.
- No interrupt is generated in case of single byte communication. However this bit rises at the same time as the RXNE bit which itself generates an interrupt. In case of multibuffer communication an interrupt will be issued if the EIE bit is set in the USART\_CR3 register.

The NF bit is reset by a USART\_SR register read operation followed by a USART\_DR register read operation.

**Note:** *Oversampling by 8 is not available in the Smartcard, IrDA and LIN modes. In those modes, the OVER8 bit is forced to '0' by hardware.*

**Figure 283. Data sampling when oversampling by 16**



**Figure 284. Data sampling when oversampling by 8****Table 144. 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\_DR register.
- No interrupt is generated in case of single byte communication. However this bit rises at the same time as the RXNE bit which itself generates an interrupt. In case of multibuffer communication an interrupt will be issued if the EIE bit is set in the USART\_CR3 register.

The FE bit is reset by a USART\_SR register read operation followed by a USART\_DR register read operation.

### Configurable stop bits during reception

The number of stop bits to be received can be configured through the control bits of Control Register 2 - it can be either 1 or 2 in normal mode and 0.5 or 1.5 in Smartcard mode.

1. **0.5 stop bit (reception in Smartcard mode):** No sampling is done for 0.5 stop bit. As a consequence, no framing error and no break frame can be detected when 0.5 stop bit is selected.
2. **1 stop bit:** Sampling for 1 stop Bit is done on the 8<sup>th</sup>, 9<sup>th</sup> and 10<sup>th</sup> samples.
3. **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 bit. Sampling for 1.5 stop bits is done on the 16<sup>th</sup>, 17<sup>th</sup> and 18<sup>th</sup> samples (1 baud clock period after the beginning of the stop bit). The 1.5 stop bit can be decomposed into 2 parts: one 0.5 baud clock period during which nothing happens, followed by 1 normal stop bit period during which sampling occurs halfway through. Refer to [Section 25.4.11](#) for more details.
4. **2 stop bits:** Sampling for 2 stop bits is done on the 8<sup>th</sup>, 9<sup>th</sup> and 10<sup>th</sup> 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.

#### 25.4.4 Fractional baud rate generation

The baud rate for the receiver and transmitter (Rx and Tx) are both set to the same value as programmed in the Mantissa and Fraction values of USARTDIV.

##### Equation 1: Baud rate for standard USART (SPI mode included)

$$\text{Tx/Rx baud} = \frac{f_{CK}}{8 \times (2 - \text{OVER8}) \times \text{USARTDIV}}$$

##### Equation 2: Baud rate in Smartcard, LIN and IrDA modes

$$\text{Tx/Rx baud} = \frac{f_{CK}}{16 \times \text{USARTDIV}}$$

USARTDIV is an unsigned fixed point number that is coded on the USART\_BRR register.

- When OVER8=0, the fractional part is coded on 4 bits and programmed by the DIV\_fraction[3:0] bits in the USART\_BRR register
- When OVER8=1, the fractional part is coded on 3 bits and programmed by the DIV\_fraction[2:0] bits in the USART\_BRR register, and bit DIV\_fraction[3] must be kept cleared.

**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.*

##### How to derive USARTDIV from USART\_BRR register values when OVER8=0

###### Example 1:

If DIV\_Mantissa = 0d27 and DIV\_Fraction = 0d12 (USART\_BRR = 0x1BC), then  
Mantissa (USARTDIV) = 0d27

Fraction (USARTDIV) = 12/16 = 0d0.75

Therefore USARTDIV = 0d27.75

**Example 2:**

To program USARTDIV = 0d25.62

This leads to:

DIV\_Fraction = 16\*0d0.62 = 0d9.92

The nearest real number is 0d10 = 0xA

DIV\_Mantissa = mantissa (0d25.620) = 0d25 = 0x19

Then, USART\_BRR = 0x19A hence USARTDIV = 0d25.625

**Example 3:**

To program USARTDIV = 0d50.99

This leads to:

DIV\_Fraction = 16\*0d0.99 = 0d15.84

The nearest real number is 0d16 = 0x10 => overflow of DIV\_frac[3:0] => carry must be added up to the mantissa

DIV\_Mantissa = mantissa (0d50.990 + carry) = 0d51 = 0x33

Then, USART\_BRR = 0x330 hence USARTDIV = 0d51.000

## How to derive USARTDIV from USART\_BRR register values when OVER8=1

**Example 1:**

If DIV\_Mantissa = 0x27 and DIV\_Fraction[2:0]= 0d6 (USART\_BRR = 0x1B6), then

Mantissa (USARTDIV) = 0d27

Fraction (USARTDIV) = 6/8 = 0d0.75

Therefore USARTDIV = 0d27.75

**Example 2:**

To program USARTDIV = 0d25.62

This leads to:

DIV\_Fraction = 8\*0d0.62 = 0d4.96

The nearest real number is 0d5 = 0x5

DIV\_Mantissa = mantissa (0d25.620) = 0d25 = 0x19

Then, USART\_BRR = 0x195 => USARTDIV = 0d25.625

**Example 3:**

To program USARTDIV = 0d50.99

This leads to:

DIV\_Fraction = 8\*0d0.99 = 0d7.92

The nearest real number is  $0d8 = 0x8 \Rightarrow$  overflow of the DIV\_fra[2:0]  $\Rightarrow$  carry must be added up to the mantissa

DIV\_Mantissa = mantissa ( $0d50.990 + \text{carry}$ ) =  $0d51 = 0x33$

Then, USART\_BRR =  $0x0330 \Rightarrow$  USARTDIV =  $0d51.000$

**Table 145. Error calculation for programmed baud rates at  $f_{PCLK} = 8$  MHz or  $f_{PCLK} = 12$  MHz, oversampling by 16<sup>(1)</sup>**

Oversampling by 16 (OVER8=0)							
Baud rate7		$f_{PCLK} = 8$ MHz			$f_{PCLK} = 12$ MHz		
S.No	Desired	Actual	Value programmed in the baud rate register	% Error = (Calculated - Desired) B.rate / Desired B.rate	Actual	Value programmed in the baud rate register	% Error
1	1.2 KBps	1.2 KBps	416.6875	0	1.2 KBps	625	0
2	2.4 KBps	2.4 KBps	208.3125	0.01	2.4 KBps	312.5	0
3	9.6 KBps	9.604 KBps	52.0625	0.04	9.6 KBps	78.125	0
4	19.2 KBps	19.185 KBps	26.0625	0.08	19.2 KBps	39.0625	0
5	38.4 KBps	38.462 KBps	13	0.16	38.339 KBps	19.5625	0.16
6	57.6 KBps	57.554 KBps	8.6875	0.08	57.692 KBps	13	0.16
7	115.2 KBps	115.942 KBps	4.3125	0.64	115.385 KBps	6.5	0.16
8	230.4 KBps	228.571 KBps	2.1875	0.79	230.769 KBps	3.25	0.16
9	460.8 KBps	470.588 KBps	1.0625	2.12	461.538 KBps	1.625	0.16

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 146. Error calculation for programmed baud rates at  $f_{PCLK} = 8$  MHz or  $f_{PCLK} = 12$  MHz, oversampling by 8<sup>(1)</sup>**

Oversampling by 8 (OVER8 = 1)							
Baud rate		$f_{PCLK} = 8$ MHz			$f_{PCLK} = 12$ MHz		
S.No	Desired	Actual	Value programmed in the baud rate register	% Error = (Calculated - Desired) B.rate / Desired B.rate	Actual	Value programmed in the baud rate register	% Error
1	1.2 KBps	1.2 KBps	833.375	0	1.2 KBps	1250	0
2	2.4 KBps	2.4 KBps	416.625	0.01	2.4 KBps	625	0
3	9.6 KBps	9.604 KBps	104.125	0.04	9.6 KBps	156.25	0
4	19.2 KBps	19.185 KBps	52.125	0.08	19.2 KBps	78.125	0
5	38.4 KBps	38.462 KBps	26	0.16	38.339 KBps	39.125	0.16

**Table 146. Error calculation for programmed baud rates at  $f_{PCLK} = 8 \text{ MHz}$  or  $f_{PCLK} = 12 \text{ MHz}$ , oversampling by 8<sup>(1)</sup> (continued)**

Oversampling by 8 (OVER8 = 1)							
Baud rate		$f_{PCLK} = 8 \text{ MHz}$			$f_{PCLK} = 12 \text{ MHz}$		
S.No	Desired	Actual	Value programmed in the baud rate register	% Error = (Calculated - Desired) B.rate / Desired B.rate	Actual	Value programmed in the baud rate register	% Error
6	57.6 KBps	57.554 KBps	17.375	0.08	57.692 KBps	26	0.16
7	115.2 KBps	115.942 KBps	8.625	0.64	115.385 KBps	13	0.16
8	230.4 KBps	228.571 KBps	4.375	0.79	230.769 KBps	6.5	0.16
9	460.8 KBps	470.588 KBps	2.125	2.12	461.538 KBps	3.25	0.16
10	921.6 KBps	888.889 KBps	1.125	3.55	923.077 KBps	1.625	0.16

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 147. Error calculation for programmed baud rates at  $f_{PCLK} = 16 \text{ MHz}$  or  $f_{PCLK} = 24 \text{ MHz}$ , oversampling by 16<sup>(1)</sup>**

Oversampling by 16 (OVER8 = 0)							
Baud rate		$f_{PCLK} = 16 \text{ MHz}$			$f_{PCLK} = 24 \text{ MHz}$		
S.No	Desired	Actual	Value programmed in the baud rate register	% Error = (Calculated - Desired) B.rate / Desired B.rate	Actual	Value programmed in the baud rate register	% Error
1	1.2 KBps	1.2 KBps	833.3125	0	1.2	1250	0
2	2.4 KBps	2.4 KBps	416.6875	0	2.4	625	0
3	9.6 KBps	9.598 KBps	104.1875	0.02	9.6	156.25	0
4	19.2 KBps	19.208 KBps	52.0625	0.04	19.2	78.125	0
5	38.4 KBps	38.369 KBps	26.0625	0.08	38.4	39.0625	0
6	57.6 KBps	57.554 KBps	17.375	0.08	57.554	26.0625	0.08
7	115.2 KBps	115.108 KBps	8.6875	0.08	115.385	13	0.16
8	230.4 KBps	231.884 KBps	4.3125	0.64	230.769	6.5	0.16
9	460.8 KBps	457.143 KBps	2.1875	0.79	461.538	3.25	0.16
10	921.6 KBps	941.176 KBps	1.0625	2.12	923.077	1.625	0.16

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 148. Error calculation for programmed baud rates at  $f_{PCLK} = 16 \text{ MHz}$  or  $f_{PCLK} = 24 \text{ MHz}$ , oversampling by 8<sup>(1)</sup>**

Oversampling by 8 (OVER8=1)							
Baud rate		$f_{PCLK} = 16 \text{ MHz}$			$f_{PCLK} = 24 \text{ MHz}$		
S.No	Desired	Actual	Value programmed in the baud rate register	% Error = (Calculated - Desired) B.rate / Desired B.rate	Actual	Value programmed in the baud rate register	% Error
1	1.2 KBps	1.2 KBps	1666.625	0	1.2 KBps	2500	0
2	2.4 KBps	2.4 KBps	833.375	0	2.4 KBps	1250	0
3	9.6 KBps	9.598 KBps	208.375	0.02	9.6 KBps	312.5	0
4	19.2 KBps	19.208 KBps	104.125	0.04	19.2 KBps	156.25	0
5	38.4 KBps	38.369 KBps	52.125	0.08	38.4 KBps	78.125	0
6	57.6 KBps	57.554 KBps	34.75	0.08	57.554 KBps	52.125	0.08
7	115.2 KBps	115.108 KBps	17.375	0.08	115.385 KBps	26	0.16
8	230.4 KBps	231.884 KBps	8.625	0.64	230.769 KBps	13	0.16
9	460.8 KBps	457.143 KBps	4.375	0.79	461.538 KBps	6.5	0.16
10	921.6 KBps	941.176 KBps	2.125	2.12	923.077 KBps	3.25	0.16
11	2 MBps	2000 KBps	1	0	2000 KBps	1.5	0
12	3 MBps	NA	NA	NA	3000 KBps	1	0

1. The lower the CPU clock the lower the accuracy for a particular baud rate. The upper limit of the achievable baud rate can be fixed with these data.

**Table 149. Error calculation for programmed baud rates at  $f_{PCLK} = 8 \text{ MHz}$  or  $f_{PCLK} = 16 \text{ MHz}$ , oversampling by 16<sup>(1)</sup>**

Oversampling by 16 (OVER8=0)							
Baud rate		$f_{PCLK} = 8 \text{ MHz}$			$f_{PCLK} = 16 \text{ MHz}$		
S.No	Desired	Actual	Value programmed in the baud rate register	% Error = (Calculated - Desired) B.Rate / Desired B.Rate	Actual	Value programmed in the baud rate register	% Error
1.	2.4 KBps	2.400 KBps	208.3125	0.00%	2.400 KBps	416.6875	0.00%
2.	9.6 KBps	9.604 KBps	52.0625	0.04%	9.598 KBps	104.1875	0.02%
3.	19.2 KBps	19.185 KBps	26.0625	0.08%	19.208 KBps	52.0625	0.04%
4.	57.6 KBps	57.554 KBps	8.6875	0.08%	57.554 KBps	17.3750	0.08%
5.	115.2 KBps	115.942 KBps	4.3125	0.64%	115.108 KBps	8.6875	0.08%
6.	230.4 KBps	228.571 KBps	2.1875	0.79%	231.884 KBps	4.3125	0.64%
7.	460.8 KBps	470.588 KBps	1.0625	2.12%	457.143 KBps	2.1875	0.79%

**Table 149. Error calculation for programmed baud rates at  $f_{PCLK} = 8 \text{ MHz}$  or  $f_{PCLK} = 16 \text{ MHz}$ , oversampling by 16<sup>(1)</sup> (continued)**

Oversampling by 16 (OVER8=0)							
Baud rate		$f_{PCLK} = 8 \text{ MHz}$			$f_{PCLK} = 16 \text{ MHz}$		
S.No	Desired	Actual	Value programmed in the baud rate register	% Error = (Calculated - Desired)B.Rate /Desired B.Rate	Actual	Value programmed in the baud rate register	% Error
8.	896 KBps	NA	NA	NA	888.889 KBps	1.1250	0.79%
9.	921.6 KBps	NA	NA	NA	941.176 KBps	1.0625	2.12%

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 150. Error calculation for programmed baud rates at  $f_{PCLK} = 8 \text{ MHz}$  or  $f_{PCLK} = 16 \text{ MHz}$ , oversampling by 8<sup>(1)</sup>**

Oversampling by 8 (OVER8=1)							
Baud rate		$f_{PCLK} = 8 \text{ MHz}$			$f_{PCLK} = 16 \text{ MHz}$		
S.No	Desired	Actual	Value programmed in the baud rate register	% Error = (Calculated - Desired)B.Rate /Desired B.Rate	Actual	Value programmed in the baud rate register	% Error
1.	2.4 KBps	2.400 KBps	416.625	0.01%	2.400 KBps	833.375	0.00%
2.	9.6 KBps	9.604 KBps	104.125	0.04%	9.598 KBps	208.375	0.02%
3.	19.2 KBps	19.185 KBps	52.125	0.08%	19.208 KBps	104.125	0.04%
4.	57.6 KBps	57.557 KBps	17.375	0.08%	57.554 KBps	34.750	0.08%
5.	115.2 KBps	115.942 KBps	8.625	0.64%	115.108 KBps	17.375	0.08%
6.	230.4 KBps	228.571 KBps	4.375	0.79%	231.884 KBps	8.625	0.64%
7.	460.8 KBps	470.588 KBps	2.125	2.12%	457.143 KBps	4.375	0.79%
8.	896 KBps	888.889 KBps	1.125	0.79%	888.889 KBps	2.250	0.79%
9.	921.6 KBps	888.889 KBps	1.125	3.55%	941.176 KBps	2.125	2.12%
10.	1.792 MBps	NA	NA	NA	1.7777 MBps	1.125	0.79%
11.	1.8432 MBps	NA	NA	NA	1.7777 MBps	1.125	3.55%

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 151. Error calculation for programmed baud rates at  $f_{PCLK} = 30 \text{ MHz}$  or  $f_{PCLK} = 60 \text{ MHz}$ , oversampling by 16<sup>(1)(2)</sup>**

Oversampling by 16 (OVER8=0)							
Baud rate		$f_{PCLK} = 30 \text{ MHz}$			$f_{PCLK} = 60 \text{ MHz}$		
S.No	Desired	Actual	Value programmed in the baud rate register	% Error = (Calculated - Desired)B.Rate /Desired B.Rate	Actual	Value programmed in the baud rate register	% Error
1.	2.4 KBps	2.400 KBps	781.2500	0.00%	2.400 KBps	1562.5000	0.00%
2.	9.6 KBps	9.600 KBps	195.3125	0.00%	9.600 KBps	390.6250	0.00%
3.	19.2 KBps	19.194 KBps	97.6875	0.03%	19.200 KBps	195.3125	0.00%
4.	57.6 KBps	57.582 KBps	32.5625	0.03%	57.582 KBps	65.1250	0.03%
5.	115.2 KBps	115.385 KBps	16.2500	0.16%	115.163 KBps	32.5625	0.03%
6.	230.4 KBps	230.769 KBps	8.1250	0.16%	230.769 KBps	16.2500	0.16%
7.	460.8 KBps	461.538 KBps	4.0625	0.16%	461.538 KBps	8.1250	0.16%
8.	896 KBps	909.091 KBps	2.0625	1.46%	895.522 KBps	4.1875	0.05%
9.	921.6 KBps	909.091 KBps	2.0625	1.36%	923.077 KBps	4.0625	0.16%
10.	1.792 MBps	1.1764 MBps	1.0625	1.52%	1.8182 MBps	2.0625	1.36%
11.	1.8432 MBps	1.8750 MBps	1.0000	1.73%	1.8182 MBps	2.0625	1.52%
12.	3.584 MBps	NA	NA	NA	3.2594 MBps	1.0625	1.52%
13.	3.6864 MBps	NA	NA	NA	3.7500 MBps	1.0000	1.73%

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.
2. Only USART1 and USART6 are clocked with PCLK2. Other USARTs are clocked with PCLK1. Refer to the device datasheets for the maximum values for PCLK1 and PCLK2.

**Table 152. Error calculation for programmed baud rates at  $f_{PCLK} = 30 \text{ MHz}$  or  $f_{PCLK} = 60 \text{ MHz}$ , oversampling by 8<sup>(1)(2)</sup>**

Oversampling by 8 (OVER8=1)							
Baud rate		$f_{PCLK} = 30 \text{ MHz}$			$f_{PCLK} = 60 \text{ MHz}$		
S.No	Desired	Actual	Value programmed in the baud rate register	% Error = (Calculated - Desired)B.Rate /Desired B.Rate	Actual	Value programmed in the baud rate register	% Error
1.	2.4 KBps	2.400 KBps	1562.5000	0.00%	2.400 KBps	3125.0000	0.00%
2.	9.6 KBps	9.600 KBps	390.6250	0.00%	9.600 KBps	781.2500	0.00%
3.	19.2 KBps	19.194 KBps	195.3750	0.03%	19.200 KBps	390.6250	0.00%
4.	57.6 KBps	57.582 KBps	65.1250	0.16%	57.582 KBps	130.2500	0.03%
5.	115.2 KBps	115.385 KBps	32.5000	0.16%	115.163 KBps	65.1250	0.03%
6.	230.4 KBps	230.769 KBps	16.2500	0.16%	230.769 KBps	32.5000	0.16%

**Table 152. Error calculation for programmed baud rates at  $f_{PCLK} = 30$  MHz or  $f_{PCLK} = 60$  MHz, oversampling by 8<sup>(1)(2)</sup> (continued)**

Oversampling by 8 (OVER8=1)							
Baud rate		$f_{PCLK} = 30$ MHz			$f_{PCLK} = 60$ MHz		
S.No	Desired	Actual	Value programmed in the baud rate register	% Error = (Calculated - Desired)B.Rate /Desired B.Rate	Actual	Value programmed in the baud rate register	% Error
7.	460.8 KBps	461.538 KBps	8.1250	0.16%	461.538 KBps	16.2500	0.16%
8.	896 KBps	909.091 KBps	4.1250	1.46%	895.522 KBps	8.3750	0.05%
9.	921.6 KBps	909.091 KBps	4.1250	1.36%	923.077 KBps	8.1250	0.16%
10.	1.792 MBps	1.7647 MBps	2.1250	1.52%	1.8182 MBps	4.1250	1.46%
11.	1.8432 MBps	1.8750 MBps	2.0000	1.73%	1.8182 MBps	4.1250	1.36%
12.	3.584 MBps	3.7500 MBps	1.0000	4.63%	3.5294 MBps	2.1250	1.52%
13.	3.6864 MBps	3.7500 MBps	1.0000	1.73%	3.7500 MBps	2.0000	1.73%
14.	7.168 MBps	NA	NA	NA	7.5000 MBps	1.0000	4.63%
15.	7.3728 MBps	NA	NA	NA	7.5000 MBps	1.0000	1.73%

- 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.
- Only USART1 and USART6 are clocked with PCLK2. Other USARTs are clocked with PCLK1. Refer to the device datasheets for the maximum values for PCLK1 and PCLK2.

**Table 153. Error calculation for programmed baud rates at  $f_{PCLK} = 42$  MHz or  $f_{PCLK} = 84$  Hz, oversampling by 16<sup>(1)(2)</sup>**

Oversampling by 16 (OVER8=0)							
Baud rate		$f_{PCLK} = 42$ MHz			$f_{PCLK} = 84$ MHz		
S.No	Desired	Actual	Value programmed in the baud rate register	% Error = (Calculated - Desired)B.Rate /Desired B.Rate	Actual	Value programmed in the baud rate register	% Error
1	1.2 KBps	1.2 KBps	2187.5	0	1.2 KBps	NA	0
2	2.4 KBps	2.4 KBps	1093.75	0	2.4 KBps	2187.5	0
3	9.6 KBps	9.6 KBps	273.4375	0	9.6 KBps	546.875	0
4	19.2 KBps	19.195 KBps	136.75	0.02	19.2 KBps	273.4375	0
5	38.4 KBps	38.391 KBps	68.375	0.02	38.391 KBps	136.75	0.02
6	57.6 KBps	57.613 KBps	45.5625	0.02	57.613 KBps	91.125	0.02
7	115.2 KBps	115.068 KBps	22.8125	0.11	115.226 KBps	45.5625	0.02
8	230.4 KBps	230.769 KBps	11.375	0.16	230.137 KBps	22.8125	0.11
9	460.8 KBps	461.538 KBps	5.6875	0.16	461.538 KBps	11.375	0.16

**Table 153. Error calculation for programmed baud rates at  $f_{PCLK} = 42$  MHz or  $f_{PCLK} = 84$  Hz, oversampling by 16<sup>(1)(2)</sup> (continued)**

Oversampling by 16 (OVER8=0)							
Baud rate		$f_{PCLK} = 42$ MHz			$f_{PCLK} = 84$ MHz		
S.No	Desired	Actual	Value programmed in the baud rate register	% Error = (Calculated - Desired)B.Rate /Desired B.Rate	Actual	Value programmed in the baud rate register	% Error
10	921.6 KBps	913.043 KBps	2.875	0.93	923.076 KBps	5.6875	0.93
11	1.792 MBps	1.826 MBps	1.4375	1.9	1.787 MBps	2.9375	0.27
12	1.8432 MBps	1.826 MBps	1.4375	0.93	1.826 MBps	2.875	0.93
13	3.584 MBps	NA	NA	NA	3.652 MBps	1.4375	1.9
14	3.6864 MBps	NA	NA	NA	3.652 MBps	1.4375	0.93

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.

2. Only USART1 and USART6 are clocked with PCLK2. Other USARTs are clocked with PCLK1. Refer to the device datasheets for the maximum values for PCLK1 and PCLK2.

**Table 154. Error calculation for programmed baud rates at  $f_{PCLK} = 42$  MHz or  $f_{PCLK} = 84$  MHz, oversampling by 8<sup>(1)(2)</sup>**

Oversampling by 8 (OVER8=1)							
Baud rate		$f_{PCLK} = 42$ MHz			$f_{PCLK} = 84$ MHz		
S.No	Desired	Actual	Value programmed in the baud rate register	% Error = (Calculated - Desired)B.Rate /Desired B.Rate	Actual	Value programmed in the baud rate register	% Error
1.	1.2 KBps	1.2 KBps	NA	0	1.2 KBps	NA	0
2.	2.4 KBps	2.4 KBps	2187.5	0	2.4 KBps	NA	0
3.	9.6 KBps	9.6 KBps	546.875	0	9.6 KBps	1093.75	0
4.	19.2 KBps	19.195 KBps	273.5	0.02	19.2 KBps	546.875	0
5.	38.4 KBps	38.391 KBps	136.75	0.02	38.391 KBps	273.5	0.02
6.	57.6 KBps	57.613 KBps	91.125	0.02	57.613 KBps	182.25	0.02
7.	115.2 KBps	115.068 KBps	45.625	0.11	115.226 KBps	91.125	0.02
8.	230.4 KBps	230.769 KBps	22.75	0.11	230.137 KBps	45.625	0.11
9.	460.8 KBps	461.538 KBps	11.375	0.16	461.538 KBps	22.75	0.16
10.	921.6 KBps	913.043 KBps	5.75	0.93	923.076 KBps	11.375	0.93
11.	1.792 MBps	1.826 MBps	2.875	1.9	1.787 MBps	5.875	0.27
12.	1.8432 MBps	1.826 MBps	2.875	0.93	1.826 MBps	5.75	0.93
13.	3.584 MBps	3.5 MBps	1.5	2.34	3.652 MBps	2.875	1.9
14.	3.6864 MBps	3.82 MBps	1.375	3.57	3.652 MBps	2.875	0.93

**Table 154. Error calculation for programmed baud rates at  $f_{PCLK} = 42$  MHz or  $f_{PCLK} = 84$  MHz, oversampling by 8<sup>(1)(2)</sup> (continued)**

Oversampling by 8 (OVER8=1)							
Baud rate		$f_{PCLK} = 42$ MHz			$f_{PCLK} = 84$ MHz		
S.No	Desired	Actual	Value programmed in the baud rate register	% Error = (Calculated - Desired)B.Rate /Desired B.Rate	Actual	Value programmed in the baud rate register	% Error
15.	7.168 MBps	NA	NA	NA	7 MBps	1.5	2.34
16.	7.3728 MBps	NA	NA	NA	7.636 MBps	1.375	3.57
18.	9 MBps	NA	NA	NA	9.333 MBps	1.125	3.7
20.	10.5 MBps	NA	NA	NA	10.5 MBps	1	0

1. The lower the CPU clock the lower the accuracy for a particular baud rate. The upper limit of the achievable baud rate can be fixed with these data.
2. Only USART1 and USART6 are clocked with PCLK2. Other USARTs are clocked with PCLK1. Refer to the device datasheets for the maximum values for PCLK1 and PCLK2.

#### 25.4.5 USART receiver tolerance to clock deviation

The USART asynchronous receiver works correctly only if the total clock system deviation is smaller than the USART receiver tolerance. The causes which contribute to the total deviation are:

- DTRA: Deviation due to the transmitter error (which also includes the deviation of the transmitter local oscillator)
- DQUANT: Error due to the baud rate quantization of the receiver
- DREC: Deviation of the receiver 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)

$$\text{DTRA} + \text{DQUANT} + \text{DREC} + \text{DTCL} < \text{USART receiver tolerance}$$

The USART receiver tolerance to properly receive data is equal to the maximum tolerated deviation and depends on the following choices:

- 10- or 11-bit character length defined by the M bit in the USART\_CR1 register
- oversampling by 8 or 16 defined by the OVER8 bit in the USART\_CR1 register
- use of fractional baud rate or not
- 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 155. USART receiver tolerance when DIV fraction is 0**

M bit	OVER8 bit = 0		OVER8 bit = 1	
	ONEBIT=0	ONEBIT=1	ONEBIT=0	ONEBIT=1
0	3.75%	4.375%	2.50%	3.75%
1	3.41%	3.97%	2.27%	3.41%

**Table 156. USART receiver tolerance when DIV\_Fraction is different from 0**

M bit	OVER8 bit = 0		OVER8 bit = 1	
	ONEBIT=0	ONEBIT=1	ONEBIT=0	ONEBIT=1
0	3.33%	3.88%	2%	3%
1	3.03%	3.53%	1.82%	2.73%

**Note:**

The figures specified in [Table 155](#) and [Table 156](#) may slightly differ in the special case when the received frames contain some Idle frames of exactly 10-bit times when M=0 (11-bit times when M=1).

## 25.4.6 Multiprocessor communication

There is a possibility of performing multiprocessor communication with the USART (several USARTs connected in a network). For instance one of the USARTs can be the master, its TX output is connected to the RX input of the other USART. The others are slaves, their respective TX outputs are logically ANDed together and connected to the RX input of the master.

In multiprocessor configurations it is often desirable that only the intended message recipient should actively receive the full message contents, thus reducing redundant USART service overhead for all non addressed receivers.

The non addressed devices may be placed in mute mode by means of the muting function. In mute mode:

- None of the reception status bits can be set.
- All the receive interrupts are inhibited.
- The RWU bit in USART\_CR1 register is set to 1. RWU can be controlled automatically by hardware or written by the software under certain conditions.

The USART can enter or exit from mute mode using one of two methods, depending on the WAKE bit in the USART\_CR1 register:

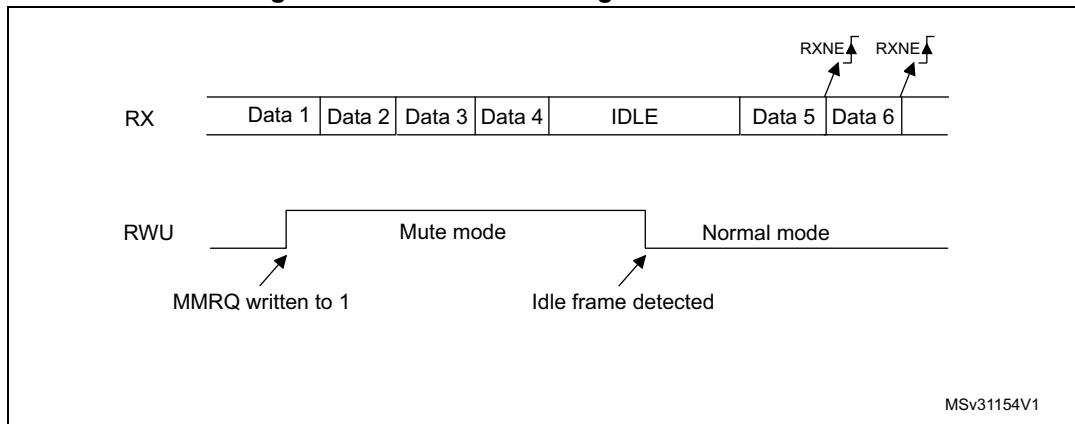
- Idle Line detection if the WAKE bit is reset,
- Address Mark detection if the WAKE bit is set.

### Idle line detection (WAKE=0)

The USART enters mute mode when the RWU bit is written to 1.

It wakes up when an Idle frame is detected. Then the RWU bit is cleared by hardware but the IDLE bit is not set in the USART\_SR register. RWU can also be written to 0 by software.

An example of mute mode behavior using Idle line detection is given in [Figure 285](#).

**Figure 285. Mute mode using Idle line detection**

MSv31154V1

**Address mark detection (WAKE=1)**

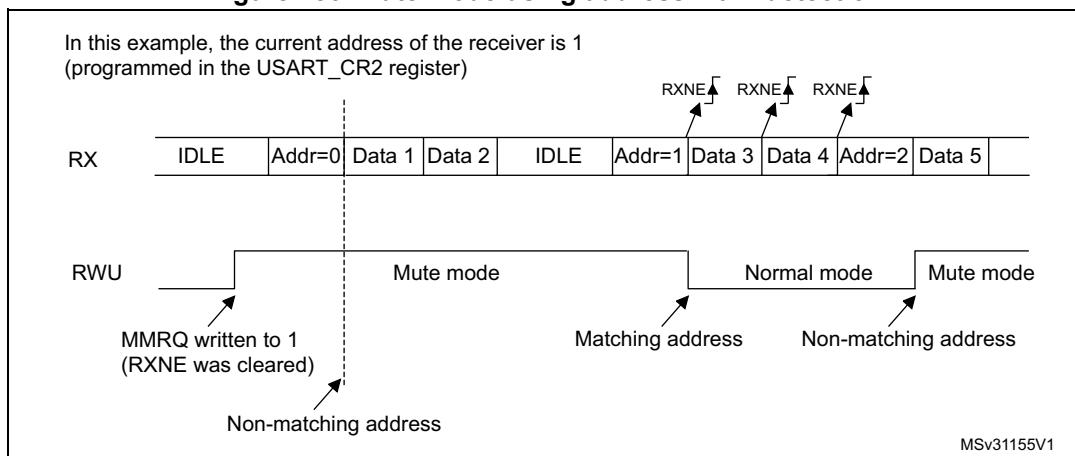
In this mode, bytes are recognized as addresses if their MSB is a '1' else they are considered as data. In an address byte, the address of the targeted receiver is put on the 4 LSB. This 4-bit word is compared by the receiver with its own address which is programmed in the ADD bits in the USART\_CR2 register.

The USART enters mute mode when an address character is received which does not match its programmed address. In this case, the RWU bit is set by hardware. The RXNE flag is not set for this address byte and no interrupt nor DMA request is issued as the USART would have entered mute mode.

It exits from mute mode when an address character is received which matches the programmed address. Then the RWU bit is cleared and subsequent bytes are received normally. The RXNE bit is set for the address character since the RWU bit has been cleared.

The RWU bit can be written to as 0 or 1 when the receiver buffer contains no data (RXNE=0 in the USART\_SR register). Otherwise the write attempt is ignored.

An example of mute mode behavior using address mark detection is given in [Figure 286](#).

**Figure 286. Mute mode using address mark detection**

MSv31155V1

### 25.4.7 Parity control

Parity control (generation of parity bit in transmission and parity checking in reception) can be enabled by setting the PCE bit in the USART\_CR1 register. Depending on the frame length defined by the M bit, the possible USART frame formats are as listed in [Table 157](#).

**Table 157. Frame formats**

M bit	PCE bit	USART frame <sup>(1)</sup>
0	0	SB   8 bit data   STB
0	1	SB   7-bit data   PB   STB
1	0	SB   9-bit data   STB
1	1	SB   8-bit data PB   STB

1. Legends: SB: start bit, STB: stop bit, PB: parity bit.

#### Even parity

The parity bit is calculated to obtain an even number of “1s” inside the frame made of the 7 or 8 LSB bits (depending on whether M is equal to 0 or 1) and the parity bit.

E.g.: data=00110101; 4 bits set => parity bit will be 0 if even parity is selected (PS bit in USART\_CR1 = 0).

#### Odd parity

The parity bit is calculated to obtain an odd number of “1s” inside the frame made of the 7 or 8 LSB bits (depending on whether M is equal to 0 or 1) and the parity bit.

E.g.: data=00110101; 4 bits set => 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\_SR register and an interrupt is generated if PEIE is set in the USART\_CR1 register. The PE flag is cleared by a software sequence (a read from the status register followed by a read or write access to the USART\_DR data register).

**Note:** *In case of wakeup by an address mark: the MSB bit of the data is taken into account to identify an address but not the parity bit. And the receiver does not check the parity of the address data (PE is not set in case of a parity error).*

#### 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)).

**Note:** *The software routine that manages the transmission can activate the software sequence which clears the PE flag (a read from the status register followed by a read or write access to the data register). When operating in half-duplex mode, depending on the software, this can cause the PE flag to be unexpectedly cleared.*

### 25.4.8 LIN (local interconnection network) mode

The LIN mode is selected by setting the LINEN bit in the USART\_CR2 register. In LIN mode, the following bits must be kept cleared:

- STOP[1:0] and CLKEN in the USART\_CR2 register
- SCEN, HDSEL and IREN in the USART\_CR3 register.

#### LIN transmission

The same procedure explained in [Section 25.4.2](#) has to be applied for LIN Master transmission than for normal USART transmission with the following differences:

- Clear the M bit to configure 8-bit word length.
- Set the LINEN bit to enter LIN mode. In this case, setting the SBK bit sends 13 '0 bits as a break character. Then a bit of value '1 is sent to allow the next start detection.

#### LIN reception

A break detection circuit is implemented on the USART interface. The detection is totally independent from the normal USART receiver. A break can be detected whenever it occurs, during Idle state or during a frame.

When the receiver is enabled (RE=1 in USART\_CR1), the circuit looks at the RX input for a start signal. The method for detecting start bits is the same when searching break characters or data. After a start bit has been detected, the circuit samples the next bits exactly like for the data (on the 8th, 9th and 10th samples). If 10 (when the LBDL = 0 in USART\_CR2) or 11 (when LBDL=1 in USART\_CR2) consecutive bits are detected as '0, and are followed by a delimiter character, the LBD flag is set in USART\_SR. If the LBDIE bit=1, an interrupt is generated. Before validating the break, the delimiter is checked for as it signifies that the RX line has returned to a high level.

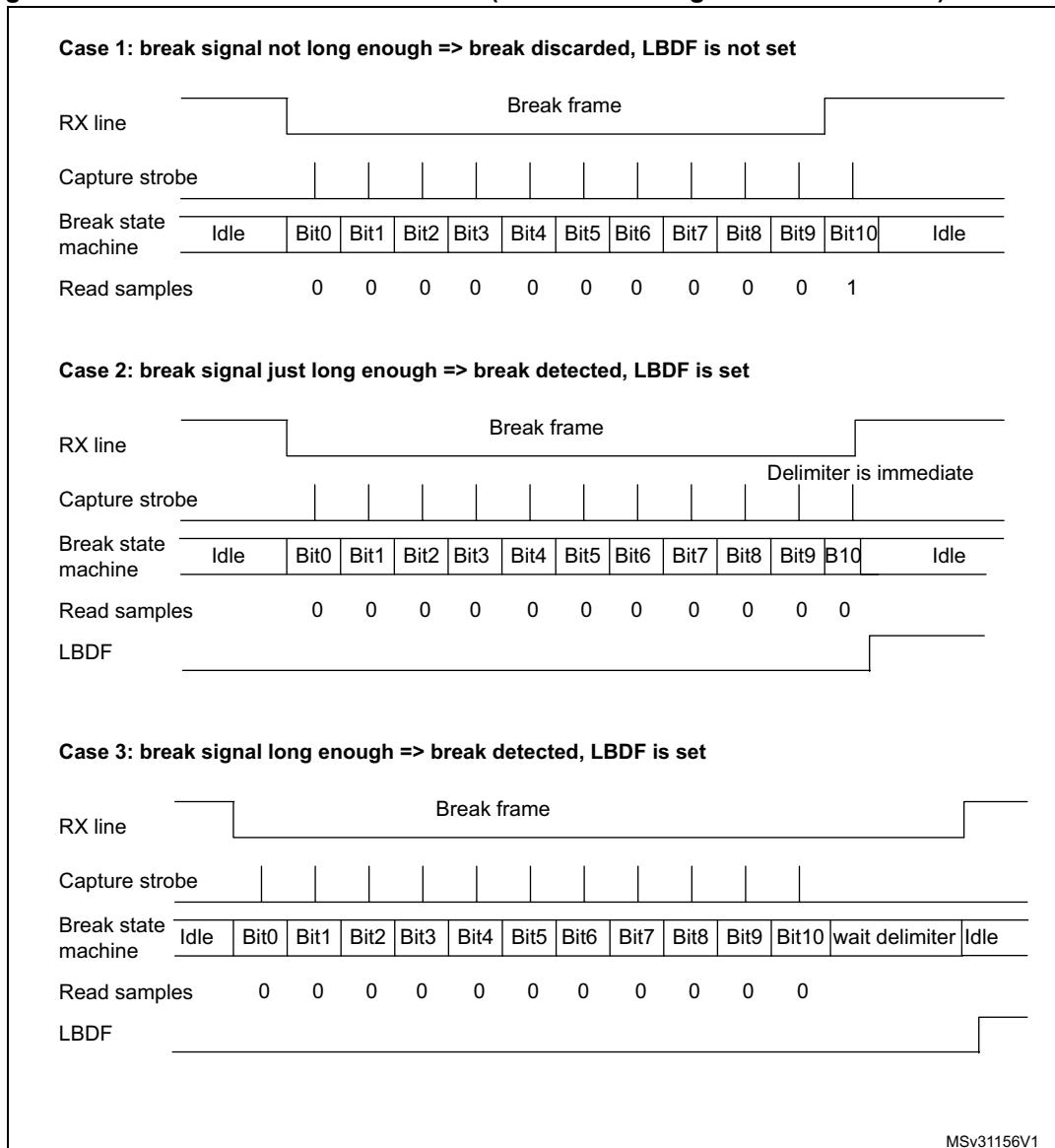
If a '1 is sampled before the 10 or 11 have occurred, the break detection circuit cancels the current detection and searches for a start bit again.

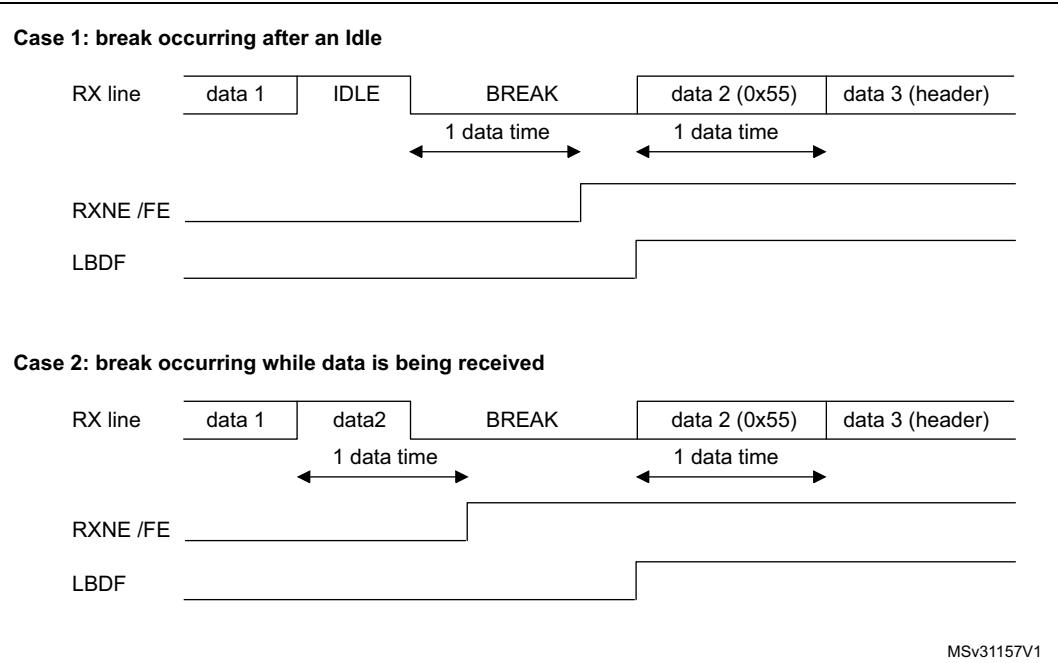
If the LIN mode is disabled (LINEN=0), the receiver continues working as normal USART, without taking into account the break detection.

If the LIN mode is enabled (LINEN=1), as soon as a framing error occurs (i.e. stop bit detected at '0, which will be the case for any break frame), the receiver stops until the break detection circuit receives either a '1, if the break word was not complete, or a delimiter character if a break has been detected.

The behavior of the break detector state machine and the break flag is shown in [Figure 287](#).

Examples of break frames are given on [Figure 288](#), where we suppose that LBDL=1 (11-bit break length), and M=0 (8-bit data).

**Figure 287. Break detection in LIN mode (11-bit break length - LBDL bit is set)**

**Figure 288. Break detection in LIN mode vs. Framing error detection**

### 25.4.9 USART synchronous mode

The synchronous mode is selected by writing the CLKEN bit in the USART\_CR2 register to 1. In synchronous mode, the following bits must be kept cleared:

- LINEN bit in the USART\_CR2 register,
- SCEN, HDSEL and IREN bits in the USART\_CR3 register.

The USART allows the user to control a bidirectional synchronous serial communications in master mode. The SCLK pin is the output of the USART transmitter clock. No clock pulses are sent to the SCLK pin during start bit and stop bit. Depending on the state of the LBCL bit in the USART\_CR2 register clock pulses will or will not be generated during the last valid data bit (address mark). The CPOL bit in the USART\_CR2 register allows the user to select the clock polarity, and the CPHA bit in the USART\_CR2 register allows the user to select the phase of the external clock (see [Figure 289](#), [Figure 290](#) and [Figure 291](#)).

During the Idle state, preamble and send break, the external SCLK clock is not activated.

In synchronous mode the USART transmitter works exactly like in asynchronous mode. But as SCLK is synchronized with TX (according to CPOL and CPHA), the data on TX is synchronous.

In this mode the USART receiver works in a different manner compared to the asynchronous mode. If RE=1, the data is sampled on SCLK (rising or falling edge, depending on CPOL and CPHA), without any oversampling. A setup and a hold time must be respected (which depends on the baud rate: 1/16 bit time).

**Note:** *The SCLK pin works in conjunction with the TX pin. Thus, the clock is provided only if the transmitter is enabled (TE=1) and a data is being transmitted (the data register USART\_DR*

*(has been written). This means that it is not possible to receive a synchronous data without transmitting data.*

*The LBCL, CPOL and CPHA bits have to be selected when both the transmitter and the receiver are disabled ( $TE=RE=0$ ) to ensure that the clock pulses function correctly. These bits should not be changed while the transmitter or the receiver is enabled.*

*It is advised that TE and RE are set in the same instruction in order to minimize the setup and the hold time of the receiver.*

*The USART supports master mode only: it cannot receive or send data related to an input clock (SCLK is always an output).*

Figure 289. USART example of synchronous transmission

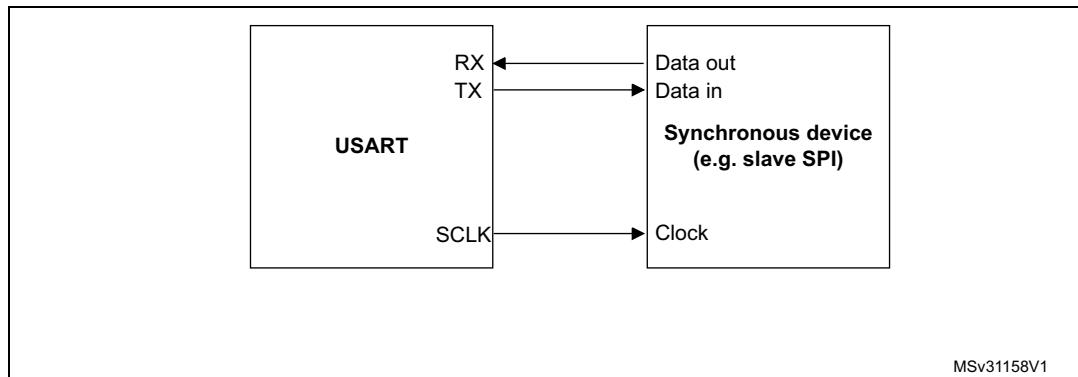


Figure 290. USART data clock timing diagram (M=0)

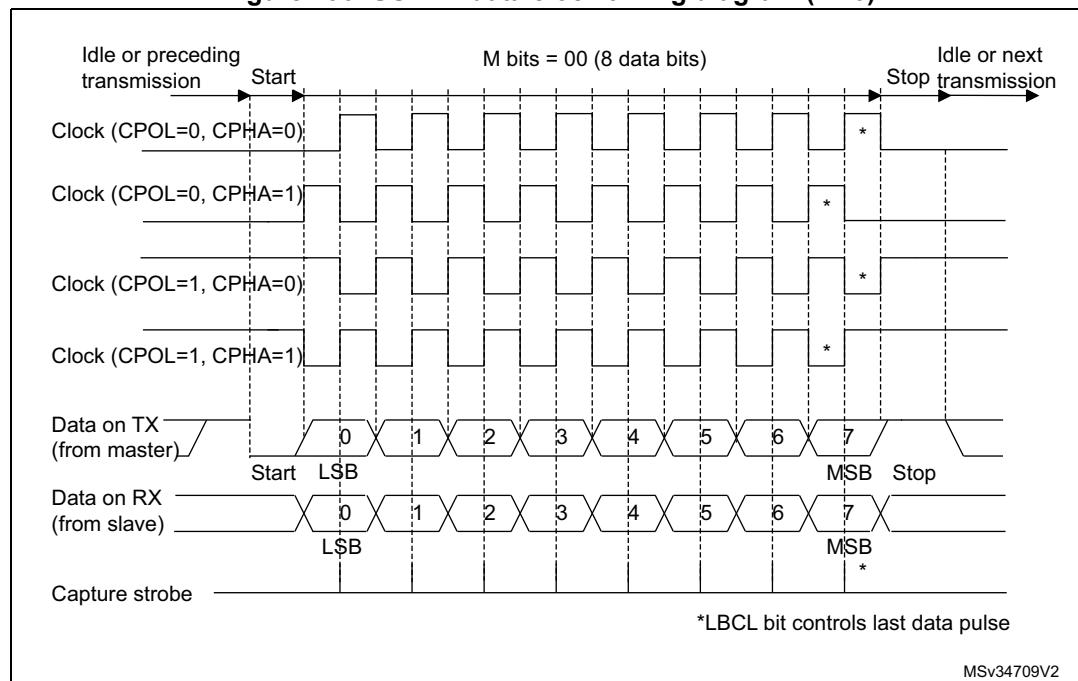


Figure 291. USART data clock timing diagram (M=1)

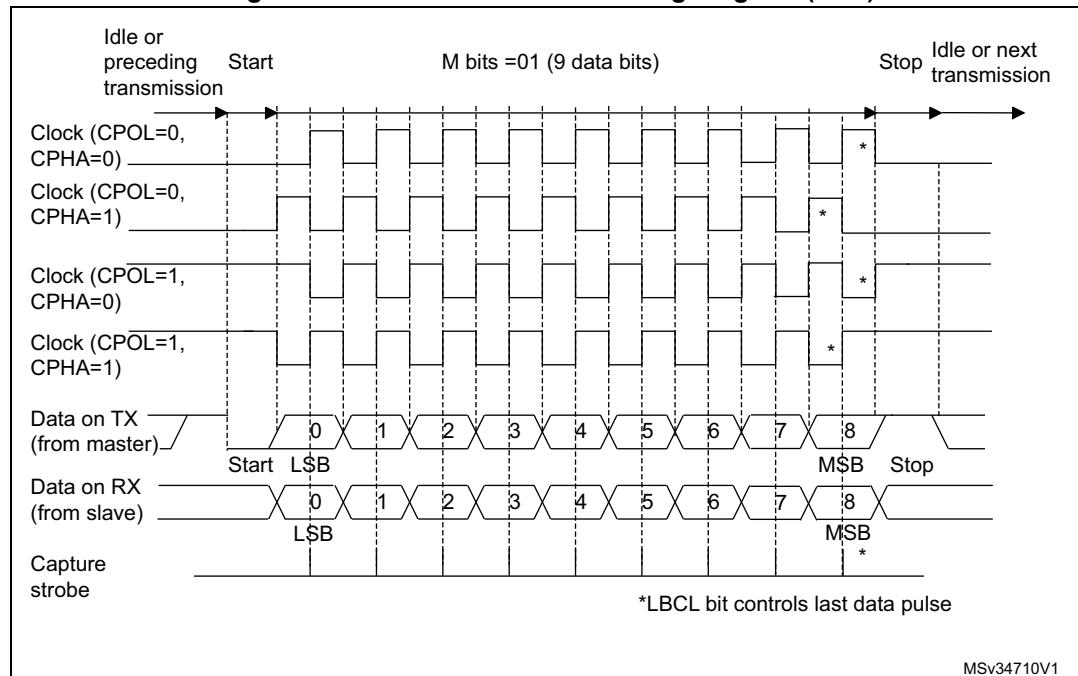
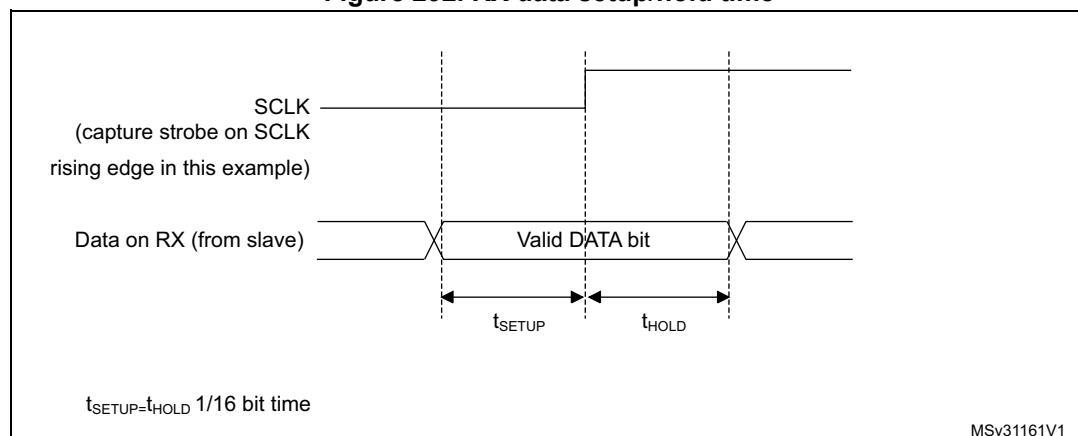


Figure 292. RX data setup/hold time

**Note:**

The function of SCLK is different in Smartcard mode. Refer to the Smartcard mode chapter for more details.

#### 25.4.10 Single-wire half-duplex communication

The single-wire half-duplex mode is selected by setting the HDSEL bit in the USART\_CR3 register. In this mode, the following bits must be kept cleared:

- LINEN and CLKEN bits in the USART\_CR2 register,
- SCEN and IREN bits in the USART\_CR3 register.

The USART can be configured to follow a single-wire half-duplex protocol where the TX and RX lines are internally connected. The selection between half- and full-duplex communication is made with a control bit 'HALF DUPLEX SEL' (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 floating input (or output high open-drain) when not driven by the USART.

Apart from this, the communications are similar to what is done in normal USART mode. The conflicts on the line must be managed by the software (by the use of a centralized arbiter, for instance). In particular, the transmission is never blocked by hardware and continue to occur as soon as a data is written in the data register while the TE bit is set.

#### 25.4.11 Smartcard

The Smartcard mode is selected by setting the SCEN bit in the USART\_CR3 register. In smartcard mode, the following bits must be kept cleared:

- LINEN bit in the USART\_CR2 register,
- HDSEL and IREN bits in the USART\_CR3 register.

Moreover, the CLKEN bit may be set in order to provide a clock to the smartcard.

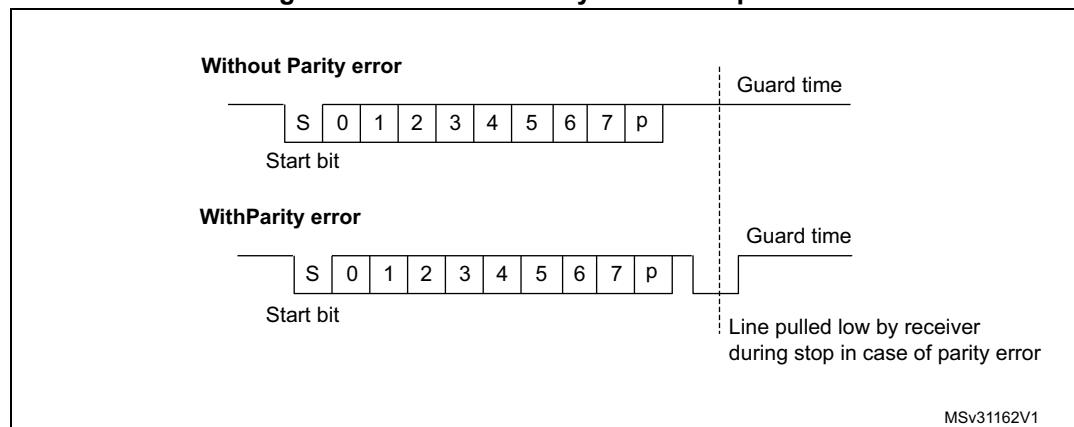
The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard. The USART should be configured as:

- 8 bits plus parity: where M=1 and PCE=1 in the USART\_CR1 register
- 1.5 stop bits when transmitting and receiving: where STOP=11 in the USART\_CR2 register.

*Note:* *It is also possible to choose 0.5 stop bit for receiving but it is recommended to use 1.5 stop bits for both transmitting and receiving to avoid switching between the two configurations.*

*Figure 293* shows examples of what can be seen on the data line with and without parity error.

**Figure 293. 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 is a single wire half duplex communication protocol.

- Transmission of data from the transmit shift register is guaranteed to be delayed by a minimum of 1/2 baud clock. In normal operation a full transmit shift register will start

shifting on the next baud clock edge. In Smartcard mode this transmission is further delayed by a guaranteed 1/2 baud clock.

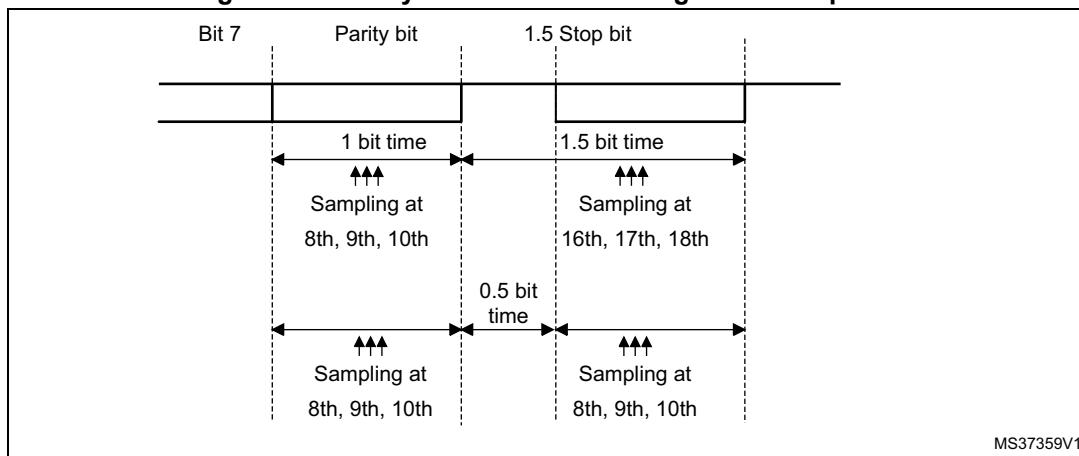
- If a parity error is detected during reception of a frame programmed with a 0.5 or 1.5 stop bit period, the transmit line is pulled low for a baud clock period after the completion of the receive frame. This is to indicate to the Smartcard that the data transmitted to USART has not been correctly received. This NACK signal (pulling transmit line low for 1 baud clock) will cause a framing error on the transmitter side (configured with 1.5 stop bits). The application can handle re-sending of data according to the protocol. A parity error is ‘NACK’ed by the receiver if the NACK control bit is set, otherwise a NACK is not transmitted.
- The assertion of the TC flag can be delayed by programming the Guard Time register. In normal operation, TC is asserted when the transmit shift register is empty and no further transmit requests are outstanding. In Smartcard mode an empty transmit shift register triggers the guard time counter to count up to the programmed value in the Guard Time register. TC is forced low during this time. When the guard time counter reaches the programmed value TC is asserted high.
- The de-assertion of TC flag is unaffected by Smartcard mode.
- If a framing error is detected on the transmitter end (due to a NACK from the receiver), the NACK will not be detected as a start bit by the receive block of the transmitter. According to the ISO protocol, the duration of the received NACK can be 1 or 2 baud clock periods.
- On the receiver side, if a parity error is detected and a NACK is transmitted the receiver will not detect the NACK as a start bit.

**Note:** A break character is not significant in Smartcard mode. A 0x00 data with a framing error will be treated as data and not as a break.

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 294 details how the NACK signal is sampled by the USART. In this example the USART is transmitting a data and is configured with 1.5 stop bits. The receiver part of the USART is enabled in order to check the integrity of the data and the NACK signal.

**Figure 294. Parity error detection using the 1.5 stop bits**



MS37359V1

The USART can provide a clock to the smartcard through the SCLK output. In smartcard mode, SCLK is not associated to the communication but is simply derived from the internal peripheral input clock through a 5-bit prescaler. The division ratio is configured in the

prescaler register USART\_GTPR. SCLK frequency can be programmed from  $f_{CK}/2$  to  $f_{CK}/62$ , where  $f_{CK}$  is the peripheral input clock.

### 25.4.12 IrDA SIR ENDEC block

The IrDA mode is selected by setting the IREN bit in the USART\_CR3 register. In IrDA mode, the following bits must be kept cleared:

- LINEN, STOP and CLKEN bits in the USART\_CR2 register,
- SCEN and HDSEL bits in the USART\_CR3 register.

The IrDA SIR physical layer specifies use of a Return to Zero, Inverted (RZI) modulation scheme that represents logic 0 as an infrared light pulse (see [Figure 295](#)).

The SIR Transmit encoder modulates the Non Return to Zero (NRZ) transmit bit stream output from USART. The output pulse stream is transmitted to an external output driver and infrared LED. USART supports only bit rates up to 115.2Kbps for the SIR ENDEC. In normal mode the transmitted pulse width is specified as 3/16 of a bit period.

The SIR receive decoder demodulates the return-to-zero bit stream from the infrared detector and outputs the received NRZ serial bit stream to USART. The decoder input is normally HIGH (marking state) in the Idle state. The transmit encoder output has the opposite polarity to the decoder input. A start bit is detected when the decoder input is low.

- IrDA is a half duplex communication protocol. If the Transmitter is busy (i.e. the USART is sending data to the IrDA encoder), any data on the IrDA receive line will be ignored by the IrDA decoder and if the Receiver is busy (USART is receiving decoded data from the USART), data on the TX from the USART to IrDA will not be encoded by IrDA. While receiving data, transmission should be avoided as the data to be transmitted could be corrupted.
- A '0' is transmitted as a high pulse and a '1' is transmitted as a '0'. The width of the pulse is specified as 3/16th of the selected bit period in normal mode (see [Figure 296](#)).
- The SIR decoder converts the IrDA compliant receive signal into a bit stream for USART.
- The SIR receive logic interprets a high state as a logic one and low pulses as logic zeros.
- The transmit encoder output has the opposite polarity to the decoder input. The SIR output is in low state when Idle.
- The IrDA specification requires the acceptance of pulses greater than 1.41 us. The acceptable pulse width is programmable. Glitch detection logic on the receiver end filters out pulses of width less than 2 PSC periods (PSC is the prescaler value programmed in the IrDA low-power Baud Register, USART\_GTPR). Pulses of width less than 1 PSC period are always rejected, but those of width greater than one and less than two periods may be accepted or rejected, those greater than 2 periods will be accepted as a pulse. The IrDA encoder/decoder doesn't work when PSC=0.
- The receiver can communicate with a low-power transmitter.
- In IrDA mode, the STOP bits in the USART\_CR2 register must be configured to "1 stop bit".

### IrDA low-power mode

#### *Transmitter:*

In low-power mode the pulse width is not maintained at 3/16 of the bit period. Instead, the width of the pulse is 3 times the low-power baud rate which can be a minimum of 1.42 MHz. Generally this value is 1.8432 MHz ( $1.42\text{ MHz} < \text{PSC} < 2.12\text{ MHz}$ ). 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  $1/\text{PSC}$ . A valid low is accepted only if its duration is greater than 2 periods of the IrDA low-power Baud clock (PSC value in USART\_GTPR).

**Note:** *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 295. IrDA SIR ENDEC- block diagram

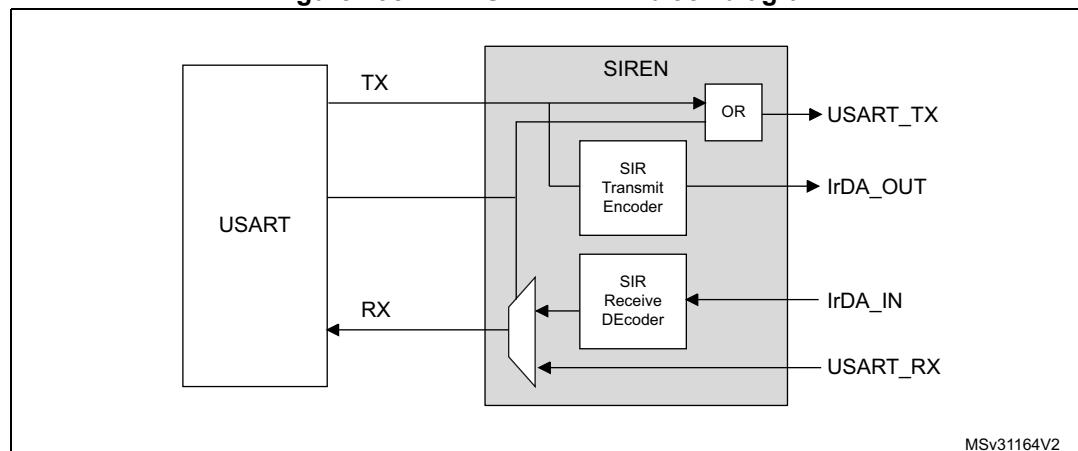
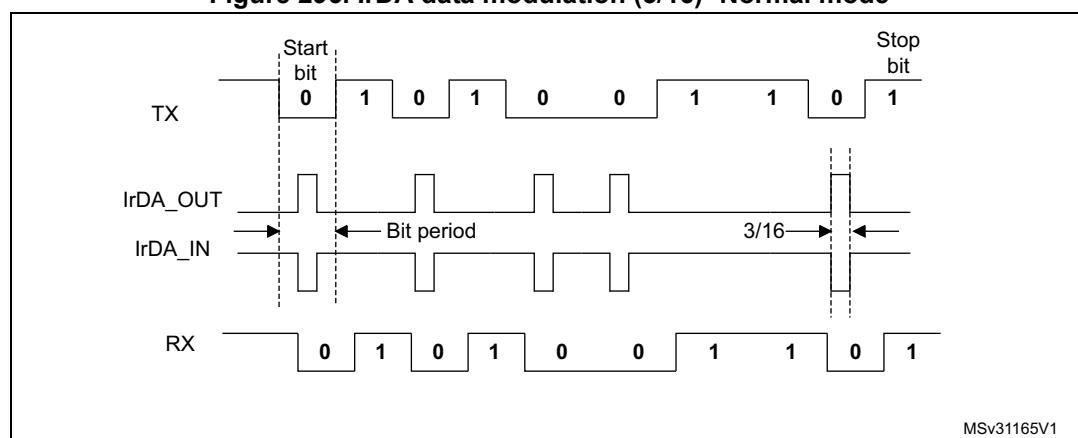


Figure 296. IrDA data modulation (3/16) -Normal mode



### 25.4.13 Continuous communication using DMA

The USART is capable of continuous communication using the DMA. The DMA requests for Rx buffer and Tx buffer are generated independently.

#### Transmission using DMA

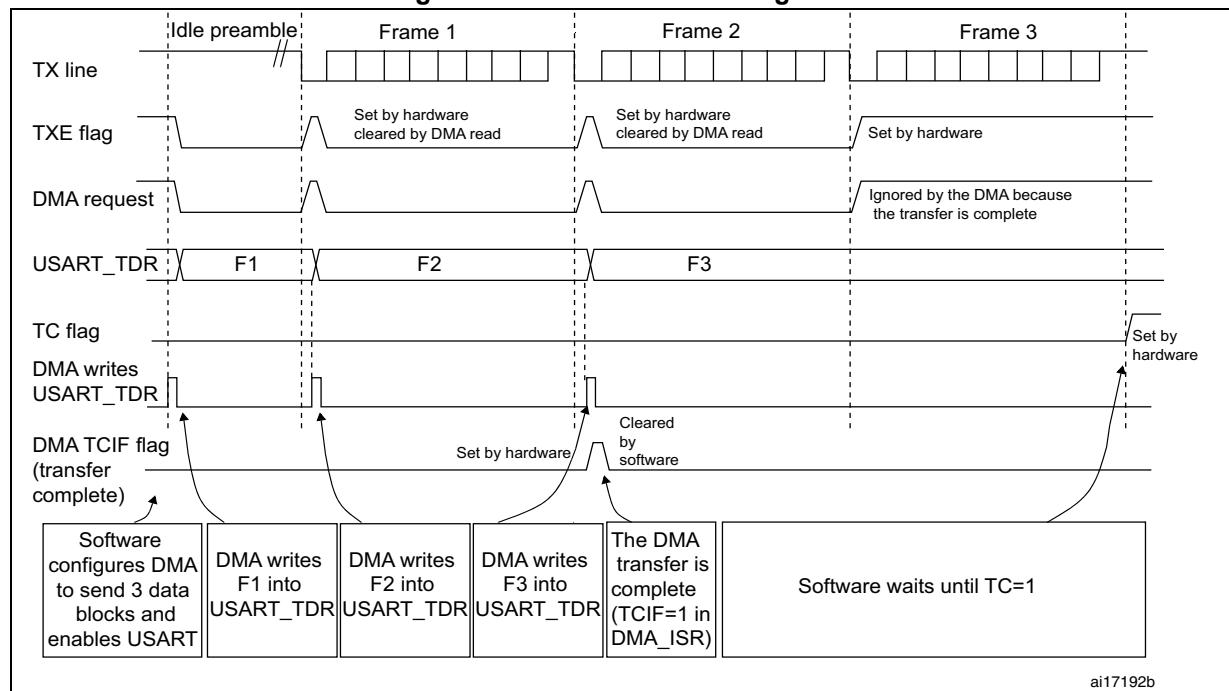
DMA mode can be enabled for transmission by setting DMAT bit in the USART\_CR3 register. Data is loaded from a SRAM area configured using the DMA peripheral (refer to the DMA specification) to the USART\_DR register whenever the TXE bit is set. To map a DMA channel for USART transmission, use the following procedure (x denotes the channel number):

1. Write the USART\_DR register address in the DMA control register to configure it as the destination of the transfer. The data will be moved to this address from memory after each TXE event.
2. Write the memory address in the DMA control register to configure it as the source of the transfer. The data will be loaded into the USART\_DR register from this memory area after each TXE event.
3. Configure the total number of bytes to be transferred to the DMA control register.
4. Configure the channel priority in the DMA register
5. Configure DMA interrupt generation after half/ full transfer as required by the application.
6. Clear the TC bit in the SR register by writing 0 to it.
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 the Stop mode. The software must wait until TC=1. The TC flag remains cleared during all data transfers and it is set by hardware at the last frame end of transmission.

Figure 297. Transmission using DMA



ai17192b

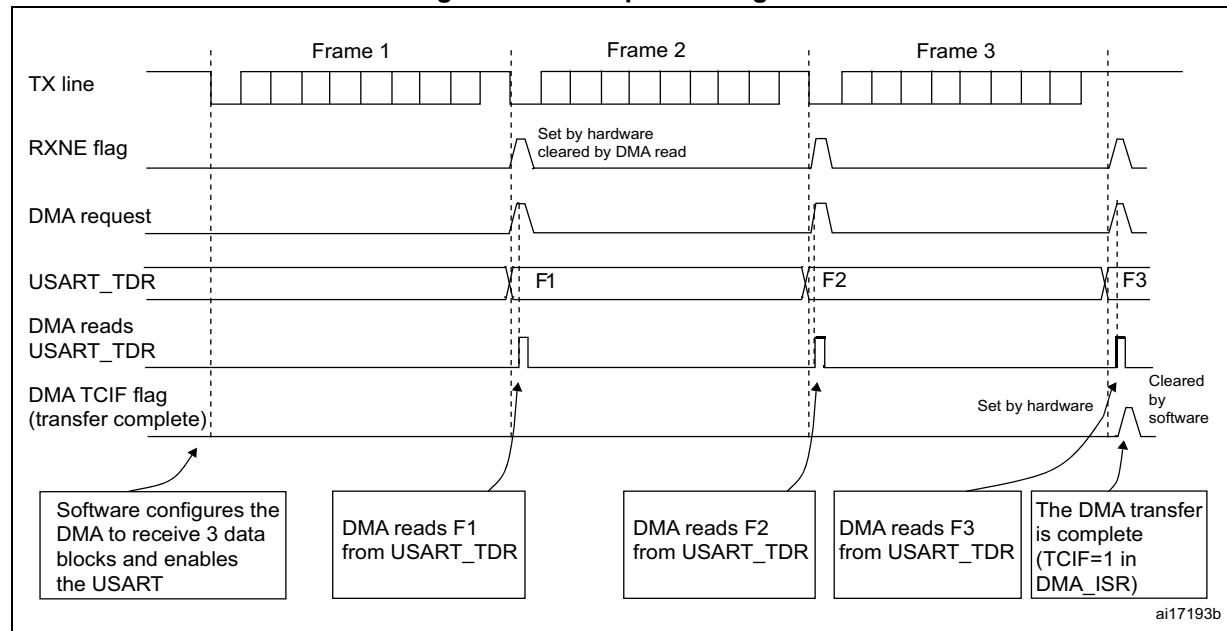
### Reception using DMA

DMA mode can be enabled for reception by setting the DMAR bit in USART\_CR3 register. Data is loaded from the USART\_DR register to a SRAM area configured using the DMA peripheral (refer to the DMA specification) whenever a data byte is received. To map a DMA channel for USART reception, use the following procedure:

1. Write the USART\_DR register address in the DMA control register to configure it as the source of the transfer. The data will be moved from this address to the memory after each RXNE event.
2. Write the memory address in the DMA control register to configure it as the destination of the transfer. The data will be loaded from USART\_DR to this memory area after each RXNE event.
3. Configure the total number of bytes to be transferred in the DMA control register.
4. Configure the channel priority in the DMA control register
5. Configure interrupt generation after half/ full transfer as required by the application.
6. Activate the channel in the DMA control register.

When the number of data transfers programmed in the DMA Controller is reached, the DMA controller generates an interrupt on the DMA channel interrupt vector. The DMAR bit should be cleared by software in the USART\_CR3 register during the interrupt subroutine.

Figure 298. Reception using DMA



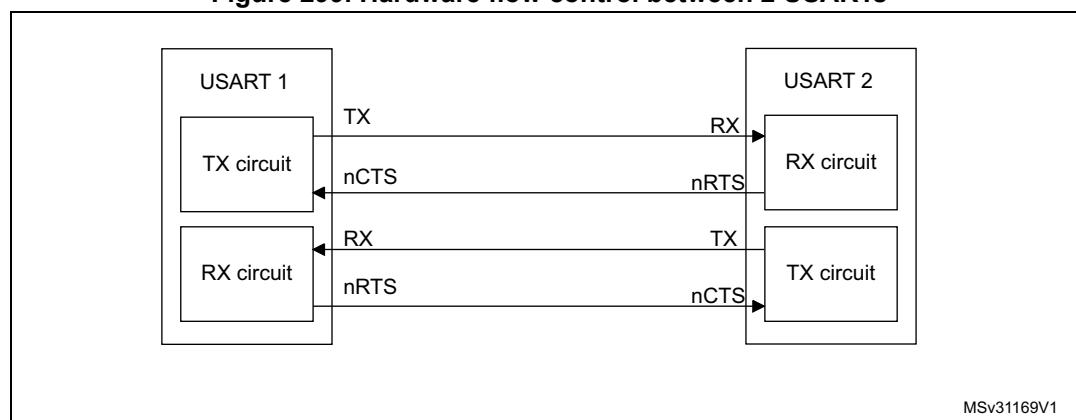
### Error flagging and interrupt generation in multibuffer communication

In case of multibuffer communication if any error occurs during the transaction the error flag will be asserted after the current byte. An interrupt will be generated if the interrupt enable flag is set. For framing error, overrun error and noise flag which are asserted with RXNE in case of single byte reception, there will be separate error flag interrupt enable bit (EIE bit in the USART\_CR3 register), which if set will issue an interrupt after the current byte with either of these errors.

#### 25.4.14 Hardware flow control

It is possible to control the serial data flow between 2 devices by using the nCTS input and the nRTS output. The [Figure 299](#) shows how to connect 2 devices in this mode:

Figure 299. Hardware flow control between 2 USARTs

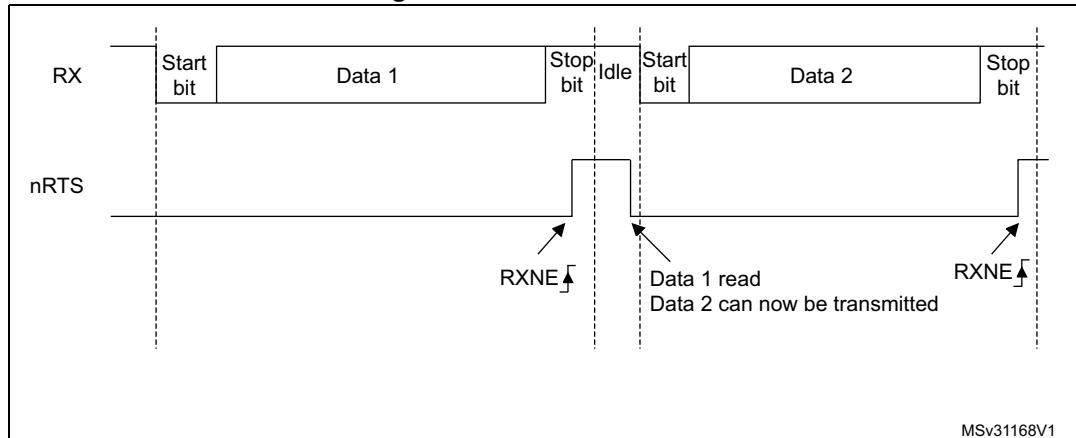


RTS and CTS flow control can be enabled independently by writing respectively RTSE and CTSE bits to 1 (in the USART\_CR3 register).

### RTS flow control

If the RTS flow control is enabled ( $RTSE=1$ ), then  $nRTS$  is asserted (tied low) as long as the USART receiver is ready to receive a new data. When the receive register is full,  $nRTS$  is deasserted, indicating that the transmission is expected to stop at the end of the current frame. [Figure 300](#) shows an example of communication with RTS flow control enabled.

**Figure 300. RTS flow control**

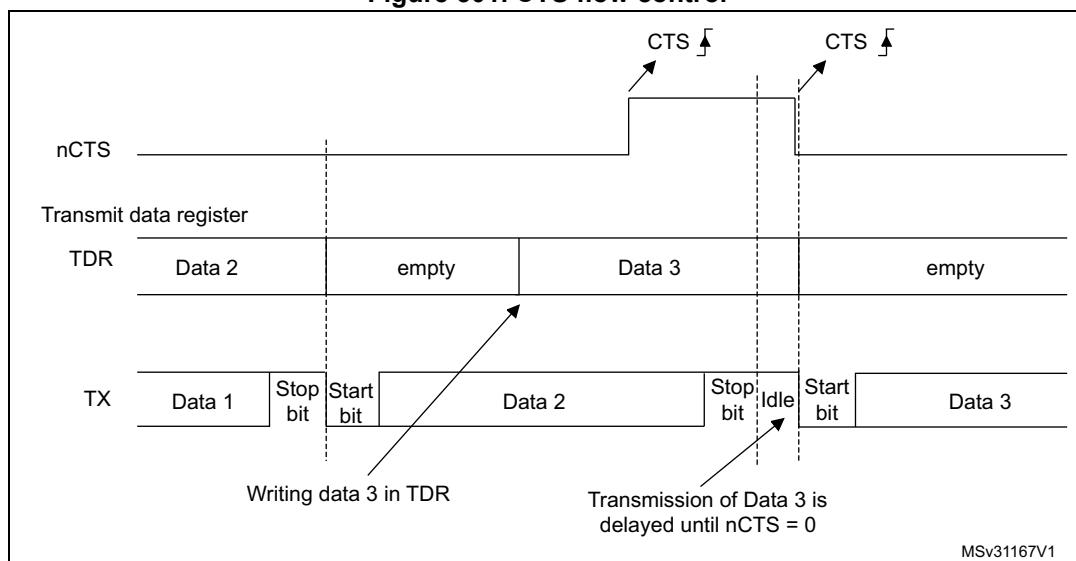


### CTS flow control

If the CTS flow control is enabled ( $CTSE=1$ ), then the transmitter checks the  $nCTS$  input before transmitting the next frame. If  $nCTS$  is asserted (tied low), then the next data is transmitted (assuming that a data is to be transmitted, in other words, if  $TXE=0$ ), else the transmission does not occur. When  $nCTS$  is deasserted during a transmission, the current transmission is completed before the transmitter stops.

When  $CTSE=1$ , the CTSIF status bit is automatically set by hardware as soon as the  $nCTS$  input toggles. It indicates when the receiver becomes ready or not ready for communication. An interrupt is generated if the CTSIE bit in the USART\_CR3 register is set. The figure below shows an example of communication with CTS flow control enabled.

**Figure 301. CTS flow control**



Note: **Special behavior of break frames:** when the CTS flow is enabled, the transmitter does not check the nCTS input state to send a break.

## 25.5 USART interrupts

Table 158. USART interrupt requests

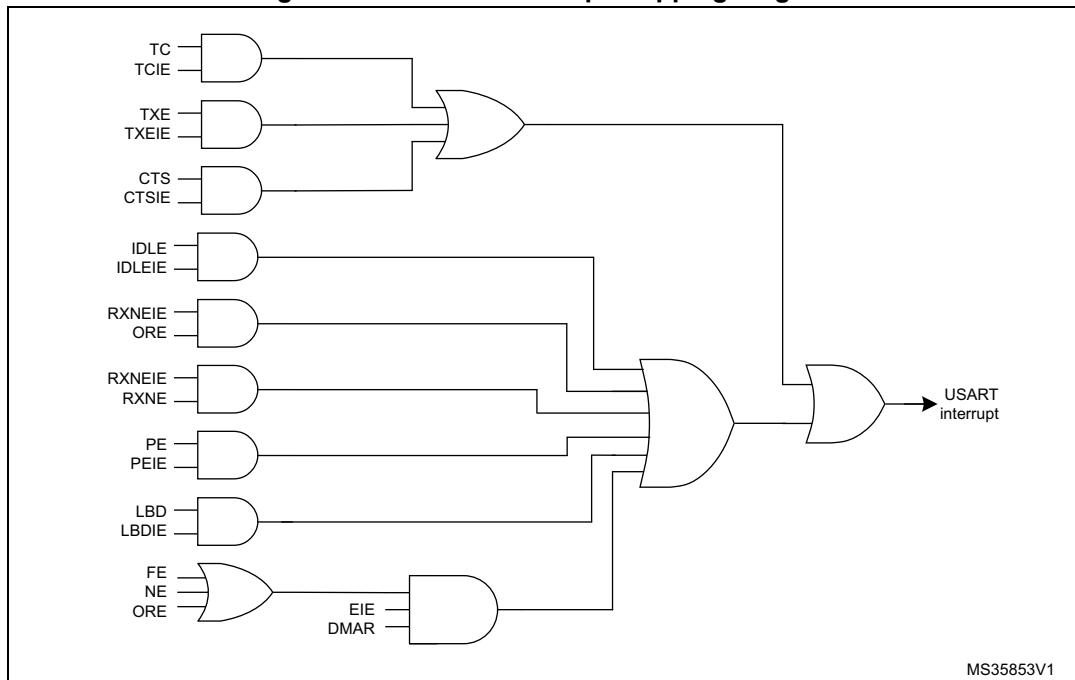
Interrupt event	Event flag	Enable control bit
Transmit Data Register Empty	TXE	TXEIE
CTS flag	CTS	CTSIE
Transmission Complete	TC	TCIE
Received Data Ready to be Read	RXNE	RXNEIE
Overrun Error Detected	ORE	
Idle Line Detected	IDLE	IDLEIE
Parity Error	PE	PEIE
Break Flag	LBD	LBDIE
Noise Flag, Overrun error and Framing Error in multibuffer communication	NF or ORE or FE	EIE

The USART interrupt events are connected to the same interrupt vector (see [Figure 302](#)).

- During transmission: Transmission Complete, Clear to Send or Transmit Data Register empty interrupt.
- While receiving: Idle Line detection, Overrun error, Receive Data register not empty, Parity error, LIN break detection, Noise Flag (only in multi buffer communication) and Framing Error (only in multi buffer communication).

These events generate an interrupt if the corresponding Enable Control Bit is set.

Figure 302. USART interrupt mapping diagram



## 25.6 USART registers

Refer to [Section 1.1 on page 51](#) for a list of abbreviations used in register descriptions.

The peripheral registers have to be accessed by half-words (16 bits) or words (32 bits).

### 25.6.1 Status register (USART\_SR)

Address offset: 0x00

Reset value: 0x00C0 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.						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
						rc_w0	rc_w0	r	rc_w0	rc_w0	r	r	r	r	r

Bits 31:10 Reserved, must be kept at reset value

Bit 9 **CTS**: CTS flag

This bit is set by hardware when the nCTS input toggles, if the CTSE bit is set. It is cleared by software (by writing it to 0). An interrupt is generated if CTSIE=1 in the USART\_CR3 register.

- 0: No change occurred on the nCTS status line
- 1: A change occurred on the nCTS status line

*Note: This bit is not available for UART4 & UART5.*

Bit 8 **LBD**: LIN break detection flag

This bit is set by hardware when the LIN break is detected. It is cleared by software (by writing it to 0). An interrupt is generated if LBDIE = 1 in the USART\_CR2 register.

- 0: LIN Break not detected
- 1: LIN break detected

*Note: An interrupt is generated when LBD=1 if LBDIE=1*

Bit 7 **TXE**: Transmit data register empty

This bit is set by hardware when the content of the TDR register has been transferred into the shift register. An interrupt is generated if the TXEIE bit =1 in the USART\_CR1 register. It is cleared by a write to the USART\_DR register.

- 0: Data is not transferred to the shift register
- 1: Data is transferred to the shift register)

*Note: This bit is used during single buffer transmission.*

Bit 6 **TC**: Transmission complete

This bit is set by hardware 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 a software sequence (a read from the USART\_SR register followed by a write to the USART\_DR register). The TC bit can also be cleared by writing a '0' to it. This clearing sequence is recommended only for multibuffer communication.

- 0: Transmission is not complete
- 1: Transmission is complete

Bit 5 **RXNE**: Read data register not empty

This bit is set by hardware when the content of the RDR shift register has been transferred to the USART\_DR register. An interrupt is generated if RXNEIE=1 in the USART\_CR1 register. It is cleared by a read to the USART\_DR register. The RXNE flag can also be cleared by writing a zero to it. This clearing sequence is recommended only for multibuffer communication.

- 0: Data is not received
- 1: Received data is ready to be read.

Bit 4 **IDLE**: IDLE line detected

This bit is set by hardware when an Idle Line is detected. An interrupt is generated if the IDLEIE=1 in the USART\_CR1 register. It is cleared by a software sequence (an read to the USART\_SR register followed by a read to the USART\_DR register).

- 0: No Idle Line is detected
- 1: Idle Line is detected

*Note: The IDLE bit will not be set again until the RXNE bit has been set itself (i.e. a new idle line occurs).*

Bit 3 **ORE**: Overrun error

This bit is set by hardware when the word currently being received in the shift register is ready to be transferred into the RDR register while RXNE=1. An interrupt is generated if RXNEIE=1 in the USART\_CR1 register. It is cleared by a software sequence (an read to the USART\_SR register followed by a read to the USART\_DR register).

- 0: No Overrun error
- 1: Overrun error is detected

*Note: When this bit is set, the RDR register content will not be lost but the shift register will be overwritten. An interrupt is generated on ORE flag in case of Multi Buffer communication if the EIE bit is set.*

Bit 2 **NF**: Noise detected flag

This bit is set by hardware when noise is detected on a received frame. It is cleared by a software sequence (an read to the USART\_SR register followed by a read to the USART\_DR register).

- 0: No noise is detected
- 1: Noise is detected

*Note: This bit does not generate interrupt as it appears at the same time as the RXNE bit which itself generates an interrupting interrupt is generated on NF flag in case of Multi Buffer 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 25.4.5: USART receiver tolerance to clock deviation on page 817](#)).*

Bit 1 **FE**: Framing error

This bit is set by hardware when a de-synchronization, excessive noise or a break character is detected. It is cleared by a software sequence (an read to the USART\_SR register followed by a read to the USART\_DR register).

- 0: No Framing error is detected
- 1: Framing error or break character is detected

*Note: This bit does not generate interrupt as it appears at the same time as the RXNE bit which itself generates an interrupt. If the word currently being transferred causes both frame error and overrun error, it will be transferred and only the ORE bit will be set.*

*An interrupt is generated on FE flag in case of Multi Buffer communication if the EIE bit is set.*

Bit 0 **PE**: Parity error

This bit is set by hardware when a parity error occurs in receiver mode. It is cleared by a software sequence (a read from the status register followed by a read or write access to the USART\_DR data register). The software must wait for the RXNE flag to be set before clearing the PE bit.

An interrupt is generated if PEIE = 1 in the USART\_CR1 register.

- 0: No parity error
- 1: Parity error

## 25.6.2 Data register (USART\_DR)

Address offset: 0x04

Reset value: 0x0000 0000

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

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

Bits 8:0 **DR[8:0]**: Data value

Contains the Received or Transmitted data character, depending on whether it is read from or written to.

The Data register performs a double function (read and write) since it is composed of two registers, one for transmission (TDR) and one for reception (RDR)

The TDR register provides the parallel interface between the internal bus and the output shift register (see Figure 1).

The RDR register provides the parallel interface between the input shift register and the internal bus.

When transmitting with the parity enabled (PCE bit set to 1 in the USART\_CR1 register), the value written in the MSB (bit 7 or bit 8 depending on the data length) has no effect because it is replaced by the parity.

When receiving with the parity enabled, the value read in the MSB bit is the received parity bit.

## 25.6.3 Baud rate register (USART\_BRR)

*Note:* The baud counters stop counting if the TE or RE bits are disabled respectively.

Address offset: 0x08

Reset value: 0x0000 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
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 15:4 **DIV\_Mantissa[11:0]**: mantissa of USARTDIV

These 12 bits define the mantissa of the USART Divider (USARTDIV)

Bits 3:0 **DIV\_Fraction[3:0]**: fraction of USARTDIV

These 4 bits define the fraction of the USART Divider (USARTDIV). When OVER8=1, the DIV\_Fraction3 bit is not considered and must be kept cleared.

## 25.6.4 Control register 1 (USART\_CR1)

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
OVER8	Res.	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
rw		rw	rw	rw	rw	rw	rw	rw	rw						

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

Bit 15 **OVER8**: Oversampling mode

- 0: oversampling by 16
- 1: oversampling by 8

*Note: Oversampling by 8 is not available in the Smartcard, IrDA and LIN modes: when SCEN=1,IREN=1 or LINEN=1 then OVER8 is forced to '0 by hardware.*

Bit 14 Reserved, must be kept at reset value

Bit 13 **UE**: USART enable

When this bit is cleared the USART prescalers and outputs are stopped and the end of the current byte transfer in order to reduce power consumption. This bit is set and cleared by software.  
 0: USART prescaler and outputs disabled  
 1: USART enabled

Bit 12 **M**: Word length

This bit determines the word length. It is set or cleared by software.  
 0: 1 Start bit, 8 Data bits, n Stop bit  
 1: 1 Start bit, 9 Data bits, n Stop bit

*Note: The M bit must not be modified during a data transfer (both transmission and reception)*

Bit 11 **WAKE**: Wakeup method

This bit determines the USART wakeup method, it is set or cleared by software.  
 0: Idle Line  
 1: Address Mark

Bit 10 **PCE**: Parity control enable

This bit selects the hardware parity control (generation and detection). When the parity control is enabled, the computed parity is inserted at the MSB position (9th bit if M=1; 8th bit if M=0) and parity is checked on the received data. This bit is set and cleared by software. Once it is set, PCE is active after the current byte (in reception and in transmission).  
 0: Parity control disabled  
 1: Parity control enabled

Bit 9 **PS**: Parity selection

This bit selects the odd or even parity when the parity generation/detection is enabled (PCE bit set). It is set and cleared by software. The parity will be selected after the current byte.  
 0: Even parity  
 1: Odd parity

**Bit 8 PEIE:** PE interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated whenever PE=1 in the USART\_SR register

**Bit 7 TXEIE:** TXE interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated whenever TXE=1 in the USART\_SR register

**Bit 6 TCIE:** Transmission complete interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated whenever TC=1 in the USART\_SR register

**Bit 5 RXNEIE:** RXNE interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated whenever ORE=1 or RXNE=1 in the USART\_SR register

**Bit 4 IDLEIE:** IDLE interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated whenever IDLE=1 in the USART\_SR register

**Bit 3 TE:** Transmitter enable

This bit enables the transmitter. It is set and cleared by software.

0: Transmitter is disabled

1: Transmitter is enabled

*Note:* 1: During transmission, a "0" pulse on the TE bit ("0" followed by "1") sends a preamble (idle line) after the current word, except in smartcard mode.

2: When TE is set there is a 1 bit-time delay before the transmission starts.

**Bit 2 RE:** Receiver enable

This bit enables the receiver. It is set and cleared by software.

0: Receiver is disabled

1: Receiver is enabled and begins searching for a start bit

**Bit 1 RWU:** Receiver wakeup

This bit determines if the USART is in mute mode or not. It is set and cleared by software and can be cleared by hardware when a wakeup sequence is recognized.

0: Receiver in active mode

1: Receiver in mute mode

*Note:* 1: Before selecting Mute mode (by setting the RWU bit) the USART must first receive a data byte, otherwise it cannot function in Mute mode with wakeup by Idle line detection.

2: In Address Mark Detection wakeup configuration (WAKE bit=1) the RWU bit cannot be modified by software while the RXNE bit is set.

**Bit 0 SBK:** Send break

This bit set is used to send break characters. It can be set and cleared by software. It should be set by software, and will be reset by hardware during the stop bit of break.

0: No break character is transmitted

1: Break character will be transmitted

### 25.6.5 Control register 2 (USART\_CR2)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LINEN	STOP[1:0]	CLKEN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	Res.					ADD[3:0]
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 **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 Synch Breaks (13 low bits) using the SBK bit in the USART\_CR1 register, and to detect LIN Sync breaks.

Bits 13:12 **STOP**: STOP bits

These bits are used for programming the stop bits.

00: 1 Stop bit

01: 0.5 Stop bit

10: 2 Stop bits

11: 1.5 Stop bit

*Note:* The 0.5 Stop bit and 1.5 Stop bit are not available for UART4 & UART5.

Bit 11 **CLKEN**: Clock enable

This bit allows the user to enable the SCLK pin.

0: SCLK pin disabled

1: SCLK pin enabled

This bit is not available for UART4 & UART5.

Bit 10 **CPOL**: Clock polarity

This bit allows the user to select the polarity of the clock output on the SCLK pin in synchronous mode. It works in conjunction with the CPHA bit to produce the desired clock/data relationship

0: Steady low value on SCLK pin outside transmission window.

1: Steady high value on SCLK pin outside transmission window.

This bit is not available for UART4 & UART5.

Bit 9 **CPHA**: Clock phase

This bit allows the user to select the phase of the clock output on the SCLK pin in synchronous mode. It works in conjunction with the CPOL bit to produce the desired clock/data relationship (see figures 290 to 291)

0: The first clock transition is the first data capture edge

1: The second clock transition is the first data capture edge

*Note:* This bit is not available for UART4 & UART5.

Bit 8 **LBCL**: Last bit clock pulse

This bit allows the user to select whether the clock pulse associated with the last data bit transmitted (MSB) has to be output on the SCLK pin in synchronous mode.

- 0: The clock pulse of the last data bit is not output to the SCLK pin
- 1: The clock pulse of the last data bit is output to the SCLK pin

*Note:* 1: The last bit is the 8th or 9th data bit transmitted depending on the 8 or 9 bit format selected by the M bit in the USART\_CR1 register.

2: This bit is not available for UART4 & UART5.

## 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 LBD=1 in the USART\_SR register

Bit 5 **LBDL**: lin break detection length

This bit is for selection between 11 bit or 10 bit break detection.

- 0: 10-bit break detection
- 1: 11-bit break detection

## Bit 4 Reserved, must be kept at reset value

Bits 3:0 **ADD[3:0]**: Address of the USART node

This bit-field gives the address of the USART node.

This is used in multiprocessor communication during mute mode, for wake up with address mark detection.

*Note:* These 3 bits (CPOL, CPHA, LBCL) should not be written while the transmitter is enabled.

### 25.6.6 Control register 3 (USART\_CR3)

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
Res	Res	Res	Res	ONEBIT	CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

## Bits 31:12 Reserved, must be kept at reset value

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

Bit 10 **CTSIE**: CTS interrupt enable

- 0: Interrupt is inhibited

1: An interrupt is generated whenever CTS=1 in the USART\_SR register

*Note:* This bit is not available for UART4 & UART5.

**Bit 9 CTSE:** CTS enable

0: CTS hardware flow control disabled

1: CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0). If the nCTS input is deasserted while a data is being transmitted, then the transmission is completed before stopping. If a data is written into the data register while nCTS is deasserted, the transmission is postponed until nCTS is asserted.

*Note: This bit is not available for UART4 & UART5.*

**Bit 8 RTSE:** RTS enable

0: RTS hardware flow control disabled

1: RTS interrupt enabled, data is only requested when there is space in the receive buffer. The transmission of data is expected to cease after the current character has been transmitted. The nRTS output is asserted (tied to 0) when a data can be received.

*Note: This bit is not available for UART4 & UART5.*

**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

*Note: This bit is not available for UART4 & UART5.*

**Bit 4 NACK:** Smartcard NACK enable

0: NACK transmission in case of parity error is disabled

1: NACK transmission during parity error is enabled

*Note: This bit is not available for UART4 & UART5.*

**Bit 3 HDSEL:** Half-duplex selection

Selection of Single-wire Half-duplex mode

0: Half duplex mode is not selected

1: Half duplex mode is selected

**Bit 2 IRLP:** IrDA low-power

This bit is used for selecting between normal and low-power IrDA modes

0: Normal mode

1: Low-power mode

**Bit 1 IREN:** IrDA mode enable

This bit is set and cleared by software.

0: IrDA disabled

1: IrDA enabled

Bit 0 **EIE**: Error interrupt enable

Error Interrupt Enable Bit is required to enable interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the USART\_SR register) in case of Multi Buffer Communication (DMAR=1 in the USART\_CR3 register).

0: Interrupt is inhibited

1: An interrupt is generated whenever DMAR=1 in the USART\_CR3 register and FE=1 or ORE=1 or NF=1 in the USART\_SR register.

### 25.6.7 Guard time and prescaler register (USART\_GTPR)

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
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 gives the Guard time value in terms of number of baud clocks.

This is used in Smartcard mode. The Transmission Complete flag is set after this guard time value.

*Note: This bit is not available for UART4 & UART5.*

Bits 7:0 **PSC[7:0]**: Prescaler value

– In IrDA Low-power mode:

**PSC[7:0]** = IrDA Low-Power Baud Rate

Used for programming the prescaler for dividing the system clock to achieve the low-power frequency:

The source clock is divided by the value given in the register (8 significant bits):

00000000: Reserved - do not program this value

00000001: divides the source clock by 1

00000010: divides the source clock by 2

...

– In normal IrDA mode: PSC must be set to 00000001.

– In smartcard mode:

**PSC[4:0]**: Prescaler value

Used for programming the prescaler for dividing the system clock to provide the smartcard clock.

The value given in the register (5 significant bits) is multiplied by 2 to give the division factor of the source clock frequency:

00000: Reserved - do not program this value

00001: divides the source clock by 2

00010: divides the source clock by 4

00011: divides the source clock by 6

...

*Note: 1: Bits [7:5] have no effect if Smartcard mode is used.*

*2: This bit is not available for UART4 & UART5.*

## 25.6.8 USART register map

The table below gives the USART register map and reset values.

**Table 159. USART register map and reset values**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x00	<b>USART_SR</b>	Res.	Res.																															
	Reset value	Res.	Res.																															
0x04	<b>USART_DR</b>	Res.	DR[8:0]																															
	Reset value	Res.	Res.																															
0x08	<b>USART_BRR</b>	Res.	DIV_Mantissa[15:4]	DIV_Fraction[3:0]																														
	Reset value	Res.	Res.																															
0x0C	<b>USART_CR1</b>	Res.	Res.																															
	Reset value	Res.	Res.																															
0x10	<b>USART_CR2</b>	Res.	Res.																															
	Reset value	Res.	Res.																															
0x14	<b>USART_CR3</b>	Res.	GT[7:0]	PSC[7:0]																														
	Reset value	Res.	Res.	Res.																														
0x18	<b>USART_GTPR</b>	Res.	Res.	Res.																														
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Refer to [Section 2.2.2 on page 56](#) for the register boundary addresses.

## 26 Serial peripheral interface/ inter-IC sound (SPI/I2S)

### 26.1 Introduction

The SPI/I<sup>2</sup>S interface can be used to communicate with external devices using the SPI protocol or the I<sup>2</sup>S audio protocol. SPI or I<sup>2</sup>S mode is selectable by software. SPI 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 (I<sup>2</sup>S) protocol is also a synchronous serial communication interface. It can operate in slave or master mode with half-duplex communication. Full duplex operations are possible by combining two I<sup>2</sup>S blocks.

It can address four different audio standards including the Philips I<sup>2</sup>S standard, the MSB- and LSB-justified standards and the PCM standard.

#### 26.1.1 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)
- 8-bit to 16-bit transfer frame format 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
- 1-byte/word transmission and reception buffer with DMA capability: Tx and Rx requests

### 26.1.2 SPI extended features

- SPI TI mode support

### 26.1.3 I2S features

- Half-duplex communication (only transmitter or receiver)
- Master or slave operations
- 8-bit programmable linear prescaler to reach accurate audio sample frequencies (from 8 kHz to 192 kHz)
- Data format may be 16-bit, 24-bit or 32-bit
- Packet frame is fixed to 16-bit (16-bit data frame) or 32-bit (16-bit, 24-bit, 32-bit data frame) by audio channel
- Programmable clock polarity (steady state)
- Underrun flag in slave transmission mode, overrun flag in reception mode (master and slave) and Frame Error Flag in reception and transmitter mode (slave only)
- 16-bit register for transmission and reception with one data register for both channel sides
- Supported I<sup>2</sup>S protocols:
  - I<sup>2</sup>S Philips standard
  - MSB-Justified standard (Left-Justified)
  - LSB-Justified standard (Right-Justified)
  - PCM standard (with short and long frame synchronization on 16-bit channel frame or 16-bit data frame extended to 32-bit channel frame)
- Data direction is always MSB first
- DMA capability for transmission and reception (16-bit wide)
- Master clock can be output to drive an external audio component. Ratio is fixed at  $256 \times F_S$  (where  $F_S$  is the audio sampling frequency)
- I<sup>2</sup>S (I2S1, I2S2 and I2S3) clock can be derived from an external clock mapped on the I2S\_CKIN pin.

## 26.2 SPI/I2S implementation

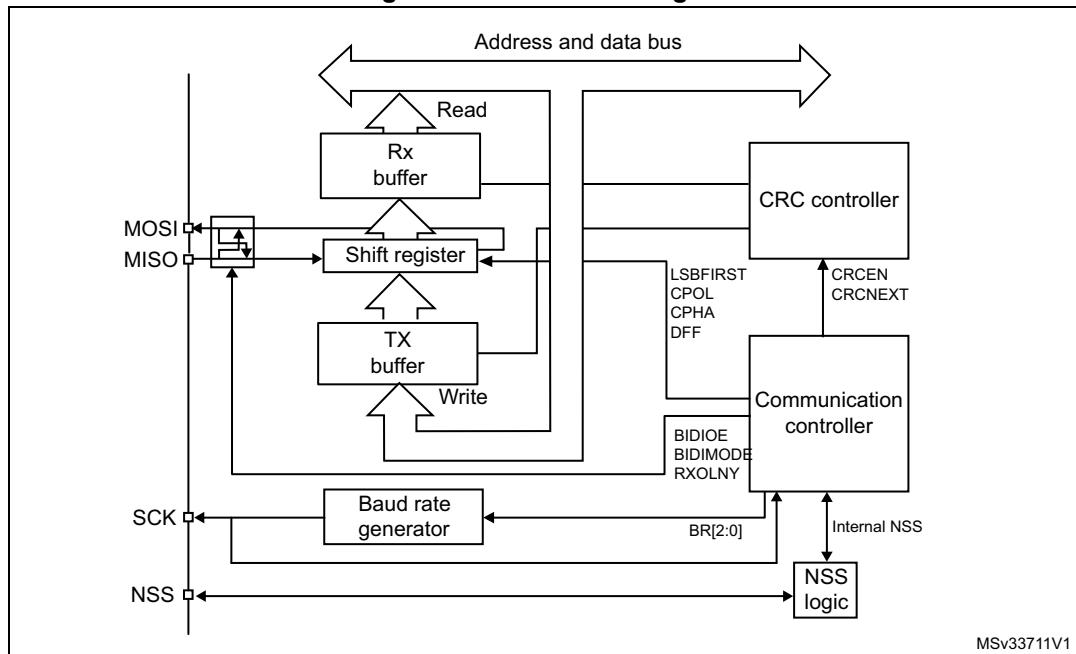
This manual describes the full set of features implemented in SPI1 and SPI2.

## 26.3 SPI functional description

### 26.3.1 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 303](#).

**Figure 303. SPI block diagram**



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 26.3.5: 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.

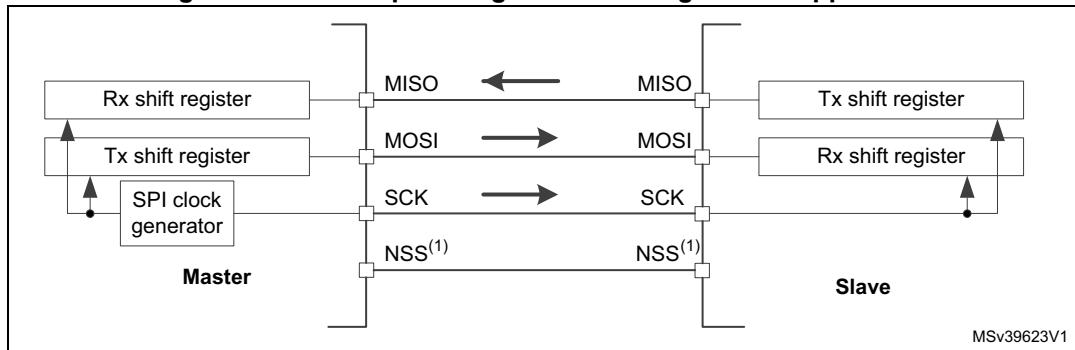
### 26.3.2 Communications between one master and one slave

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 304. Full-duplex single master/ single slave application**

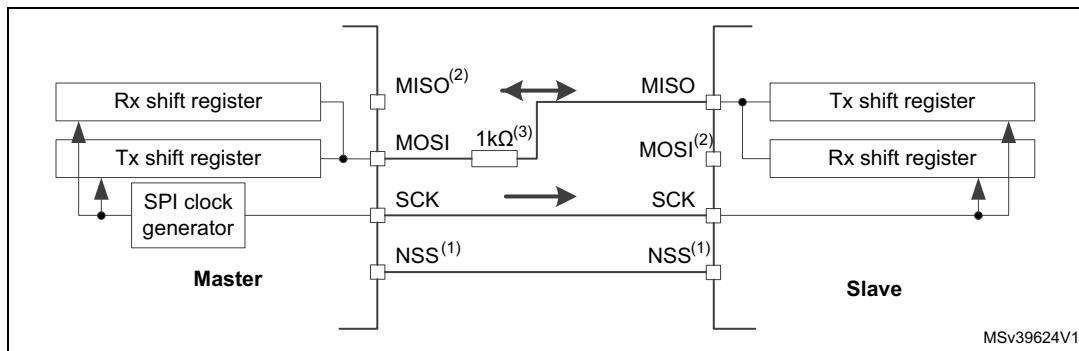


1. The NSS pins can be used to provide a hardware control flow between master and slave. Optionally, the pins can be left unused by the peripheral. Then the flow has to be handled internally for both master and slave. For more details see [Section 26.3.5: Slave select \(NSS\) pin management](#).

#### 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 305. Half-duplex single master/ single slave application



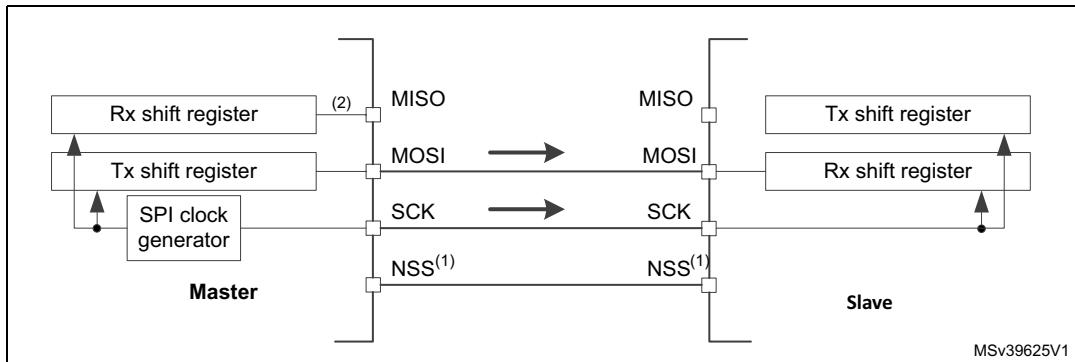
1. The NSS pins can be used to provide a hardware control flow between master and slave. Optionally, the pins can be left unused by the peripheral. Then the flow has to be handled internally for both master and slave. For more details see [Section 26.3.5: Slave select \(NSS\) pin management](#).
2. In 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 [26.3.5: 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 306. 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 26.3.5: 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.

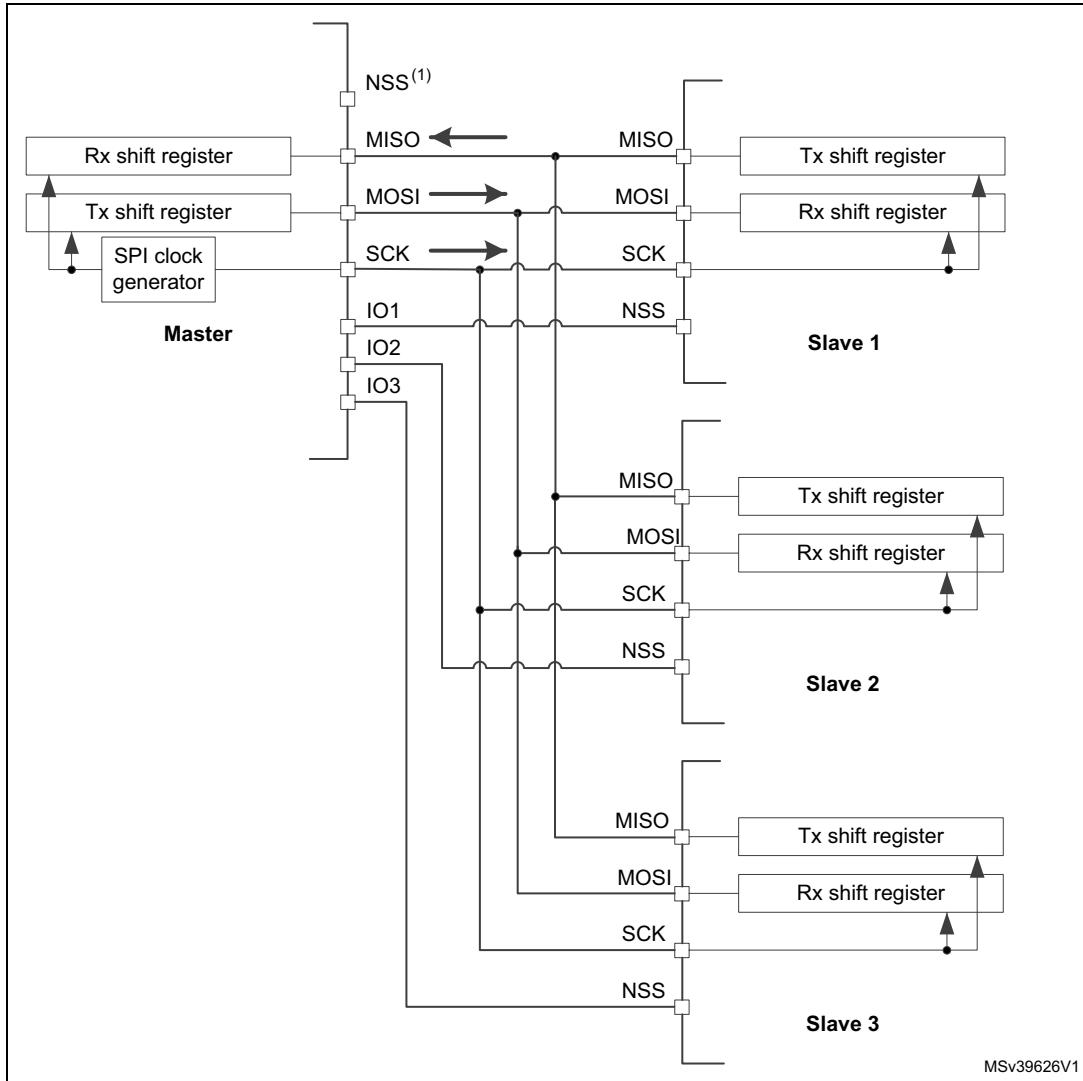
Note:

*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).*

### 26.3.3 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 307](#)). 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 307. 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 on page 181](#)).

### 26.3.4 Multi-master communication

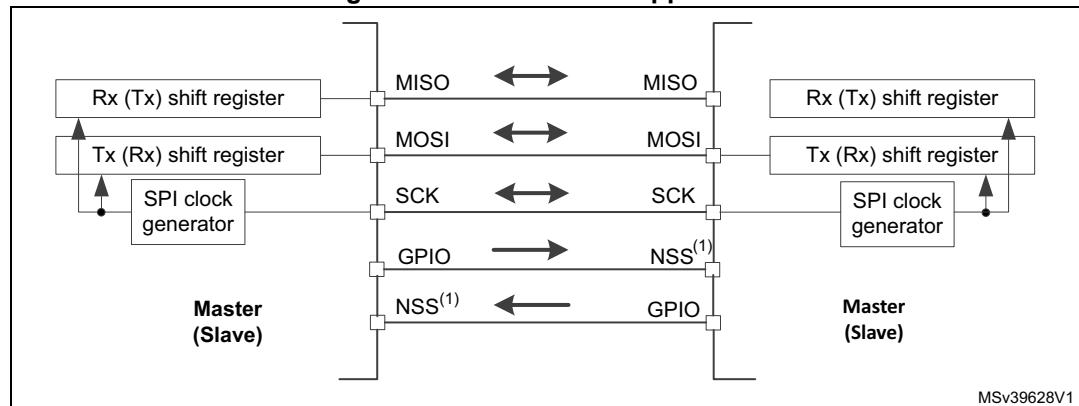
Unless SPI bus is not designed for a multi-master capability primarily, the user can use build in feature which detects a potential conflict between two nodes trying to master the bus at the same time. For this detection, NSS pin is used configured at hardware input mode.

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 308. 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.

### 26.3.5 Slave select (NSS) pin management

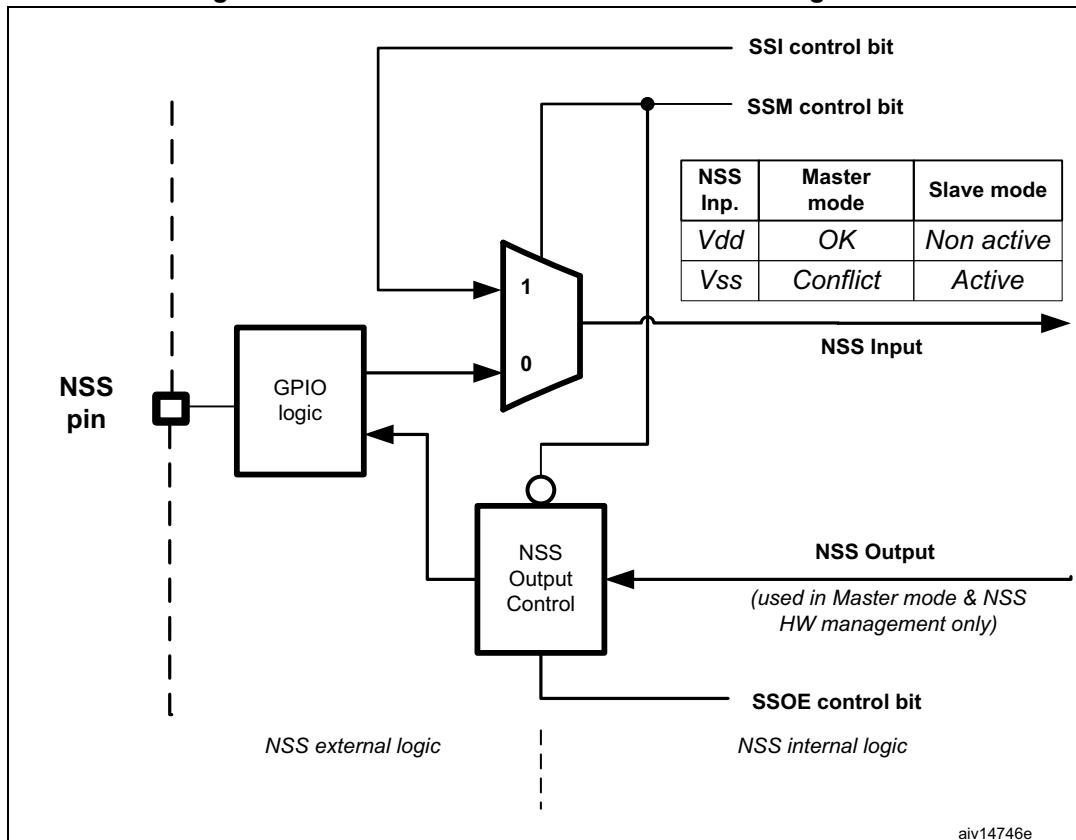
In slave mode, the NSS works as a standard “chip select” input and lets the slave communicate with the master. In master mode, NSS can be used either as output or input. As an input it can prevent multimaster bus collision, and as an output it can drive a slave select signal of a single slave.

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).
- **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 309. Hardware/software slave select management



### 26.3.6 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.

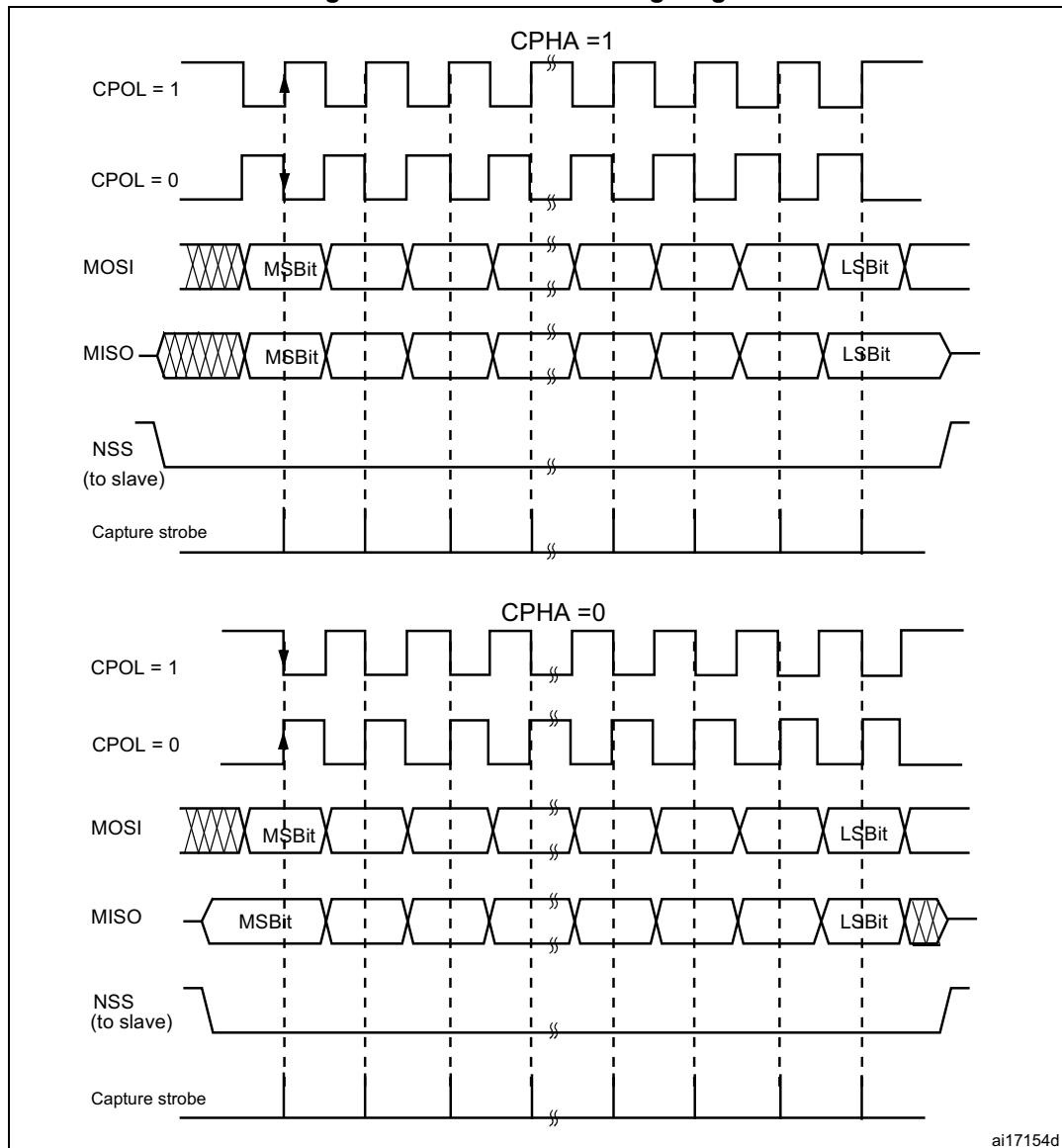
If the CPHA bit is set, the second edge on the SCK pin captures the first data bit transacted (falling edge if the CPOL bit is reset, rising edge if the CPOL bit is set). Data are latched on each occurrence of this clock transition type. If the CPHA bit is reset, the first edge on the SCK pin captures 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 310*, 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 310. Data clock timing diagram



ai17154d

Note: 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. Each data frame is 8 or 16 bit long depending on the size of the data programmed using the *DFF* bit in the *SPI\_CR1* register. The selected data frame format is applicable both for transmission and reception.

### 26.3.7 SPI configuration

The configuration procedure is almost the same for master and slave. For specific mode setups, follow the dedicated chapters. 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*: 3).
  - b) Configure the CPOL and CPHA bits combination to define one of the four relationships between the data transfer and the serial clock. (*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 CRCEN and CRCEN bits if CRC is needed (while SCK clock signal is at idle state).
  - f) Configure SSM and SSI (*Note*: 2).
  - g) Configure the MSTR bit (in multimaster NSS configuration, avoid conflict state on NSS if master is configured to prevent MODF error).
  - h) Set the DFF bit to configure the data frame format (8 or 16 bits).
3. Write to SPI\_CR2 register:
  - a) Configure SSOE (*Note*: 1 & 2).
  - b) Set the FRF bit if the TI protocol is required.
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) The step is not required in slave mode except slave working at TI mode.

### 26.3.8 Procedure for enabling SPI

It is recommended to enable the SPI slave before the master sends the clock. Otherwise, undesired data transmission might occur. The slave data register 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.

At full-duplex (or in any transmit-only mode), the master starts communicating when the SPI is enabled and data to be sent is written in the Tx Buffer.

In any master receive-only mode (RXONLY=1 or BIDIMODE=1 & BIDIOE=0), the master starts communicating and the clock starts running immediately after the SPI is enabled.

The slave starts communicating when it receives a correct clock signal from the master. The slave software must write the data to be sent before the SPI master initiates the transfer.

Refer to [Section 26.3.11: Communication using DMA \(direct memory addressing\)](#) for details on how to handle DMA.

## 26.3.9 Data transmission and reception procedures

### Rx and Tx buffers

In reception, data are received and then stored into an internal Rx buffer while in transmission, data are first stored into an internal Tx buffer before being transmitted. A read access to the SPI\_DR register returns the Rx buffered value whereas a write access to the SPI\_DR stores the written data into the Tx buffer.

### Tx buffer handling

The data frame is loaded from the Tx buffer into the shift register during the first bit transmission. Bits are then shifted out serially from the shift register to a dedicated output pin depending on LSBFIRST bit setting. The TXE flag (Tx buffer empty) is set when the data are transferred from the Tx buffer to the shift register. It indicates that the internal Tx buffer is ready to be loaded with the next data. An interrupt can be generated if the TXIE bit of the SPI\_CR2 register is set. Clearing the TXE bit is performed by writing to the SPI\_DR register.

A continuous transmit stream can be achieved if the next data to be transmitted are stored in the Tx buffer while previous frame transmission is still ongoing. When the software writes to Tx buffer while the TXE flag is not set, the data waiting for transaction is overwritten.

### Rx buffer handling

The RXNE flag (Rx buffer not empty) is set on the last sampling clock edge, when the data are transferred from the shift register to the Rx buffer. It indicates that data are ready to be read from the SPI\_DR register. An interrupt can be generated if the RXNEIE bit in the SPI\_CR2 register is set. Clearing the RXNE bit is performed by reading the SPI\_DR register.

If a device has not cleared the RXNE bit resulting from the previous data byte transmitted, an overrun condition occurs when the next value is buffered. The OVR bit is set and an interrupt is generated if the ERRIE bit is set.

Another way to manage the data exchange is to use DMA (see [Section 9.2: DMA main features](#)).

### Sequence handling

The BSY bit is set when a current data frame transaction is ongoing. When the clock signal runs continuously, the BSY flag remains set between data frames on the master side. However, on the slave side, it becomes low for a minimum duration of one SPI clock cycle between each data frame transfer.

For some configurations, the BSY flag can be used during the last data transfer to wait until the completion of the transfer.

When a receive-only mode is configured on the master side, either in half-duplex (BIDIMODE=1, BIDIOE=0) or simplex configuration (BIDIMODE=0, RXONLY=1), the master starts the receive sequence as soon as the SPI is enabled. Then the clock signal is provided by the master and it does not stop until either the SPI or the receive-only mode is disabled by the master. The master receives data frames continuously up to this moment.

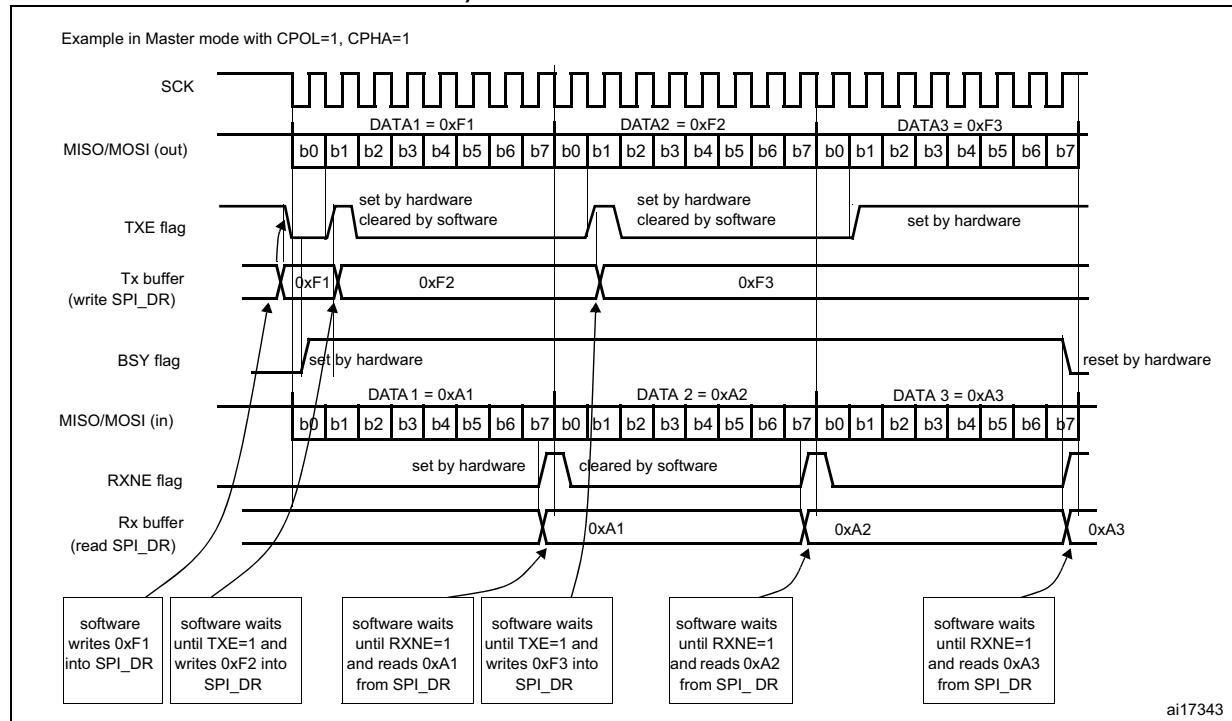
While the master can provide all the transactions in continuous mode (SCK signal is continuous), it has to respect slave capability to handle data flow and its content at anytime. When necessary, the master must slow down the communication and provide either a slower clock or separate frames or data sessions with sufficient delays. Be aware there is no

underflow error signal for slave operating in SPI mode, and that data from the slave are always transacted and processed by the master even if the slave cannot not prepare them correctly in time. It is preferable for the slave to use DMA, especially when data frames are shorter and bus rate is high.

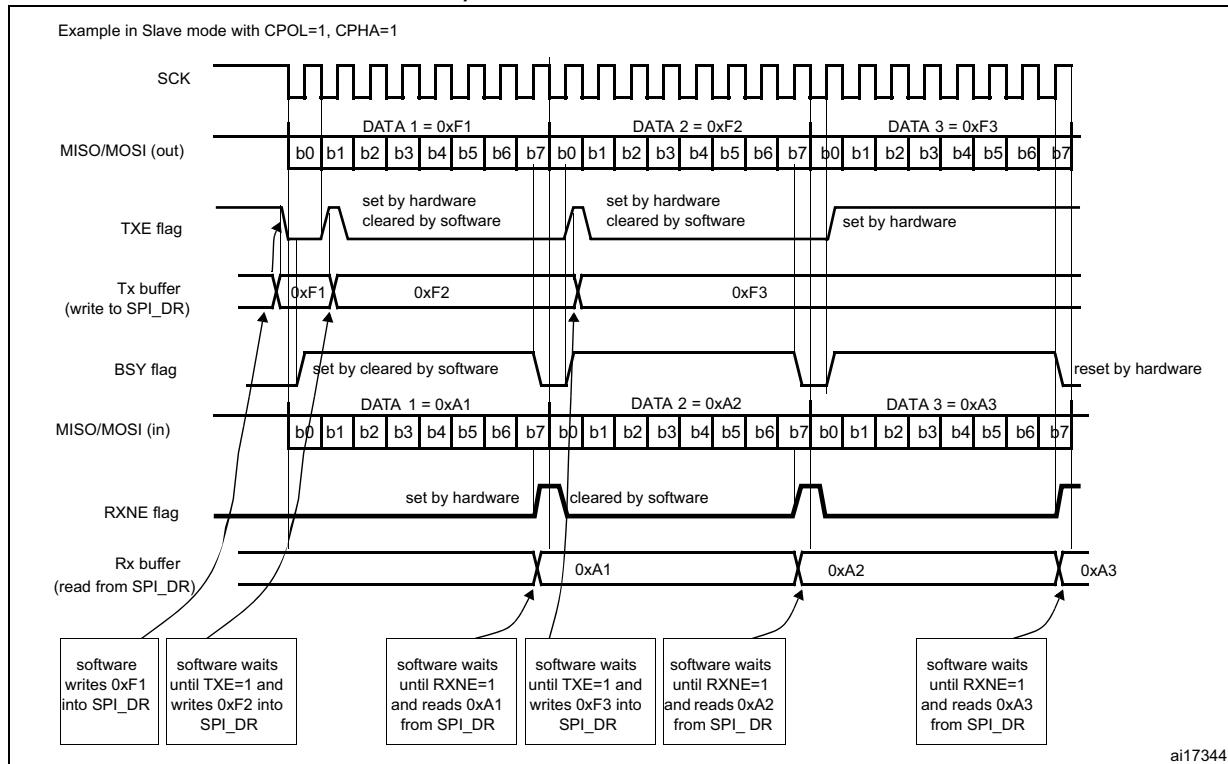
Each sequence must be encased by the NSS pulse in parallel with the multislide system to select just one of the slaves for communication. In single slave systems, using NSS to control the slave is not necessary. However, the NSS pulse can be used to synchronize the slave with the beginning of each data transfer sequence. NSS can be managed either by software or by hardware (see [Section 26.3.4: Multi-master communication](#)).

Refer to [Figure 311](#) and [Figure 312](#) for a description of continuous transfers in master / full-duplex and slave full-duplex mode.

**Figure 311. TXE/RXNE/BSY behavior in master / full-duplex mode (BIDIMODE=0, RXONLY=0) in the case of continuous transfers**



**Figure 312. TXE/RXNE/BSY behavior in slave / full-duplex mode (BIDIMODE=0, RXONLY=0) in the case of continuous transfers**



### 26.3.10 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.

Standard disable procedure is based on pulling BSY status together with TXE flag 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 an arbitrary GPIO toggle and the master has to provide proper end of NSS pulse for slave, or
- When transactions' streams from DMA 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 RXNE=1 to receive the last data.
2. Wait until TXE=1 and then wait until BSY=0 before disabling the SPI.
3. Read received data.

**Note:** During discontinuous communications, there is a 2 APB clock period delay between the write operation to the SPI\_DR register and BSY bit setting. As a consequence it is mandatory to wait first until TXE is set and then until BSY is cleared after writing the last 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 received data.

**Note:** To stop a continuous receive sequence, a specific time window must be respected during the reception of the last data frame. It starts when the first bit is sampled and ends before the last bit transfer starts.

### 26.3.11 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.

Refer to [Figure 313](#) and [Figure 314](#) for a description of the DMA transmission and reception waveforms.

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 TXE = 1 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.

**Figure 313. Transmission using DMA**

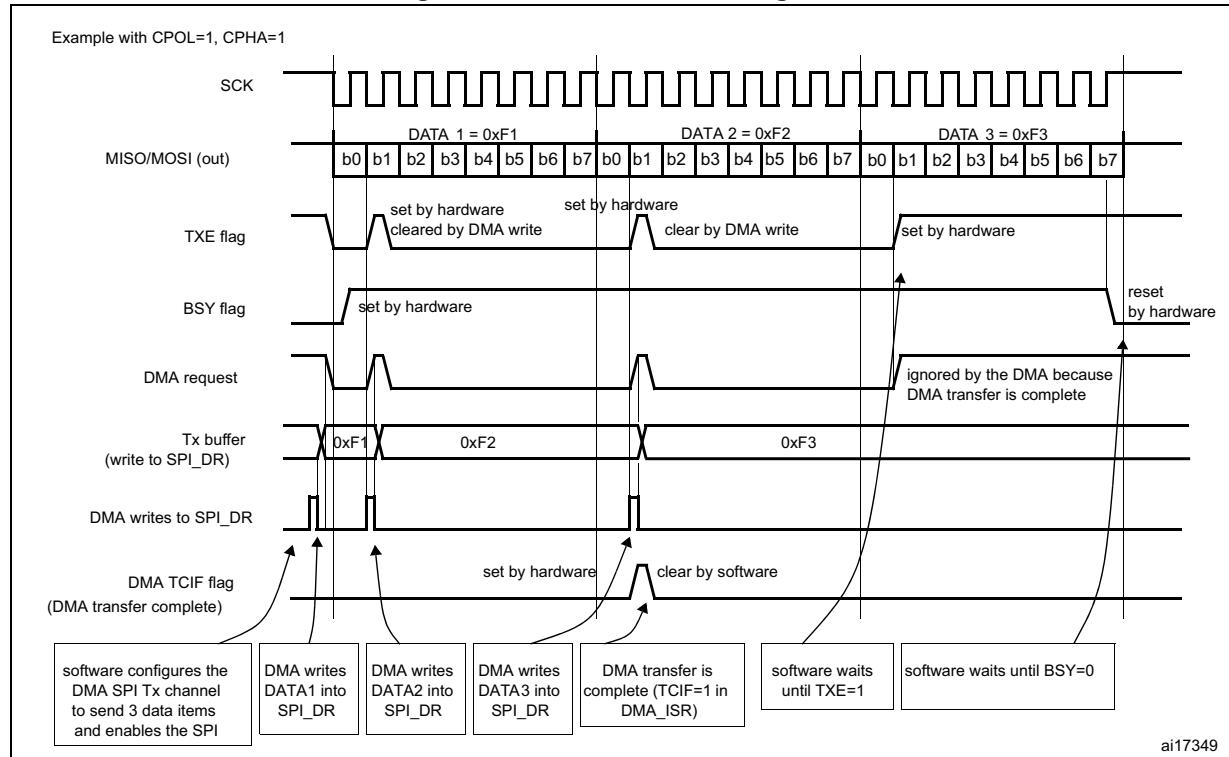
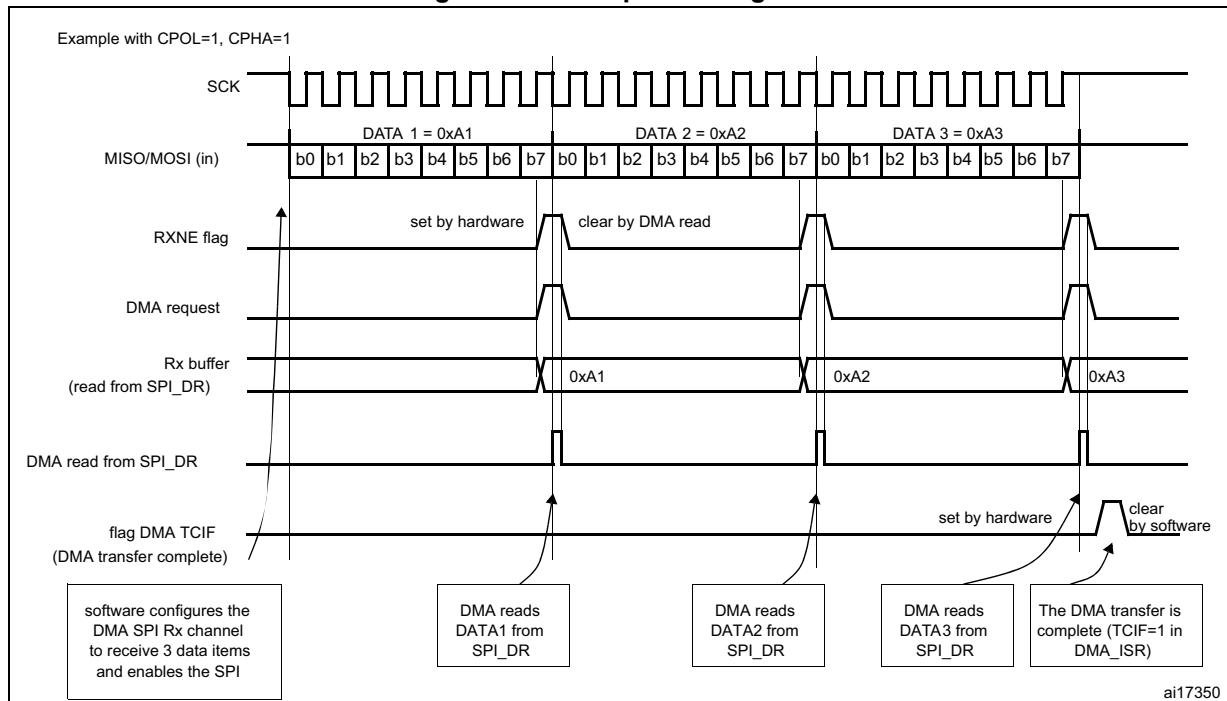


Figure 314. Reception using DMA



### 26.3.12 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)

When it is set, the TXE flag indicates that the Tx buffer is empty and that the next data to be transmitted can be loaded into the buffer. The TXE flag is cleared by writing to the SPI\_DR register.

#### Rx buffer not empty (RXNE)

When set, the RXNE flag indicates that there are valid received data in the Rx buffer. It is cleared by reading from the SPI\_DR register.

#### 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). There is one exception in master bidirectional receive mode (MSTR=1 and BDM=1 and BDOE=0) where the BSY flag is kept low during reception.

The BSY flag can be used in certain modes to detect the end of a transfer, thus preventing corruption of the last transfer when the SPI peripheral clock is disabled before entering a low-power mode or an NSS pulse end is handled by software.

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:** *It is recommended to use always the TXE and RXNE flags (instead of the BSY flags) to handle data transmission or reception operations.*

### 26.3.13 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 the master or the slave completes the reception of the next data frame while the read operation of the previous frame from the Rx buffer has not completed (case RXNE flag is set).

In this case, the content of the Rx buffer is not updated with the new data received. A read operation from the SPI\_DR register returns the frame previously received. All other subsequently transmitted data are lost.

Clearing the OVR bit is done by a read access to the SPI\_DR register followed by a read access to the SPI\_SR register.

#### 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\_RXCRC 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 re-initiated by the master when the slave SPI is enabled again.

## 26.4 SPI special features

### 26.4.1 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 315](#)). 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:

$$\frac{t_{baud\_rate}}{2} + 4 \times t_{pclk} < t_{release} < \frac{t_{baud\_rate}}{2} + 6 \times t_{pclk}$$

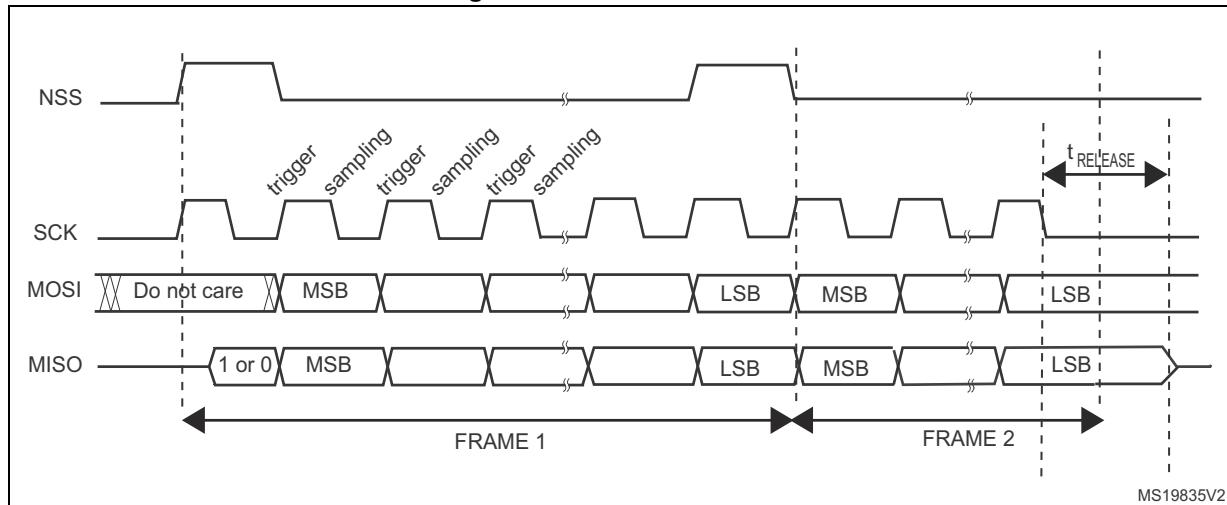
If the slave detects a misplaced NSS pulse during a data frame transaction the TIFRE flag is set.

This feature is not available for Motorola SPI communications (FRF bit set to 0).

**Note:** To detect TI frame errors in slave transmitter only mode by using the Error interrupt (ERRIE=1), the SPI must be configured in 2-line unidirectional mode by setting BIDIMODE and BIDIOE to 1 in the SPIx\_CR1 register. When BIDIMODE is set to 0, OVR is set to 1 because the data register is never read and error interrupts are always generated, while when BIDIMODE is set to 1, data are not received and OVR is never set.

Figure 315 shows the SPI communication waveforms when TI mode is selected.

Figure 315. TI mode transfer



## 26.4.2 CRC calculation

Two separate CRC calculators (on transmission and reception data flows) are implemented in order to check the reliability of transmitted and received data. The SPI offers CRC8 or CRC16 calculation depending on the data format selected through the DFF bit. The CRC is calculated serially using the polynomial programmed in the SPI\_CRCPR register.

### 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 Rx buffer like any other data frame.

A CRC-format transaction takes one more data frame to communicate at the end of data sequence.

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 Rx buffer 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 the CRC frame received from SPI\_DR as it is always loaded into it.

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.

### Resetting the SPIx\_TXCRC and SPIx\_RXCRC values

The SPIx\_TXCRC and SPIx\_RXCRC values are cleared automatically when CRC calculation is enabled.

When the SPI is configured in slave mode with the CRC feature enabled, a CRC calculation is performed even if a high level is applied on the NSS pin. This may happen for example in case of a multislave environment where the communication master addresses slaves alternately.

Between a slave disabling (high level on NSS) and a new slave enabling (low level on NSS), the CRC value should be cleared on both master and slave sides to resynchronize the master and slave respective CRC calculation.

To clear the CRC, follow the below sequence:

1. Disable the SPI
2. Clear the CRCEN bit
3. Enable the CRCEN bit
4. Enable the SPI

Note:

*When the SPI is in slave mode, the CRC calculator is sensitive to the SCK slave input clock as soon as the CRCEN bit is set, and this is the case whatever the value of the SPE bit. In order to avoid any wrong CRC calculation, the software must enable the CRC calculation only when the clock is stable (in steady state). When the SPI interface is configured as a slave, the NSS internal signal needs to be kept low between the data phase and the CRC phase once the CRCNEXT signal is released.*

*At TI mode, despite the fact that the clock phase and clock polarity setting is fixed and independent on the 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 the SPI disable sequence by re-enabling the CRCEN bit described above at both master and slave sides, else the CRC calculation can be corrupted at this specific mode.*

## 26.5 SPI interrupts

During SPI communication an interrupts can be generated by the following events:

- Transmit Tx buffer ready to be loaded
- Data received in Rx buffer
- Master mode fault
- Overrun error
- TI frame format error

Interrupts can be enabled and disabled separately.

**Table 160. SPI interrupt requests**

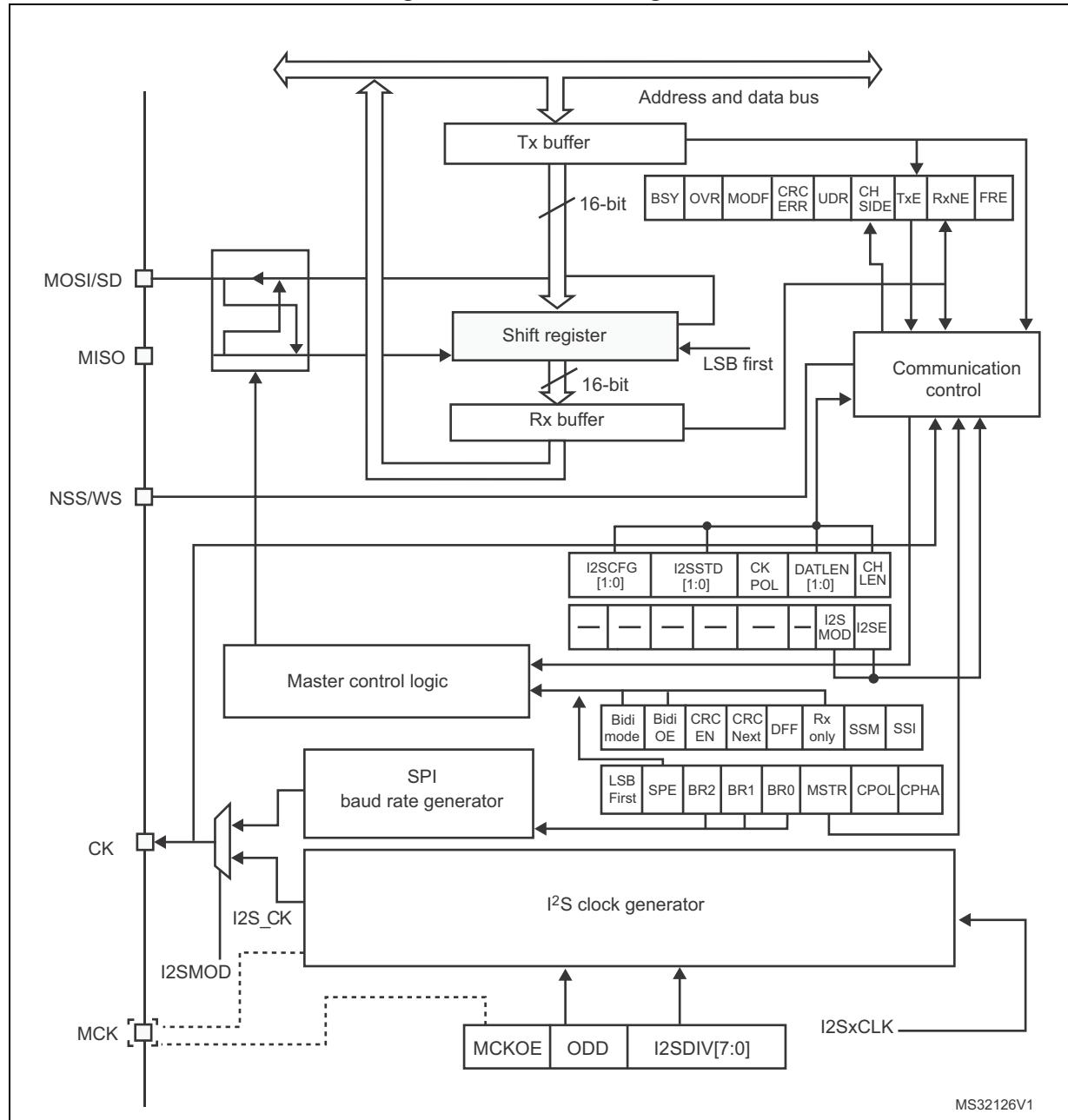
Interrupt event	Event flag	Enable Control bit
Transmit Tx buffer ready to be loaded	TXE	TXEIE
Data received in Rx buffer	RXNE	RXNEIE
Master Mode fault event	MODF	ERRIE
Overrun error	OVR	
CRC error	CRCERR	
TI frame format error	FRE	

## 26.6 I<sup>2</sup>S functional description

### 26.6.1 I<sup>2</sup>S general description

The block diagram of the I<sup>2</sup>S is shown in [Figure 316](#).

**Figure 316. I<sup>2</sup>S block diagram**



1. MCK is mapped on the MISO pin.

The SPI can function as an audio I<sup>2</sup>S interface when the I<sup>2</sup>S capability is enabled (by setting the I<sup>2</sup>SMOD bit in the SPIx\_I<sup>2</sup>SCFGR register). This interface mainly uses the same pins, flags and interrupts as the SPI.

The I<sup>2</sup>S shares three common pins with the SPI:

- SD: Serial Data (mapped on the MOSI pin) to transmit or receive the two time-multiplexed data channels (in half-duplex mode only).
- WS: Word Select (mapped on the NSS pin) is the data control signal output in master mode and input in slave mode.
- CK: Serial Clock (mapped on the SCK pin) is the serial clock output in master mode and serial clock input in slave mode.

An additional pin can be used when a master clock output is needed for some external audio devices:

- MCK: Master Clock (mapped separately) is used, when the I<sup>2</sup>S is configured in master mode (and when the MCKOE bit in the SPIx\_I2SPR register is set), to output this additional clock generated at a preconfigured frequency rate equal to  $256 \times f_S$ , where  $f_S$  is the audio sampling frequency.

The I<sup>2</sup>S uses its own clock generator to produce the communication clock when it is set in master mode. This clock generator is also the source of the master clock output. Two additional registers are available in I<sup>2</sup>S mode. One is linked to the clock generator configuration SPIx\_I2SPR and the other one is a generic I<sup>2</sup>S configuration register SPIx\_I2SCFGR (audio standard, slave/master mode, data format, packet frame, clock polarity, etc.).

The SPIx\_CR1 register and all CRC registers are not used in the I<sup>2</sup>S mode. Likewise, the SSOE bit in the SPIx\_CR2 register and the MODF and CRCERR bits in the SPIx\_SR are not used.

The I<sup>2</sup>S uses the same SPI register for data transfer (SPIx\_DR) in 16-bit wide mode.

## 26.6.2 I<sup>2</sup>S full-duplex

*Figure 317* shows how to perform full-duplex communications using two SPI2S instances. In this case, the WS and CK IOs of both SPI2S must be connected together.

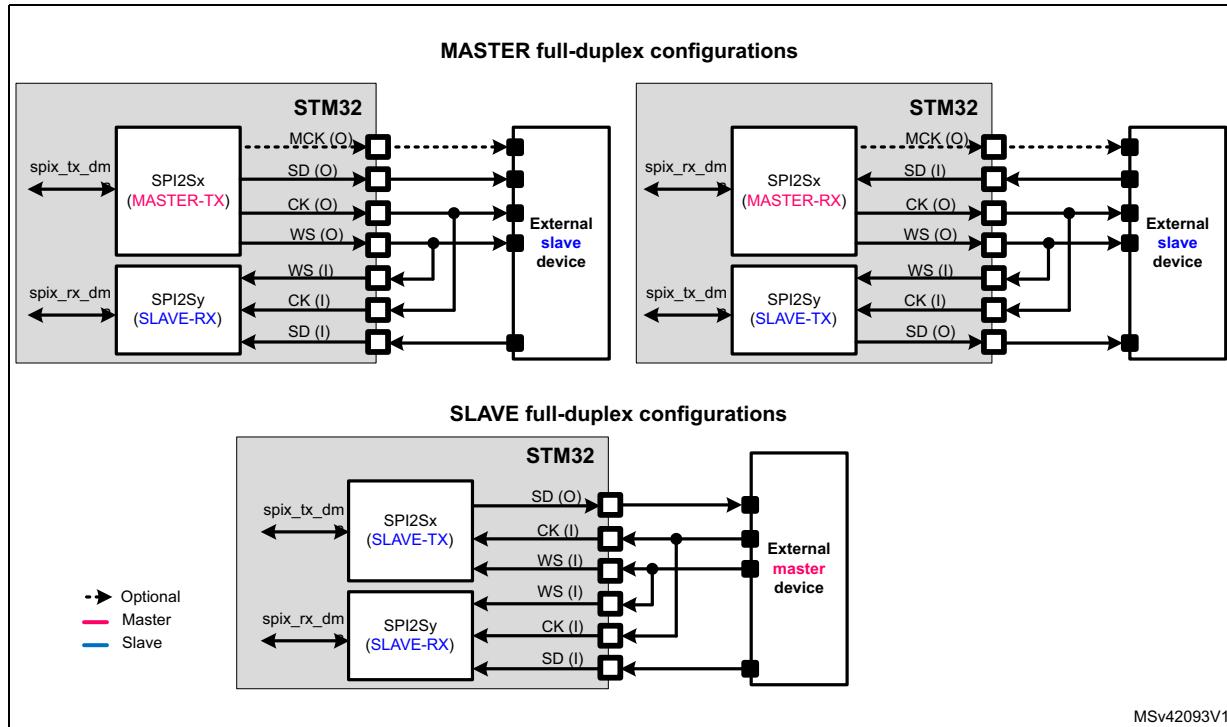
For the master full-duplex mode, one of the SPI2S block must be programmed in master (I2SCFG = '10' or '11'), and the other SPI2S block must be programmed in slave (I2SCFG = '00' or '01'). The MCK can be generated or not, depending on the application needs.

For the slave full-duplex mode, both SPI2S blocks must be programmed in slave. One of them in the slave receiver (I2SCFG = '01'), and the other in the slave transmitter (I2SCFG = '00'). The master external device then provides the bit clock (CK) and the frame synchronization (WS).

Note that the full-duplex mode can be used for all the supported standards: I<sup>2</sup>S Philips, MSB justified, LSB justified and PCM.

For the full-duplex mode, both SPI2S 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 317. Full-duplex communication



### 26.6.3 Supported audio protocols

The three-line bus has to handle only audio data generally time-multiplexed on two channels: the right channel and the left channel. However there is only one 16-bit register for transmission or reception. So, it is up to the software to write into the data register the appropriate value corresponding to each channel side, or to read the data from the data register and to identify the corresponding channel by checking the CHSIDE bit in the SPIx\_SR register. Channel left is always sent first followed by the channel right (CHSIDE has no meaning for the PCM protocol).

Four data and packet frames are available. Data may be sent with a format of:

- 16-bit data packed in a 16-bit frame
- 16-bit data packed in a 32-bit frame
- 24-bit data packed in a 32-bit frame
- 32-bit data packed in a 32-bit frame

When using 16-bit data extended on 32-bit packet, the first 16 bits (MSB) are the significant bits, the 16-bit LSB is forced to 0 without any need for software action or DMA request (only one read/write operation).

The 24-bit and 32-bit data frames need two CPU read or write operations to/from the SPIx\_DR register or two DMA operations if the DMA is preferred for the application. For 24-bit data frame specifically, the 8 non significant bits are extended to 32 bits with 0-bits (by hardware).

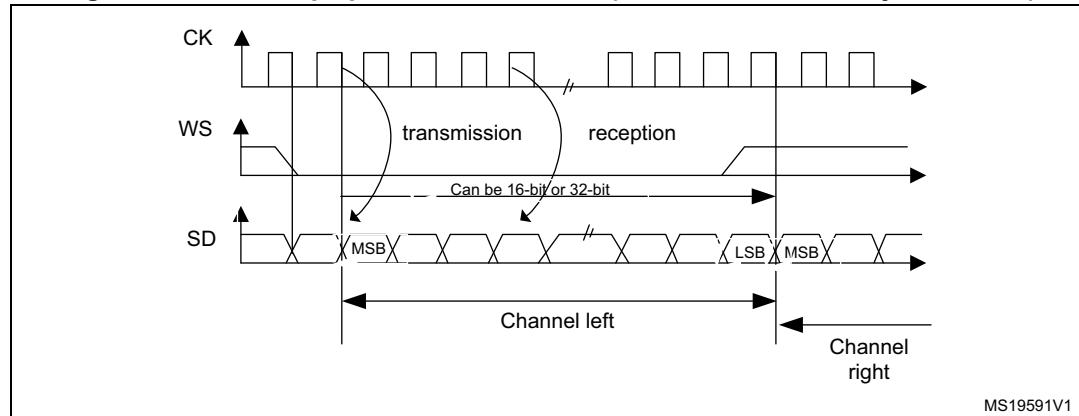
For all data formats and communication standards, the most significant bit is always sent first (MSB first).

The I<sup>2</sup>S interface supports four audio standards, configurable using the I2SSSTD[1:0] and PCMSYNC bits in the SPIx\_I2SCFGR register.

### I<sup>2</sup>S Philips standard

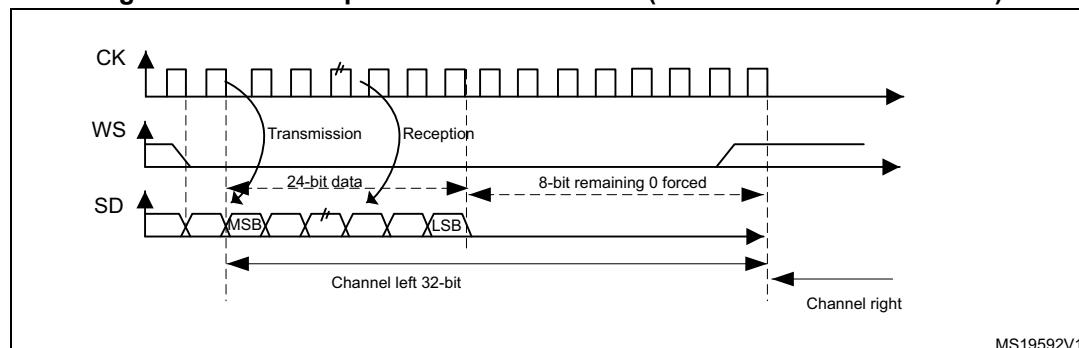
For this standard, the WS signal is used to indicate which channel is being transmitted. It is activated one CK clock cycle before the first bit (MSB) is available.

**Figure 318. I<sup>2</sup>S Philips protocol waveforms (16/32-bit full accuracy, CPOL = 0)**



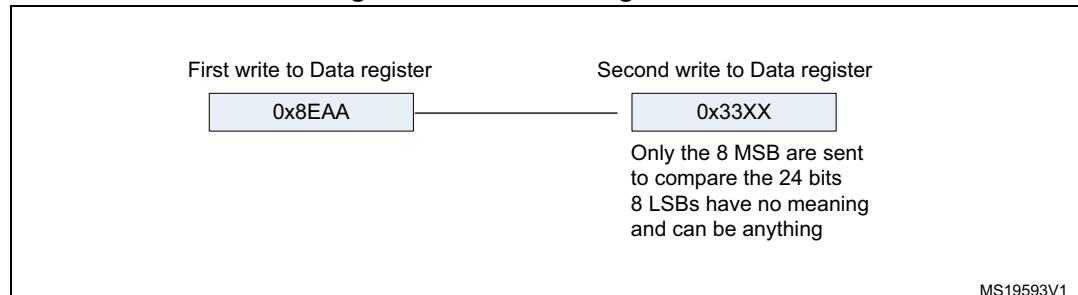
Data are latched on the falling edge of CK (for the transmitter) and are read on the rising edge (for the receiver). The WS signal is also latched on the falling edge of CK.

**Figure 319. I<sup>2</sup>S Philips standard waveforms (24-bit frame with CPOL = 0)**

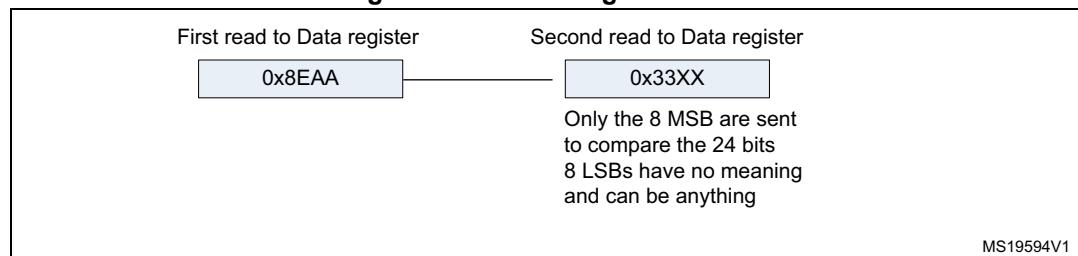
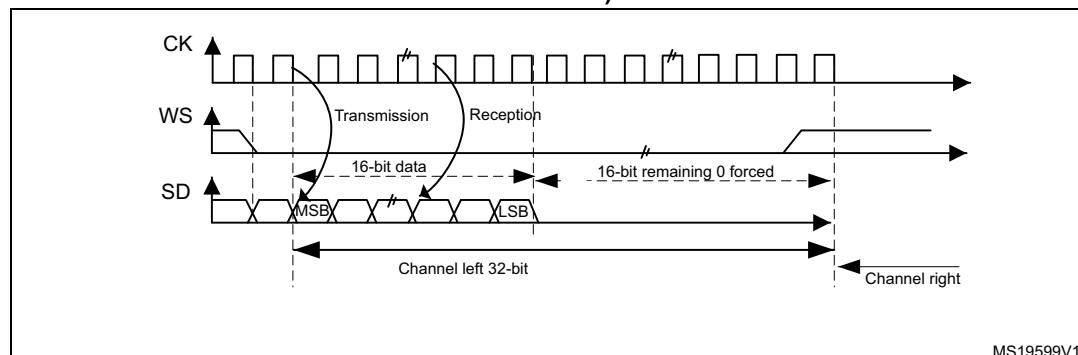


This mode needs two write or read operations to/from the SPIx\_DR register.

- In transmission mode:  
If 0x8EAA33 has to be sent (24-bit):

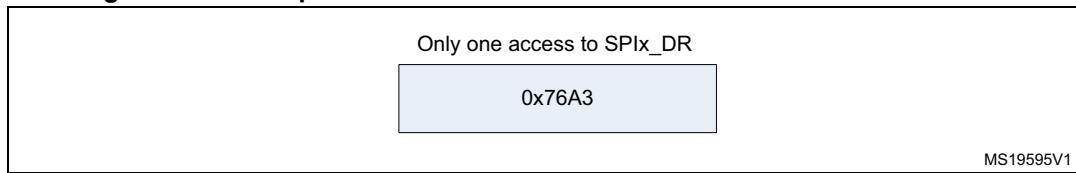
**Figure 320. Transmitting 0x8EAA33**

- In reception mode:  
If data 0x8EAA33 is received:

**Figure 321. Receiving 0x8EAA33****Figure 322. I<sup>2</sup>S Philips standard (16-bit extended to 32-bit packet frame with CPOL = 0)**

When 16-bit data frame extended to 32-bit channel frame is selected during the I<sup>2</sup>S 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 323](#) is required.

**Figure 323. Example of 16-bit data frame extended to 32-bit channel frame**

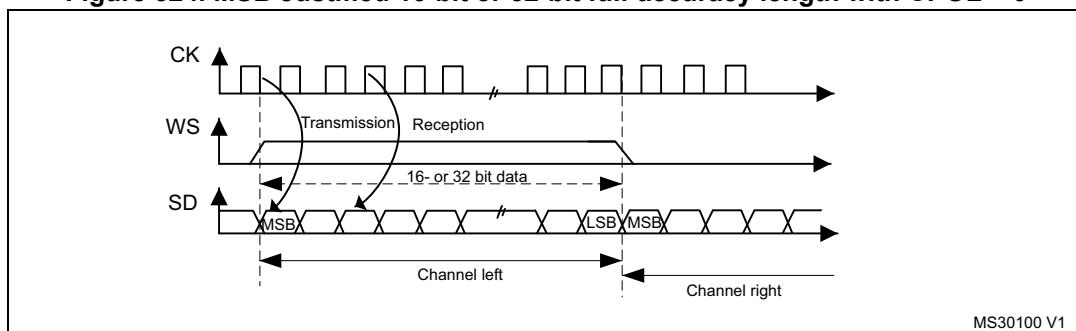
For transmission, each time an MSB is written to SPIx\_DR, the TXE flag is set and its interrupt, if allowed, is generated to load the SPIx\_DR register with the new value to send. This takes place even if 0x0000 have not yet been sent because it is done by hardware.

For reception, the RXNE flag is set and its interrupt, if allowed, is generated when the first 16 MSB half-word is received.

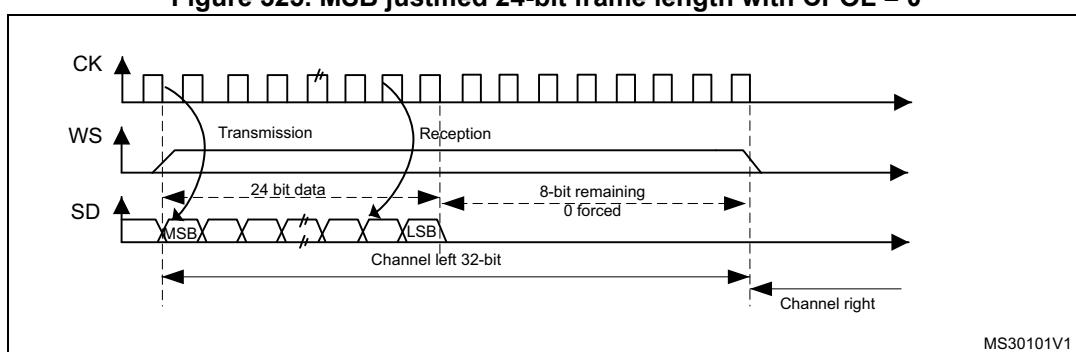
In this way, more time is provided between two write or read operations, which prevents underrun or overrun conditions (depending on the direction of the data transfer).

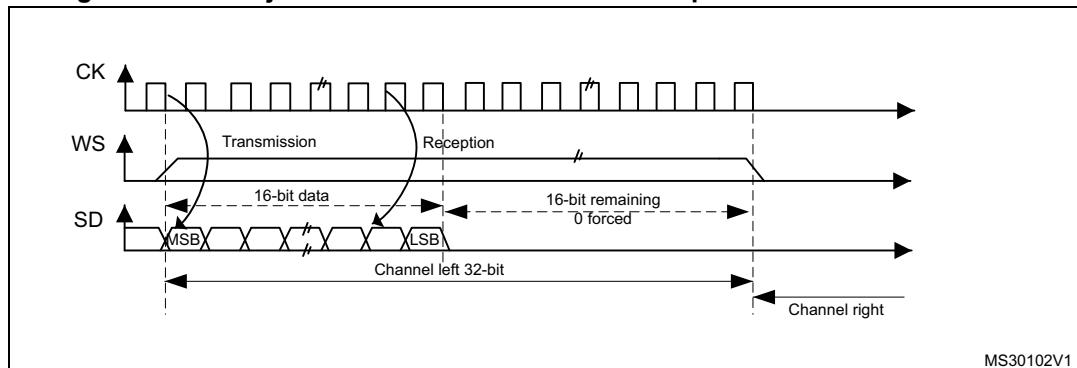
### MSB justified standard

For this standard, the WS signal is generated at the same time as the first data bit, which is the MSBit.

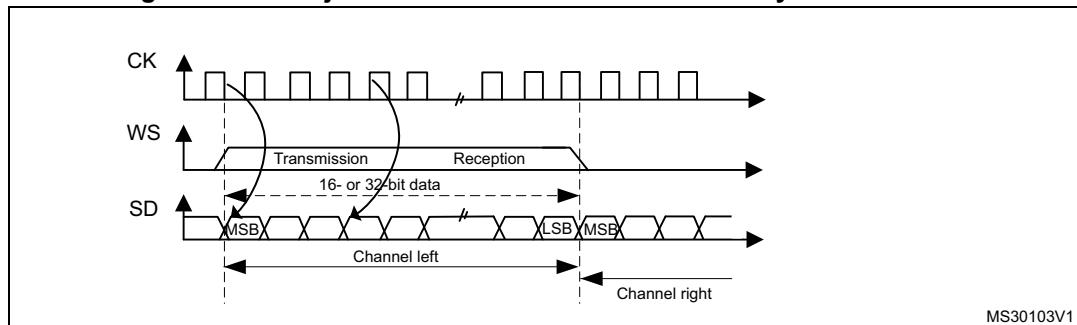
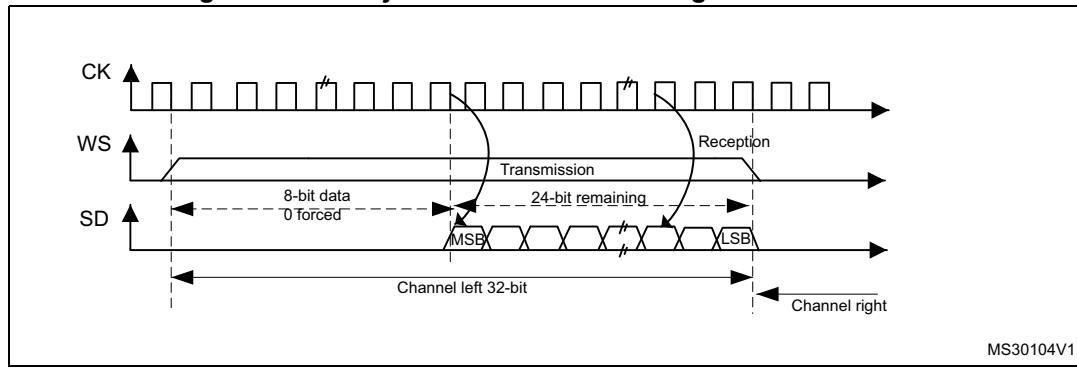
**Figure 324. MSB Justified 16-bit or 32-bit full-accuracy length with CPOL = 0**

Data are latched on the falling edge of CK (for transmitter) and are read on the rising edge (for the receiver).

**Figure 325. MSB justified 24-bit frame length with CPOL = 0**

**Figure 326. MSB justified 16-bit extended to 32-bit packet frame with CPOL = 0****LSB justified standard**

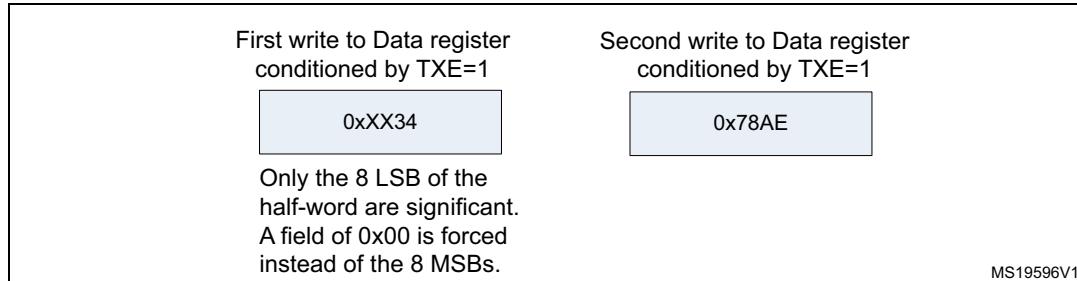
This standard is similar to the MSB justified standard (no difference for the 16-bit and 32-bit full-accuracy frame formats).

**Figure 327. LSB justified 16-bit or 32-bit full-accuracy with CPOL = 0****Figure 328. LSB justified 24-bit frame length with CPOL = 0**

- 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.

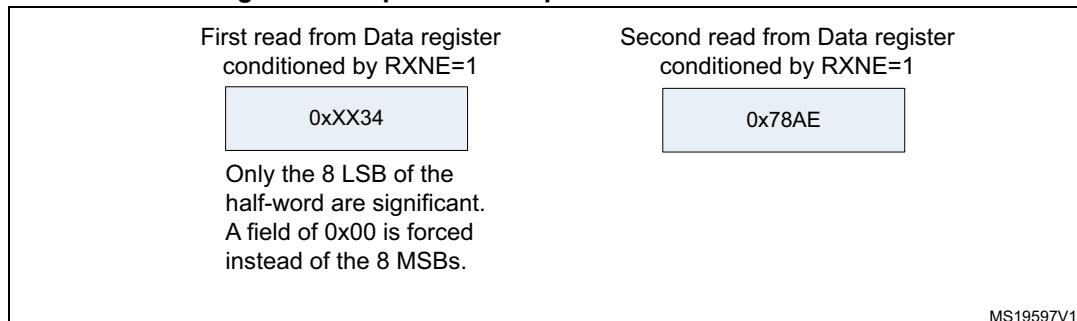
**Figure 329. Operations required to transmit 0x3478AE**



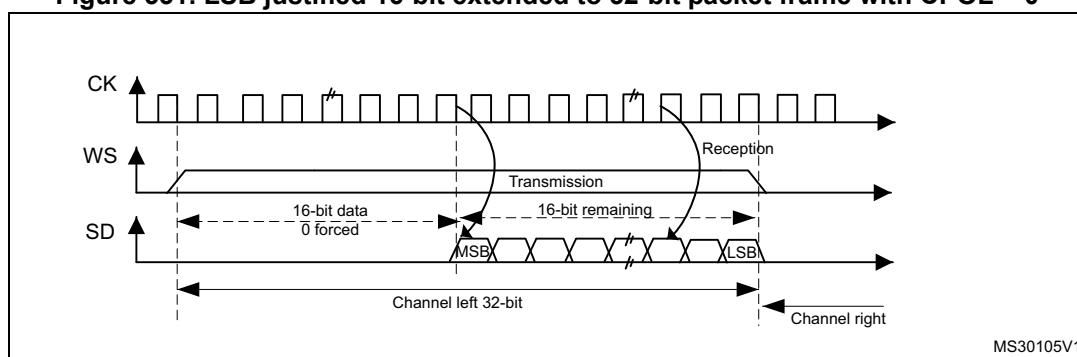
- In reception mode:

If data 0x3478AE are received, two successive read operations from the SPIx\_DR register are required on each RXNE event.

**Figure 330. Operations required to receive 0x3478AE**

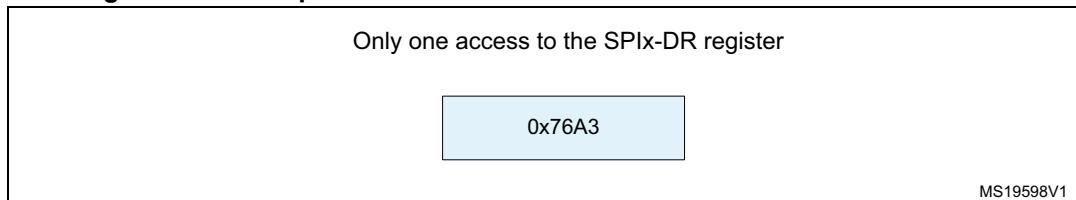


**Figure 331. LSB justified 16-bit extended to 32-bit packet frame with CPOL = 0**



When 16-bit data frame extended to 32-bit channel frame is selected during the I<sup>2</sup>S 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 332](#) is required.

**Figure 332. Example of 16-bit data frame extended to 32-bit channel frame**

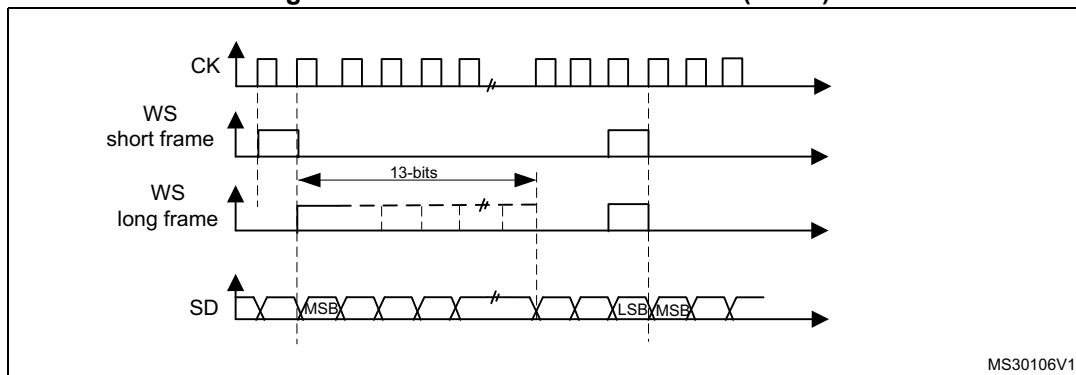
In transmission mode, when a TXE event occurs, the application has to write the data to be transmitted (in this case 0x76A3). The 0x000 field is transmitted first (extension on 32-bit). The TXE flag is set again as soon as the effective data (0x76A3) is sent on SD.

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.

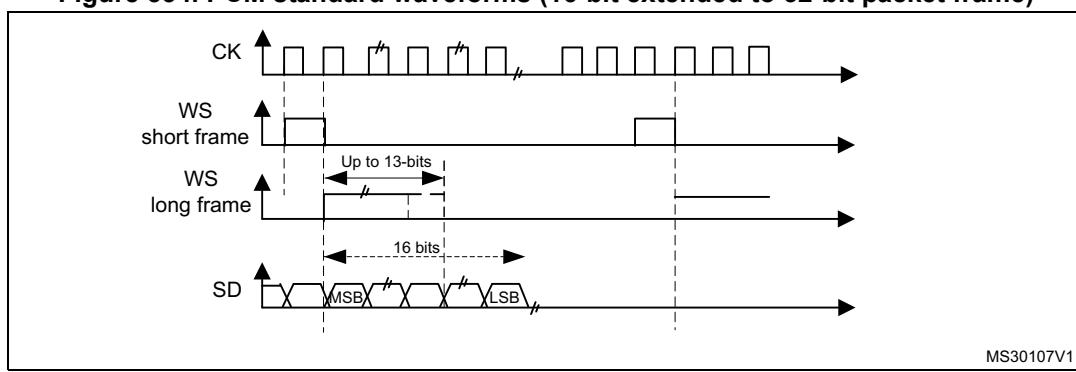
### PCM standard

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.

**Figure 333. 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 334. 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.

#### 26.6.4 Clock generator

The I<sup>2</sup>S bitrate determines the data flow on the I<sup>2</sup>S data line and the I<sup>2</sup>S clock signal frequency.

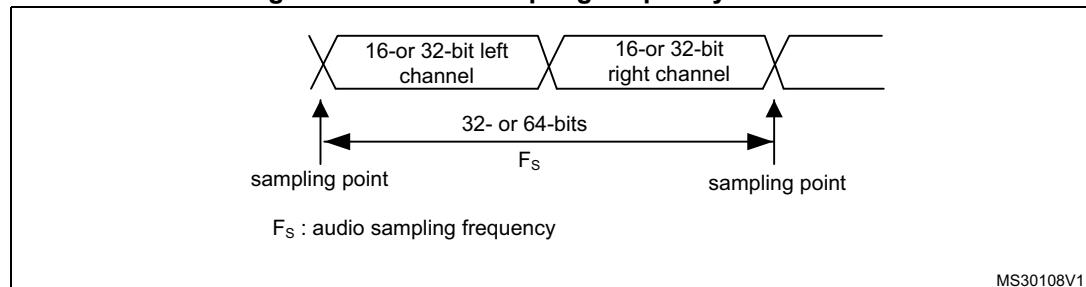
$$\text{I}^2\text{S bitrate} = \text{number of bits per channel} \times \text{number of channels} \times \text{sampling audio frequency}$$

For a 16-bit audio, left and right channel, the I<sup>2</sup>S bitrate is calculated as follows:

$$\text{I}^2\text{S bitrate} = 16 \times 2 \times f_S$$

It will be: I<sup>2</sup>S bitrate = 32 × 2 × f<sub>S</sub> if the packet length is 32-bit wide.

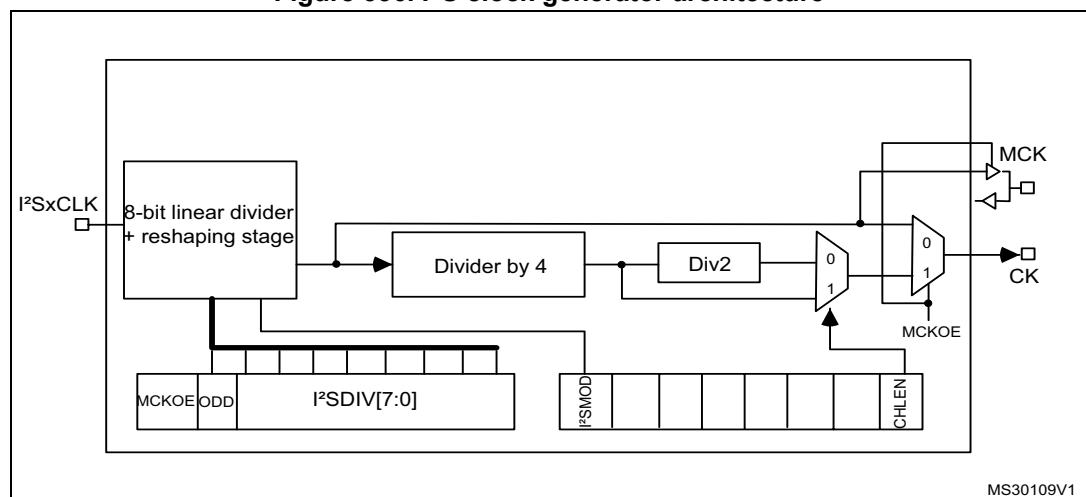
**Figure 335. Audio sampling frequency definition**



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 336](#) presents the communication clock architecture. The I<sup>2</sup>Sx clock is always the system clock.

**Figure 336. I<sup>2</sup>S clock generator architecture**



1. Where x = 2.

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:

When the master clock is generated (MCKOE in the SPIx\_I2SPR register is set):

$$f_S = I2SxCLK / [(16*2)*((2*I2SDIV)+ODD)*8)] \text{ when the channel frame is 16-bit wide}$$

$$f_S = I2SxCLK / [(32*2)*((2*I2SDIV)+ODD)*4)] \text{ when the channel frame is 32-bit wide}$$

When the master clock is disabled (MCKOE bit cleared):

$$f_S = I2SxCLK / [(16*2)*((2*I2SDIV)+ODD)] \text{ when the channel frame is 16-bit wide}$$

$$f_S = I2SxCLK / [(32*2)*((2*I2SDIV)+ODD)] \text{ when the channel frame is 32-bit wide}$$

*Table 161* provides example precision values for different clock configurations.

*Note:* Other configurations are possible that allow optimum clock precision.

**Table 161. Audio-frequency precision using standard 8 MHz HSE<sup>(1)</sup>**

SYCLK (MHz)	Data length	I2SDIV	I2SODD	MCLK	Target f <sub>S</sub> (Hz)	Real f <sub>S</sub> (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%

**Table 161. Audio-frequency precision using standard 8 MHz HSE<sup>(1)</sup> (continued)**

SYSCLK (MHz)	Data length	I2SDIV	I2SODD	MCLK	Target f <sub>S</sub> (Hz)	Real f <sub>S</sub> (KHz)	Error
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.

## 26.6.5 I<sup>2</sup>S master mode

The I<sup>2</sup>S 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 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 26.6.4: Clock generator](#)).
3. Set the I2SMOD bit in the SPIx\_I2SCFGR register to activate the I<sup>2</sup>S functions and choose the I<sup>2</sup>S standard through the I2SSTD[1:0] and PCMSYNC bits, the data length through the DATLEN[1:0] bits and the number of bits per channel by configuring the CHLEN bit. Select also the I<sup>2</sup>S master mode and direction (Transmitter or Receiver) through the I2SCFG[1:0] bits in the SPIx\_I2SCFGR register.
4. If needed, select all the potential interrupt sources and the DMA capabilities by writing the SPIx\_CR2 register.
5. The I2SE bit in SPIx\_I2SCFGR register must be set.

WS and CK are configured in output mode. MCK is also an output, if the MCKOE bit in SPIx\_I2SPR is set.

### Transmission sequence

The transmission sequence begins when a half-word is written into the Tx buffer.

Lets assume the first data written into the Tx buffer corresponds to the left channel data. When data are transferred from the Tx buffer to the shift register, TXE is set and data corresponding to the right channel have to be written into the Tx buffer. The CHSIDE flag

indicates which channel is to be transmitted. It has a meaning when the TXE flag is set because the CHSIDE flag is updated when TXE goes high.

A full frame has to be considered as a left channel data transmission followed by a right channel data transmission. It is not possible to have a partial frame where only the left channel is sent.

The data half-word is parallel loaded into the 16-bit shift register during the first bit transmission, and then shifted out, serially, to the MOSI/SD pin, MSB first. The TXE flag is set after each transfer from the Tx buffer to the shift register and an interrupt is generated if the TXEIE bit in the SPIx\_CR2 register is set.

For more details about the write operations depending on the I<sup>2</sup>S Standard-mode selected, refer to [Section 26.6.3: Supported audio protocols](#)).

To ensure a continuous audio data transmission, it is mandatory to write the SPIx\_DR register with the next data to transmit before the end of the current transmission.

To switch off the I<sup>2</sup>S, by clearing I2SE, it is mandatory to wait for TXE = 1 and BSY = 0.

### Reception sequence

The operating mode is the same as for transmission mode except for the point 3 (refer to the procedure described in [Section 26.6.5: I<sup>2</sup>S master mode](#)), where the configuration should set the master reception mode through the I2SCFG[1:0] bits.

Whatever the data or channel length, the audio data are received by 16-bit packets. This means that each time the Rx buffer is full, the RXNE flag is set and an interrupt is generated if the RXNEIE bit is set in SPIx\_CR2 register. Depending on the data and channel length configuration, the audio value received for a right or left channel may result from one or two receptions into the Rx buffer.

Clearing the RXNE bit is performed by reading the SPIx\_DR register.

CHSIDE is updated after each reception. It is sensitive to the WS signal generated by the I<sup>2</sup>S cell.

For more details about the read operations depending on the I<sup>2</sup>S Standard-mode selected, refer to [Section 26.6.3: Supported audio protocols](#).

If data are received while the previously received data have not been read yet, an overrun is generated and the OVR flag is set. If the ERRIE bit is set in the SPIx\_CR2 register, an interrupt is generated to indicate the error.

To switch off the I<sup>2</sup>S, specific actions are required to ensure that the I<sup>2</sup>S completes the transfer cycle properly without initiating a new data transfer. The sequence depends on the configuration of the data and channel lengths, and on the audio protocol mode selected. In the case of:

- 16-bit data length extended on 32-bit channel length (DATLEN = 00 and CHLEN = 1) using the LSB justified mode (I2SSTD = 10)
  - a) Wait for the second to last RXNE = 1 ( $n - 1$ )
  - b) Then wait 17 I<sup>2</sup>S clock cycles (using a software loop)
  - c) Disable the I<sup>2</sup>S (I2SE = 0)
- 16-bit data length extended on 32-bit channel length (DATLEN = 00 and CHLEN = 1) in MSB justified, I<sup>2</sup>S or PCM modes (I2SSTD = 00, I2SSTD = 01 or I2SSTD = 11, respectively)
  - a) Wait for the last RXNE

- b) Then wait 1 I<sup>2</sup>S clock cycle (using a software loop)
- c) Disable the I<sup>2</sup>S (I2SE = 0)
- For all other combinations of DATLEN and CHLEN, whatever the audio mode selected through the I2SSTD bits, carry out the following sequence to switch off the I<sup>2</sup>S:
  - a) Wait for the second to last RXNE = 1 (n – 1)
  - b) Then wait one I<sup>2</sup>S clock cycle (using a software loop)
  - c) Disable the I<sup>2</sup>S (I2SE = 0)

**Note:** *The BSY flag is kept low during transfers.*

## 26.6.6 I<sup>2</sup>S slave mode

For the slave configuration, the I<sup>2</sup>S can be configured in transmission or reception mode.

The operating mode is following mainly the same rules as described for the I<sup>2</sup>S master configuration. In slave mode, there is no clock to be generated by the I<sup>2</sup>S interface. The clock and WS signals are input from the external master connected to the I<sup>2</sup>S interface. There is then no need, for the user, to configure the clock.

The configuration steps to follow are listed below:

1. Set the I2SMOD bit in the SPIx\_I2SCFGR register to select I<sup>2</sup>S mode and choose the I<sup>2</sup>S standard through the I2SSTD[1:0] bits, the data length through the DATLEN[1:0] bits and the number of bits per channel for the frame configuring the CHLEN bit. Select also the mode (transmission or reception) for the slave through the I2SCFG[1:0] bits in SPIx\_I2SCFGR register.
2. If needed, select all the potential interrupt sources and the DMA capabilities by writing the SPIx\_CR2 register.
3. The I2SE bit in SPIx\_I2SCFGR register must be set.

### Transmission sequence

The transmission sequence begins when the external master device sends the clock and when the NSS\_WS signal requests the transfer of data. The slave has to be enabled before the external master starts the communication. The I<sup>2</sup>S data register has to be loaded before the master initiates the communication.

For the I<sup>2</sup>S, 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 I<sup>2</sup>S data register.

The CHSIDE flag indicates which channel is to be transmitted. Compared to the master transmission mode, in slave mode, CHSIDE is sensitive to the WS signal coming from the external master. This means that the slave needs to be ready to transmit the first data before the clock is generated by the master. WS assertion corresponds to left channel transmitted first.

**Note:** *The I2SE has to be written at least two PCLK cycles before the first clock of the master comes on the CK line.*

The data half-word is parallel-loaded into the 16-bit shift register (from the internal bus) during the first bit transmission, and then shifted out serially to the MOSI/SD pin MSB first. The TXE flag is set after each transfer from the Tx buffer to the shift register and an interrupt is generated if the TXEIE bit in the SPIx\_CR2 register is set.

Note that the TXE flag should be checked to be at 1 before attempting to write the Tx buffer.

For more details about the write operations depending on the I<sup>2</sup>S Standard-mode selected, refer to [Section 26.6.3: 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 I<sup>2</sup>S and to restart a data transfer starting from the left channel.

To switch off the I<sup>2</sup>S, 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 26.6.6: 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 I<sup>2</sup>S Standard-mode selected, refer to [Section 26.6.3: Supported audio protocols](#).

If data are received while the preceding received data have not yet been read, an overrun is generated and the OVR flag is set. If the bit ERRIE is set in the SPIx\_CR2 register, an interrupt is generated to indicate the error.

To switch off the I<sup>2</sup>S in reception mode, I2SE has to be cleared immediately after receiving the last RXNE = 1.

**Note:** *The external master components should have the capability of sending/receiving data in 16-bit or 32-bit packets via an audio channel.*

## 26.6.7 I<sup>2</sup>S status flags

Three status flags are provided for the application to fully monitor the state of the I<sup>2</sup>S bus.

### Busy flag (BSY)

The BSY flag is set and cleared by hardware (writing to this flag has no effect). It indicates the state of the communication layer of the I<sup>2</sup>S.

When BSY is set, it indicates that the I<sup>2</sup>S is busy communicating. There is one exception in master receive mode (I2SCFG = 11) where the BSY flag is kept low during reception.

The BSY flag is useful to detect the end of a transfer if the software needs to disable the I<sup>2</sup>S. This avoids corrupting the last transfer. For this, the procedure described below must be strictly respected.

The BSY flag is set when a transfer starts, except when the I<sup>2</sup>S is in master receiver mode.

The BSY flag is cleared:

- When a transfer completes (except in master transmit mode, in which the communication is supposed to be continuous)
- When the I<sup>2</sup>S is disabled

When communication is continuous:

- In master transmit mode, the BSY flag is kept high during all the transfers
- In slave mode, the BSY flag goes low for one I<sup>2</sup>S clock cycle between each transfer

**Note:** *Do not use the BSY flag to handle each data transmission or reception. It is better to use the TXE and RXNE flags instead.*

### Tx buffer empty flag (TXE)

When set, this flag indicates that the Tx buffer is empty and the next data to be transmitted can then be loaded into it. The TXE flag is reset when the Tx buffer already contains data to be transmitted. It is also reset when the I<sup>2</sup>S is disabled (I<sup>2</sup>SE 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 I<sup>2</sup>S 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 I<sup>2</sup>S 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).

## 26.6.8 I<sup>2</sup>S error flags

There are three error flags for the I<sup>2</sup>S 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 I<sup>2</sup>S is configured in Slave mode. It is set if the external master is changing the WS line while the slave is not expecting this change. If the synchronization is lost, the following steps are required to recover from this state and resynchronize the external master device with the I<sup>2</sup>S slave device:

1. Disable the I<sup>2</sup>S.
2. Enable it again when the correct level is detected on the WS line (WS line is high in I<sup>2</sup>S mode or low for MSB- or LSB-justified or PCM modes).

Desynchronization between master and slave devices may be due to noisy environment on the SCK 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.

## 26.6.9 I<sup>2</sup>S interrupts

*Table 162* provides the list of I<sup>2</sup>S interrupts.

**Table 162. I<sup>2</sup>S interrupt requests**

Interrupt event	Event flag	Enable control bit
Transmit buffer empty flag	TXE	TXEIE
Receive buffer not empty flag	RXNE	RXNEIE
Overrun error	OVR	ERRIE
Underrun error	UDR	
Frame error flag	FRE	

## 26.6.10 DMA features

In I<sup>2</sup>S mode, the DMA works in exactly the same way as it does in SPI mode. There is no difference except that the CRC feature is not available in I<sup>2</sup>S mode since there is no data transfer protection system.

## 26.7 SPI and I<sup>2</sup>S registers

The peripheral registers can be accessed by half-words (16-bit) or words (32-bit). SPI\_DR in addition by can be accessed by 8-bit access.

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

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

### 26.7.1 SPI control register 1 (SPI\_CR1) (not used in I<sup>2</sup>S mode)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIDI MODE	BIDI OE	CRC EN	CRC NEXT	DFF	RX ONLY	SSM	SSI	LSB FIRST	SPE	BR [2:0]			MSTR	CPOL	CPHA

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 I<sup>2</sup>S 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: This bit is not used in I<sup>2</sup>S mode.*

*In master mode, the MOSI pin is used while the MISO pin is used in slave 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.  
It is not used in I<sup>2</sup>S mode.*

Bit 12 **CRCNEXT**: CRC transfer next

- 0: Data phase (no CRC phase)
- 1: Next transfer is CRC (CRC phase)

*Note: When the SPI is configured in full-duplex or transmitter only modes, CRCNEXT must be written as soon as the last data is written to the SPI\_DR register.*

*When the SPI is configured in receiver only mode, CRCNEXT must be set after the second last data reception.*

*This bit should be kept cleared when the transfers are managed by DMA.*

*It is not used in I<sup>2</sup>S mode.*

Bit 11 **DFF**: Data frame format

- 0: 8-bit data frame format is selected for transmission/reception
- 1: 16-bit data frame format is selected for transmission/reception

*Note:* This bit should be written only when SPI is disabled (**SPE** = '0') for correct operation.  
**It is not used in I<sup>2</sup>S mode.**

Bit 10 **RXONLY**: Receive only mode enable

- This bit enables simplex communication using a single unidirectional line to receive data exclusively. Keep **BIDIMODE** bit clear when receive only mode is active.
- This bit is also useful in a multislave system in which this particular slave is not accessed, the output from the accessed slave is not corrupted.
- 0: full-duplex (Transmit and receive)
  - 1: Output disabled (Receive-only mode)

*Note:* **This bit is not used in I<sup>2</sup>S mode**

Bit 9 **SSM**: Software slave management

- When the SSM bit is set, the NSS pin input is replaced with the value from the SSI bit.
- 0: Software slave management disabled
  - 1: Software slave management enabled

*Note:* **This bit is not used in I<sup>2</sup>S mode and SPI TI mode**

Bit 8 **SSI**: Internal slave select

- This bit has an effect only when the SSM bit is set. The value of this bit is forced onto the NSS pin and the IO value of the NSS pin is ignored.

*Note:* **This bit is not used in I<sup>2</sup>S mode and SPI TI mode**

Bit 7 **LSBFIRST**: Frame format

- 0: MSB transmitted first
- 1: LSB transmitted first

*Note:* This bit should not be changed when communication is ongoing.

**It is not used in I<sup>2</sup>S mode and SPI TI mode**

Bit 6 **SPE**: SPI enable

- 0: Peripheral disabled
- 1: Peripheral enabled

*Note:* **This bit is not used in I<sup>2</sup>S mode.**

When disabling the SPI, follow the procedure described in [Section 26.3.10: Procedure for disabling the SPI](#).

Bits 5:3 **BR[2:0]**: Baud rate control

- 000: f<sub>PCLK</sub>/2
- 001: f<sub>PCLK</sub>/4
- 010: f<sub>PCLK</sub>/8
- 011: f<sub>PCLK</sub>/16
- 100: f<sub>PCLK</sub>/32
- 101: f<sub>PCLK</sub>/64
- 110: f<sub>PCLK</sub>/128
- 111: f<sub>PCLK</sub>/256

*Note:* These bits should not be changed when communication is ongoing.

**They are not used in I<sup>2</sup>S mode.**

Bit 2 **MSTR**: Master selection

- 0: Slave configuration
- 1: Master configuration

*Note: This bit should not be changed when communication is ongoing.*

*It is not used in I<sup>2</sup>S mode.*

Bit1 **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.*

*It is not used in I<sup>2</sup>S mode and SPI TI mode except the case when CRC is applied at TI mode.*

Bit 0 **CPHA**: Clock phase

- 0: The first clock transition is the first data capture edge
- 1: The second clock transition is the first data capture edge

*Note: This bit should not be changed when communication is ongoing.*

*It is not used in I<sup>2</sup>S mode and SPI TI mode except the case when CRC is applied at TI mode.*

## 26.7.2 SPI control register 2 (SPI\_CR2)

Address offset: 0x04

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TXEIE	RXNEIE	ERRIE	FRF	Res.	SSOE	TXDMAEN	RXDMAEN							
								rw	rw	rw	rw		rw	rw	rw

Bits 15:8 Reserved, must be kept at reset value.

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 (OVR, CRCERR, MODF, FRE in SPI mode, and UDR, OVR, FRE in I<sup>2</sup>S mode).

- 0: Error interrupt is masked
- 1: Error interrupt is enabled

Bit 4 **FRF**: Frame format

- 0: SPI Motorola mode
- 1 SPI TI mode

*Note: This bit is not used in I<sup>2</sup>S mode.*

Bit 3 Reserved. Forced to 0 by hardware.

Bit 2 **SSOE:** SS output enable

- 0: SS output is disabled in master mode and the cell can work in multimaster configuration
- 1: SS output is enabled in master mode and when the cell is enabled. The cell cannot work in a multimaster environment.

*Note: This bit is not used in I<sup>2</sup>S mode and SPI TI mode.*

Bit 1 **TXDMAEN:** Tx buffer DMA enable

When this bit is set, the DMA request is made whenever the TXE flag is set.

- 0: Tx buffer DMA disabled
- 1: Tx buffer DMA enabled

Bit 0 **RXDMAEN:** Rx buffer DMA enable

When this bit is set, the DMA request is made whenever the RXNE flag is set.

- 0: Rx buffer DMA disabled
- 1: Rx buffer DMA enabled

### 26.7.3 SPI status register (SPI\_SR)

Address offset: 0x08

Reset value: 0x0002

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	FRE	BSY	OVR	MODEF	CRC ERR	UDR	CHSIDE	TXE	RXNE						

Bits 15:9 Reserved. Forced to 0 by hardware.

Bit 8 **FRE:** Frame Error

- 0: No frame error
- 1: Frame error occurred.

This bit is set by hardware and cleared by software when the SPI\_SR register is read.

This bit is used in SPI TI mode or in I2S mode whatever the audio protocol selected. It detects a change on NSS or WS line which takes place in slave mode at a non expected time, informing about a desynchronization between the external master device and the slave.

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: BSY flag must be used with caution: refer to [Section 26.3.12: SPI status flags](#) and [Section 26.3.10: Procedure for disabling the SPI](#).*

Bit 6 **OVR:** Overrun flag

- 0: No overrun occurred
- 1: Overrun occurred

This flag is set by hardware and reset by a software sequence. Refer to [Section 26.3.13: SPI error flags](#) 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 26.4 on page 865](#) for the software sequence.

*Note: This bit is not used in I<sup>2</sup>S mode*

Bit 4 **CRCERR:** CRC error flag

- 0: CRC value received matches the SPI\_RXCRCR value
  - 1: CRC value received does not match the SPI\_RXCRCR value
- This flag is set by hardware and cleared by software writing 0.

*Note: This bit is not used in I<sup>2</sup>S mode.*

Bit 3 **UDR:** Underrun flag

- 0: No underrun occurred
- 1: Underrun occurred

This flag is set by hardware and reset by a software sequence. Refer to [Section 26.6.8: I2S error flags](#) 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 for SPI mode and is meaningless 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

### 26.7.4 SPI data register (SPI\_DR)

Address offset: 0x0C

Reset value: 0x0000

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

Bits 15:0 **DR[15:0]**: Data register

Data received or to be transmitted.

The data register is split into 2 buffers - one for writing (Transmit Buffer) and another one for reading (Receive buffer). A write to the data register will write into the Tx buffer and a read from the data register will return the value held in the Rx buffer.

*Note: These notes apply to SPI mode:*

*Depending on the data frame format selection bit (DFF in SPI\_CR1 register), the data sent or received is either 8-bit or 16-bit. This selection has to be made before enabling the SPI to ensure correct operation.*

*For an 8-bit data frame, the buffers are 8-bit and only the LSB of the register (SPI\_DR[7:0]) is used for transmission/reception. When in reception mode, the MSB of the register (SPI\_DR[15:8]) is forced to 0.*

*For a 16-bit data frame, the buffers are 16-bit and the entire register, SPI\_DR[15:0] is used for transmission/reception.*

### 26.7.5 SPI CRC polynomial register (SPI\_CRCPR) (not used in I<sup>2</sup>S mode)

Address offset: 0x10

Reset value: 0x0007

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

Bits 15:0 **CRCPOLY[15:0]**: CRC polynomial register

This register contains the polynomial for the CRC calculation.

The CRC polynomial (0007h) is the reset value of this register. Another polynomial can be configured as required.

*Note: These bits are not used for the I<sup>2</sup>S mode.*

## 26.7.6 SPI RX CRC register (SPI\_RXCRCR) (not used in I<sup>2</sup>S mode)

Address offset: 0x14

Reset value: 0x0000

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

Bits 15:0 **RXCRC[15:0]**: Rx CRC register

When CRC calculation is enabled, the RxCRC[15:0] bits contain the computed CRC value of the subsequently received bytes. This register is reset when the CRCEN bit in SPI\_CR1 register is written to 1. The CRC is calculated serially using the polynomial programmed in the SPI\_CRCPR register.

Only the 8 LSB bits are considered when the data frame format is set to be 8-bit data (DFF bit of SPI\_CR1 is cleared). CRC calculation is done based on any CRC8 standard.

The entire 16-bits of this register are considered when a 16-bit data frame format is selected (DFF bit of the SPI\_CR1 register is set). CRC calculation is done based on 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 for I<sup>2</sup>S mode.*

## 26.7.7 SPI TX CRC register (SPI\_TXCRCR) (not used in I<sup>2</sup>S mode)

Address offset: 0x18

Reset value: 0x0000

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

Bits 15:0 **TXCRC[15:0]**: Tx CRC register

When CRC calculation is enabled, the TxCRC[7:0] bits contain the computed CRC value of the subsequently transmitted bytes. This register is reset when the CRCEN bit of SPI\_CR1 is written to 1. The CRC is calculated serially using the polynomial programmed in the SPI\_CRCPR register.

Only the 8 LSB bits are considered when the data frame format is set to be 8-bit data (DFF bit of SPI\_CR1 is cleared). CRC calculation is done based on any CRC8 standard.

The entire 16-bits of this register are considered when a 16-bit data frame format is selected (DFF bit of the SPI\_CR1 register is set). CRC calculation is done based on 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 for I<sup>2</sup>S mode.*

## 26.7.8 SPI\_I<sup>2</sup>S configuration register (SPI\_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.	ASTREN	I2SMOD	I2SE	I2SCFG		PCMSYNC	Res.	I2SSTD		CKPOL	DATLEN		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 **ASTREN**: Asynchronous start enable.

0: The Asynchronous start is disabled. When the I<sup>2</sup>S is enabled in slave mode, the I<sup>2</sup>S slave starts the transfer when the I<sup>2</sup>S clock is received and an appropriate transition (depending on the protocol selected) is detected on the WS signal.

1: The Asynchronous start is enabled. When the I<sup>2</sup>S is enabled in slave mode, the I<sup>2</sup>S slave starts immediately the transfer when the I<sup>2</sup>S clock is received from the master without checking the expected transition of WS signal.

*Note: Note: The appropriate transition is a falling edge on WS signal when I<sup>2</sup>S Philips Standard is used, or a rising edge for other standards.*

Bit 11 **I2SMOD**: I<sup>2</sup>S mode selection

- 0: SPI mode is selected
- 1: I<sup>2</sup>S mode is selected

*Note: This bit should be configured when the SPI or I<sup>2</sup>S is disabled*

Bit 10 **I2SE**: I<sup>2</sup>S Enable

- 0: I<sup>2</sup>S peripheral is disabled
- 1: I<sup>2</sup>S peripheral is enabled

*Note: This bit is not used in SPI mode.*

Bits 9:8 **I2SCFG**: I<sup>2</sup>S configuration mode

- 00: Slave - transmit
- 01: Slave - receive
- 10: Master - transmit
- 11: Master - receive

*Note: This bit should be configured when the I<sup>2</sup>S is disabled.*

*It is 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: forced at 0 by hardware

Bits 5:4 **I2SSSTD**: I<sup>2</sup>S standard selection

- 00: I<sup>2</sup>S Philips standard.
- 01: MSB justified standard (left justified)
- 10: LSB justified standard (right justified)
- 11: PCM standard

For more details on I<sup>2</sup>S standards, refer to [Section 26.6.3 on page 871](#). Not used in SPI mode.

*Note:* For correct operation, these bits should be configured when the I<sup>2</sup>S is disabled.

Bit 3 **CKPOL**: Steady state clock polarity

- 0: I<sup>2</sup>S clock steady state is low level
- 1: I<sup>2</sup>S clock steady state is high level

*Note:* For correct operation, this bit should be configured when the I<sup>2</sup>S is disabled.

*This bit is not used in SPI mode*

Bits 2:1 **DATLEN**: 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 I<sup>2</sup>S is disabled.

*This bit is 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. Not used in SPI mode.

*Note:* For correct operation, this bit should be configured when the I<sup>2</sup>S is disabled.

## 26.7.9 SPI\_I<sup>2</sup>S prescaler register (SPI\_I2SPR)

Address offset: 0x20

Reset value: 0000 0010 (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
							rw	rw							rw

Bits 15:10 Reserved, must be kept at reset value.

Bit 9 **MCKOE**: Master clock output enable

- 0: Master clock output is disabled
- 1: Master clock output is enabled

*Note:* This bit should be configured when the I<sup>2</sup>S is disabled. It is used only when the I<sup>2</sup>S is in master mode.

*This bit 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 26.6.4 on page 878](#). Not used in SPI mode.

*Note:* This bit should be configured when the I<sup>2</sup>S is disabled. It is used only when the I<sup>2</sup>S is in master mode.

Bits 7:0 **I2SDIV**: I2S Linear prescaler

I2SDIV [7:0] = 0 or I2SDIV [7:0] = 1 are forbidden values.

Refer to [Section 26.6.4 on page 878](#). Not used in SPI mode.

*Note:* These bits should be configured when the I<sup>2</sup>S is disabled. It is used only when the I<sup>2</sup>S is in master mode.

### 26.7.10 SPI register map

The table provides shows the SPI register map and reset values.

**Table 163. SPI register map and reset values**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10
0x00	<b>SPI_CR1</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	0	0	0	0	0	0
		Reset value																					
0x04	<b>SPI_CR2</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	0	0	0	0	0	0
		Reset value																					
0x08	<b>SPI_SR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	0	0	0	0	0	0
		Reset value																					
0x0C	<b>SPI_DR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DR[15:0]						
		Reset value															0	0	0	0	0	0	0
0x10	<b>SPI_CRCPR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CRCPOLY[15:0]						
		Reset value															0	0	0	0	0	0	0
0x14	<b>SPI_RXCRCR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RxCRC[15:0]						
		Reset value															0	0	0	0	0	0	0
0x18	<b>SPI_TXCRCR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TxCRC[15:0]						
		Reset value															0	0	0	0	0	0	0
0x1C	<b>SPI_I2SCFGR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	I2SDIV						
		Reset value															MCKOE	0	0	0	0	0	0
0x20	<b>SPI_I2SPR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	I2SDIV	ODD	0	0	0	0	0
		Reset value															PCM SYNC	0	0	0	0	0	0

Refer to [Section 2.2.2 on page 56](#) for the register boundary addresses.

## 27 SPDIF receiver interface (SPDIFRX)

### 27.1 SPDIFRX interface introduction

The SPDIFRX interface handles S/PDIF audio protocol.

### 27.2 SPDIFRX main features

- Up to 4 inputs available
- Automatic symbol rate detection
- Maximum symbol rate: 12.288 MHz
- Stereo stream from 8 to 192 kHz supported
- Supports Audio IEC-60958 and IEC-61937, consumer applications
- SOPDs B, M and W insertion inside S/PDIF flow
- Parity bit management
- Communication using DMA for audio samples
- Communication using DMA for control and user channel information
- Interrupt capabilities

### 27.3 SPDIFRX functional description

The SPDIFRX peripheral, is designed to receive an S/PDIF flow compliant with IEC-60958 and IEC-61937. These standards support simple stereo streams up to high sample rate, and compressed multi-channel surround sound, such as those defined by Dolby or DTS.

The receiver provides all the necessary features to detect the symbol rate, and decode the incoming data. It is possible to use a dedicated path for the user and channel information in order to ease the interface handling. *Figure 337* shows a simplified block diagram.

The SPDIFRX\_DC block is responsible of the decoding of the S/PDIF stream received from SPDIFRX\_IN[4:1] inputs. This block re-sample the incoming signal, decode the manchester stream, recognize frames, sub-frames and blocks elements. It delivers to the REG\_IF part, decoded data, and associated status flags.

This peripheral can be fully controlled via the APB1 bus, and can handle two DMA channels:

- A DMA channel dedicated to the transfer of audio samples
- A DMA channel dedicated to the transfer of IEC60958 channel status and user information

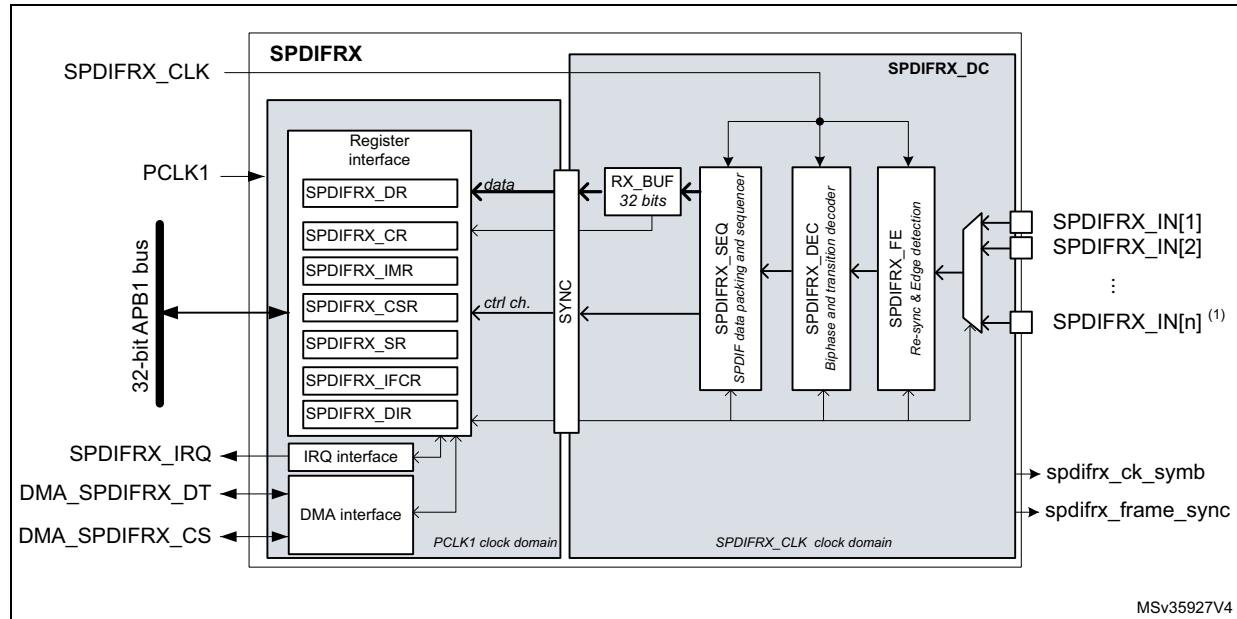
Interrupt services are also available either as an alternative function to the DMA, or for signaling error or key status of the peripheral.

The SPDIFRX also offers a signal named **spdifrx\_frame\_sync**, which toggles every time that a sub-frame's preamble is detected. So the duty cycle will be 50%, and the frequency equal to the frame rate.

This signal can be connected to timer events, in order to compute frequency drift.

In addition the SPDIFRX also provides a signal named **spdifrx\_symb\_ck** toggling at the symbol rate.

Figure 337. SPDIFRX block diagram



1. 'n' is fixed to 4.

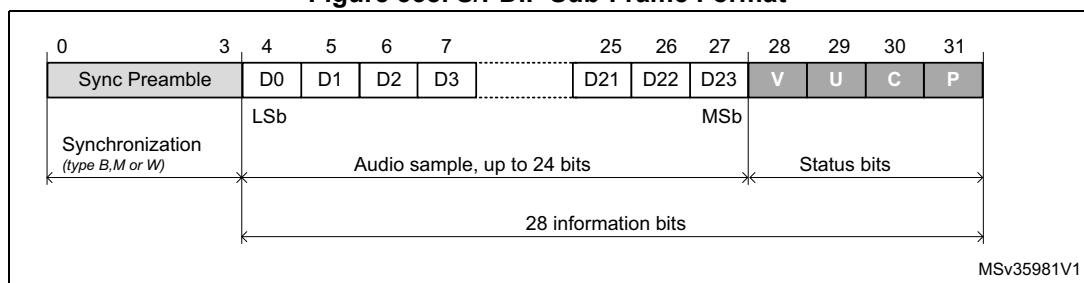
### 27.3.1 S/PDIF protocol (IEC-60958)

#### S/PDIF block

A S/PDIF frame is composed of two sub-frames (see [Figure 339](#)). Each sub-frame contains 32 bits (or time slots):

- Bits 0 to 3 carry one of the synchronization preambles
- Bits 4 to 27 carry the audio sample word in linear 2's complement representation. The most significant bit (MSB) is carried by bit 27. When a 20-bit coding range is used, bits 8 to 27 carry the audio sample word with the LSB in bit 8.
- Bit 28 (validity bit "V") indicates if the data is valid (for converting it to analog for example)
- Bit 29 (user data bit "U") carries the user data information like the number of tracks of a Compact Disk.
- Bit 30 (channel status bit "C") carries the channel status information like sample rate and protection against copy.
- Bit 31 (parity bit "P") carries a parity bit such that bits 4 to 31 inclusive carry an even number of ones and an even number of zeroes (even parity).

Figure 338. S/PDIF Sub-Frame Format

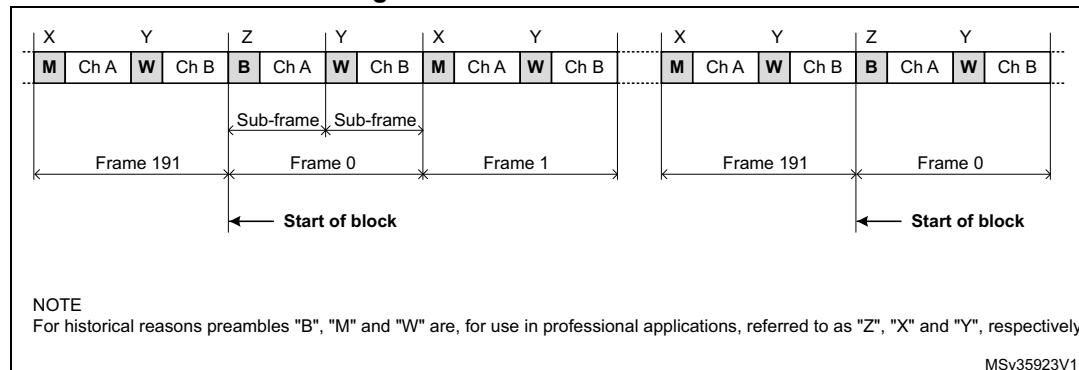


MSv35981V1

For linear coded audio applications, the first sub-frame (left or "A" channel in stereophonic operation and primary channel in monophonic operation) normally starts with preamble "M". However, the preamble changes to preamble "B" once every 192 frames to identify the start of the block structure used to organize the channel status and user information. The second sub-frame (right or "B" channel in stereophonic operation and secondary channel in monophonic operation) always starts with preamble "W".

A S/PDIF block contains 192 pairs of sub-frames of 32 bits.

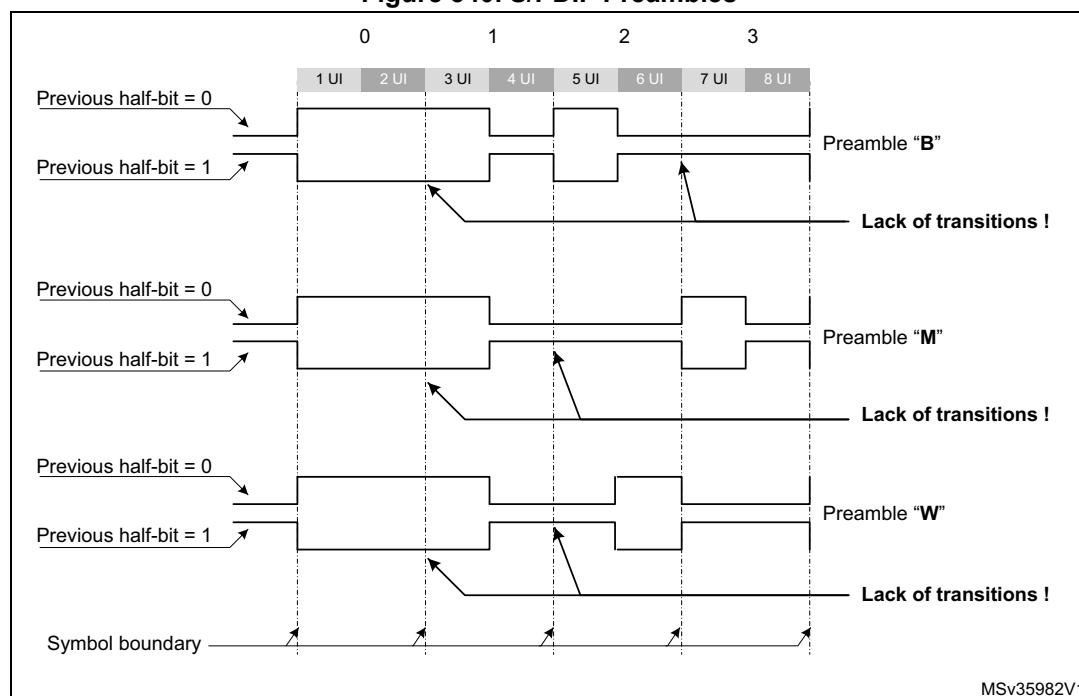
**Figure 339. S/PDIF block format**



### Synchronization preambles

The preambles patterns are inverted or not according to the previous half-bit value. This previous half-bit value is the level of the line before enabling a transfer for the first "B" preamble of the first frame. For the others preambles, this previous half-bit value is the second half-bit of the parity bit of the previous sub-frame. The preambles patterns B, M and W are described in the [Figure 340](#).

**Figure 340. S/PDIF Preambles**



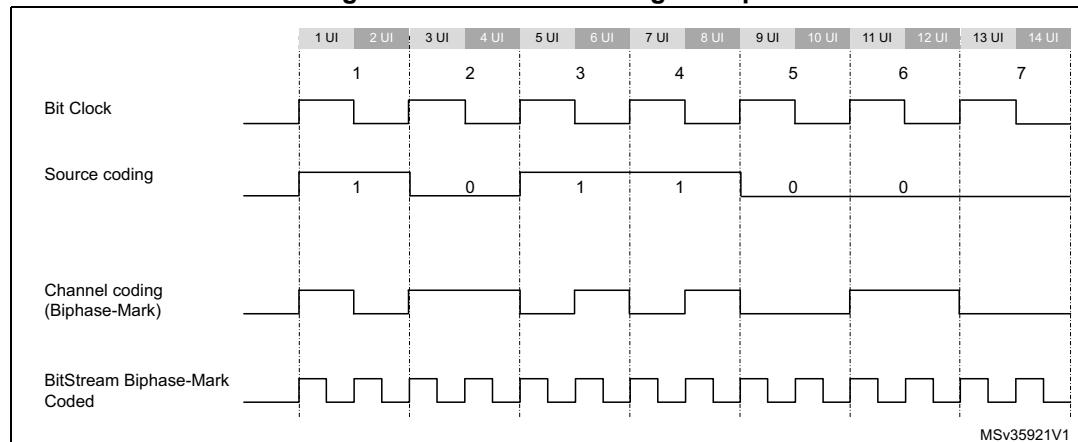
### Coding of information bits

In order to minimize the DC component value on the transmission line, and to facilitate clock recovery from the data stream, bits 4 to 31 are encoded in biphase-mark.

Each bit to be transmitted is represented by a symbol comprising two consecutive binary states. The first state of a symbol is always different from the second state of the previous symbol. The second state of the symbol is identical to the first if the bit to be transmitted is logical 0. However, it is different if the bit is logical 1. These states are named “UI” (Unit Interval) in the IEC-60958 specification.

The 24 data bits are transferred LSB first.

**Figure 341. Channel coding example**



### 27.3.2 SPDIFRX decoder (SPDIFRX\_DC)

#### Main principle

The technique used by the SPDIFRX in order to decode the S/PDIF stream is based on the measurement of the time interval between two consecutive edges. Three kinds of time intervals may be found into an S/PDIF stream:

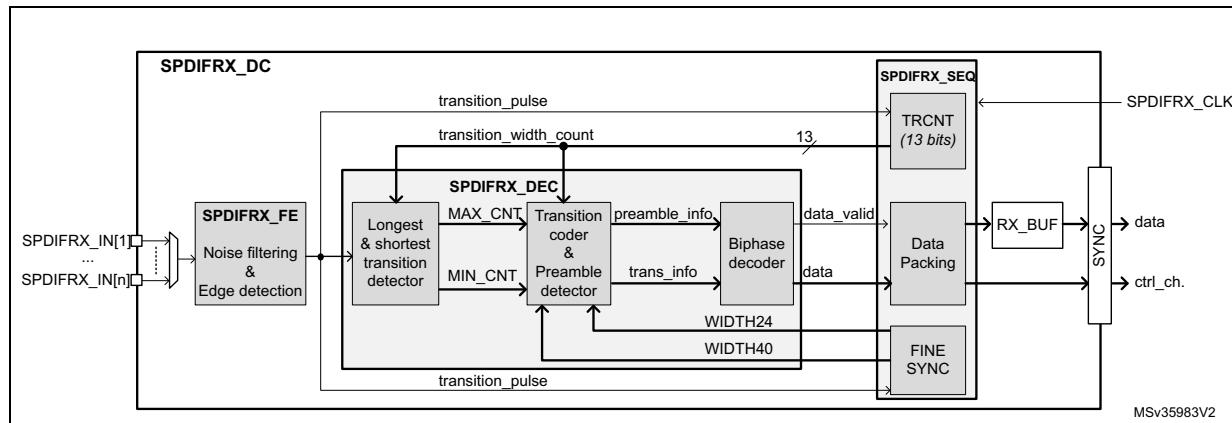
- The long time interval, having a duration of  $3 \times \text{UI}$ , noted TL. It appears only during preambles.
- The medium time interval, having a duration of  $2 \times \text{UI}$ , noted TM. It appears both in some preambles or into the information field.
- The short time interval, having a duration of  $1 \times \text{UI}$ , noted TS. It appears both in some preambles or into the information field.

The SPDIFRX\_DC block is responsible of the decoding of the received S/PDIF stream. It takes care of the following functions:

- Resampling and filtering of the incoming signal
- Estimation of the time-intervals
- Estimation of the symbol rate and synchronization
- Decoding of the serial data, and check of integrity
- Detection of the block, and sub-frame preambles
- Continuous tracking of the symbol rate

[Figure 342](#) gives a detailed view of the SPDIFRX decoder.

**Figure 342. SPDIFRX decoder**

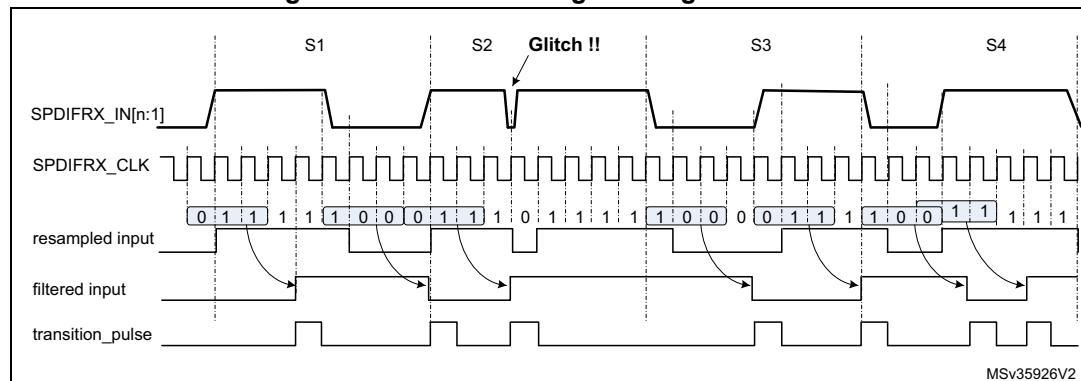


### Noise filtering & rising/falling edge detection

The S/PDIF signal received on the selected SPDIFRX\_IN is re-sampled using the SPDIFRX\_CLK clock (acquisition clock). A simple filtering is applied in order to cancel spurs. This is performed by the stage detecting the edge transitions. The edge transitions are detected as follows:

- A rising edge is detected when the sequence 0 followed by two 1 is sampled.
- A falling edge is detected when the sequence 1 followed by two 0 is sampled.
- After a rising edge, a falling edge sequence is expected.
- After a falling edge, a rising edge sequence is expected.

**Figure 343. Noise filtering and edge detection**



### Longest and shortest transition detector

The **longest and shortest transition detector** block detects the maximum (MAX\_CNT) and minimum (MIN\_CNT) duration between two transitions. The TRCNT counter is used to measure the time interval duration. It is clocked by the SPDIFRX\_CLK signal. On every transition pulse, the counter value is stored and the counter is reset to start counting again.

The maximum duration is normally found during the preamble period. This maximum duration is sent out as MAX\_CNT. The minimum duration is sent out as MIN\_CNT.

The search of the longest and shortest transition is stopped when the transition timer expires. The transition timer is like a watchdog timer that generates a trigger after 70 transitions of the incoming signal. Note that counting 70 transitions insures a delay a bit longer than a sub-frame.

Note that when the TRCNT overflows due to a too long time interval between two pulses, the SPDIFRX is stopped and the flag TERR of SPDIFRX\_SR register is set to 1.

### Transition coder and preamble detector

The **transition coder and preamble detector** block receives the MAX\_CNT and MIN\_CNT. It also receives the current transition width from the TRCNT counter (see [Figure 342](#)). This block encodes the current transition width by comparing the current transition width with two different thresholds, names TH<sub>HI</sub> and TH<sub>LO</sub>.

- If the current transition width is less than (TH<sub>LO</sub> - 1), then the data received is half part of data bit '1', and is coded as TS.
- If the current transition width is greater than (TH<sub>LO</sub> - 1), and less than TH<sub>HI</sub>, then the data received is data bit '0', and is coded as TM.
- If the current transition width is greater than TH<sub>HI</sub>, then the data received is the long pulse of preambles, and is coded as TL.
- Else an error code is generated (FERR flag is set).

The thresholds TH<sub>HI</sub> and TH<sub>LO</sub> are elaborated using two different methods.

If the peripheral is doing its initial synchronization ('coarse synchronization'), then the thresholds are computed as follow:

- TH<sub>LO</sub> = MAX\_CNT / 2.
- TH<sub>HI</sub> = MIN\_CNT + MAX\_CNT / 2.

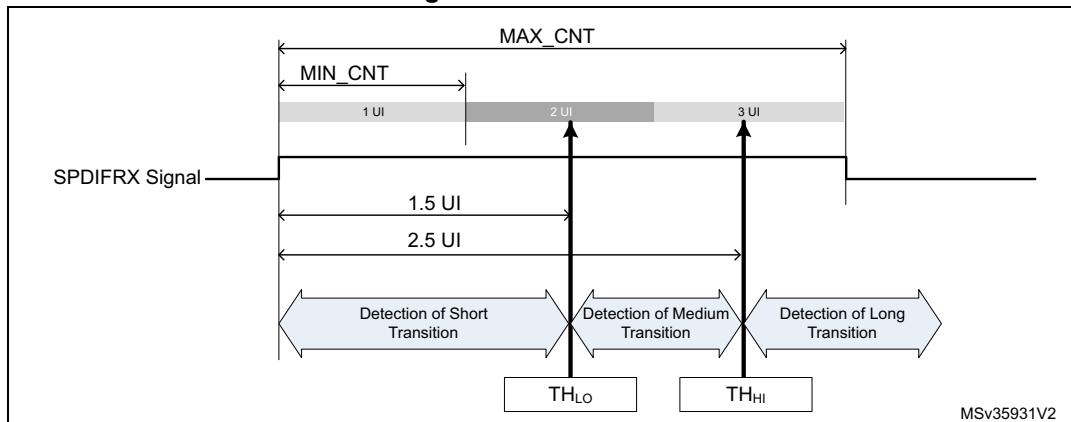
Once the 'coarse synchronization' is completed, then the SPDIFRX uses a more accurate reference in order to elaborate the thresholds. The SPDIFRX measures the length of 24 symbols (WIDTH24) for defining TH<sub>LO</sub> and the length of 40 symbols (WIDTH40) for TH<sub>HI</sub>. TH<sub>HI</sub> and TH<sub>LO</sub> are computed as follow:

- TH<sub>LO</sub> = (WIDTH24) / 32
- TH<sub>HI</sub> = (WIDTH40) / 32

This second synchronization phase is called the 'fine synchronization'. Refer to [Figure 346](#) for additional information.

As shown in the figure hereafter, TH<sub>LO</sub> is ideally equal to 1.5 UI, and to TH<sub>HI</sub> 2.5 UI.

Figure 344. Thresholds



The preamble detector checks four consecutive transitions of a specific sequence to determine if they form the part of preamble. Let us say TRANS0, TRANS1, TRANS2 and TRANS3 represent four consecutive transitions encoded as mentioned above. [Table 164](#) shows the values of these four transitions to form a preamble. Absence of this pattern indicates that these transitions form part of the data in the sub frame and bi-phase decoder will decode them.

Table 164. Transition sequence for preamble

Preamble type	Biphase data pattern	TRANS3	TRANS2	TRANS1	TRANS0
Preamble B	11101000	TL	TS	TS	TL
Preamble M	11100010	TL	TL	TS	TS
Preamble W	11100100	TL	TM	TS	TM

### Bi-phase decoder

The Bi-phase decoder decodes the input bi-phase marked data stream using the transition information provided by the **transition coder and preamble detector** block. It first waits for the preamble detection information. After the preamble detection, it decodes the following transition information:

- If the incoming transition information is TM then it is decoded as a '0'.
- Two consecutive TS are decoded as a '1'.
- Any other transition sequence generates an error signal (FERR set to 1).

After decoding 28 data bits this way, this module looks for the following preamble data. If the new preamble is not what is expected, then this block generates an error signal (FERR set to 1). Refer to [Section 27.3.8: Reception errors](#), for additional information on error flags.

### Data packing

This block is responsible of the decoding of the IEC-60958 frames and blocks. It also handles the writing into the RX\_BUF or into SPDIFRX\_CSR register.

### 27.3.3 SPDIFRX tolerance to clock deviation

The SPDIFRX tolerance to clock deviation depends on the number of sample clock cycles in one bit slot. The fastest SPDIFRX\_CLK is, the more robust the reception will be. The ratio between SPDIFRX\_CLK frequency and the symbol rate must be at least 11.

Two kinds of phenomenon (at least!) can degrade the reception quality:

- The cycle-to-cycle jitter which reflects the difference of transition length between two consecutive transitions.
- The long term jitter which reflects a cumulative effect of the cycle-to-cycle jitter. It can be seen as a low-frequency symbol modulation.

### 27.3.4 SPDIFRX synchronization

The synchronization phase starts when setting SPDIFRXEN to 0b01 or 0b11. [Figure 345](#) shows the synchronization process.

If the bit WFA of SPDIFRX\_CR register is set to 1, then the peripheral must first detect activity on the selected SPDIFRX\_IN line before starting the synchronization process. The activity detection is performed by detecting four transitions on the selected SPDIFRX\_IN. The peripheral remains in this state until transitions are not detected. This function can be particularly helpful because the IP switches in COARSE SYNC mode only if activity is present on the selected SPDIFRX\_IN input, avoiding synchronization errors. See [Section 27.4: Programming procedures](#) for additional information.

The user can still set the SPDIFRX into STATE\_IDLE by setting SPDIFRXEN to 0. If the WFA is set to 0, the peripheral starts the coarse synchronization without checking activity.

The next step consists on doing a first estimate of the thresholds (COARSE SYNC), in order to perform the fine synchronization (FINE SYNC). Due to disturbances of the SPDIFRX line, it could happen that the process is not executed first time right. For this purpose, the user can program the number of allowed re-tries (NBTR) before setting SERR error flag.

When the SPDIFRX has been able to measure properly the duration of 24 and 40 consecutive symbols then the FINE SYNC is completed, the threshold values are updated, and the flag SYNCD is set to 1. Refer to [Section : Transition coder and preamble detector](#) for additional information.

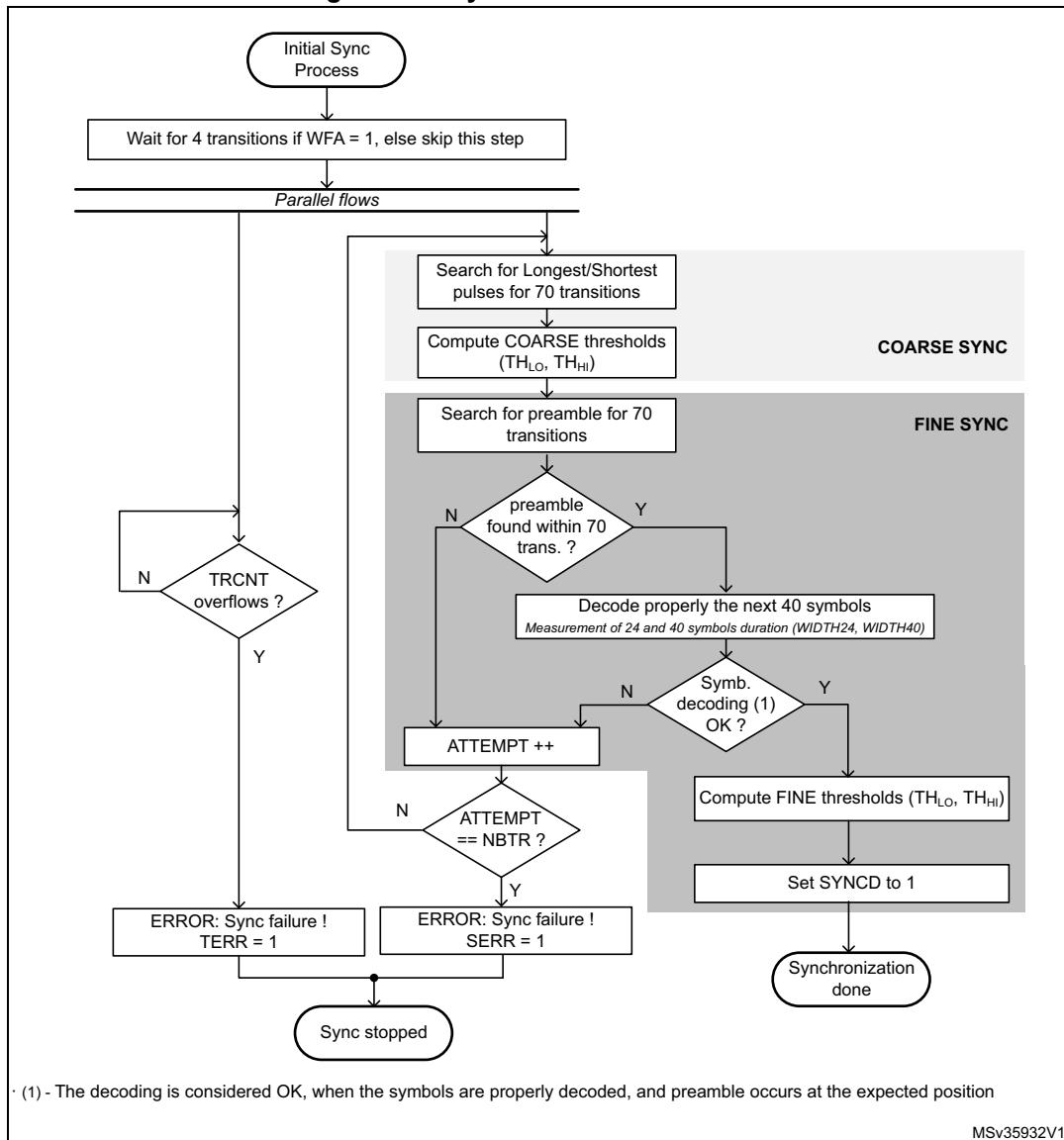
Two kinds of errors are detected:

- An overflow of the TRCNT, which generally means that there is no valid S/PDIF stream in the input line. This overflow is indicated by TERR flag.
- The number of retries reached the programmed value. This means that strong jitter is present on the S/PDIF signal. This error is indicated by SERR flag.

When the first FINE SYNC is completed, the reception of channel status (C) and user data (U) will start when the next “B” preamble is detected (see [Figure 349](#)). Then the user can read IEC-60958 C and U bits through SPDIFRX\_CSR register. According to this information the user can then select the proper settings for DRFMT and RXSTEO. For example if the user detects that the current audio stream transports encoded data, then he can put RXSTEO to 0, and DRFMT to 0b10 prior to start data reception. Note that DRFMT and RXSTEO cannot be modified when SPDIFRXEN = 0b11. Writes to these fields are ignored if SPDIFRXEN is already 0b11, though these field can be changed with the same write instruction that causes SPDIFRXEN to become 0b11.

Then the SPDIFRX waits for SPDIFRXEN = 0b11 and the “B” preamble before starting saving audio samples.

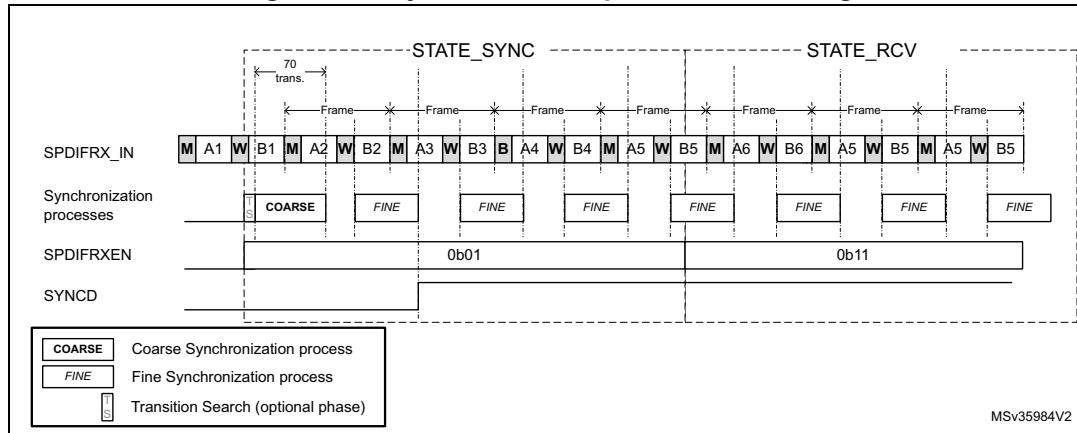
Figure 345. Synchronization flowchart



Refer to [Frame structure and synchronization error](#) for additional information concerning TRCNT overflow.

The FINE SYNC process is re-triggered every frame in order to update thresholds as shown in [Figure 346](#) in order to continuously track S/PDIF synchronization.

Figure 346. Synchronization process scheduling



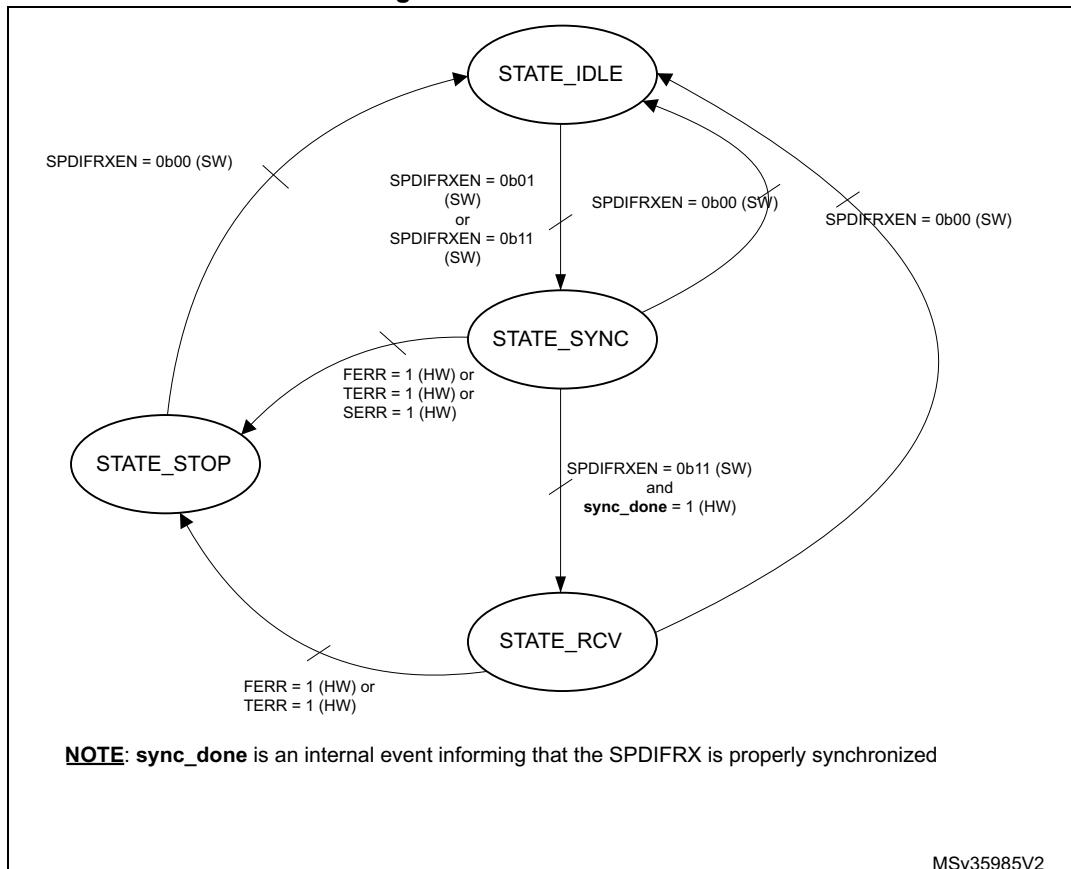
### 27.3.5 SPDIFRX handling

The software can control the state of the SPDIFRX through SPDIFRXEN field. The SPDIFRX can be into one of the following states:

- STATE\_IDLE:  
The peripheral is disabled, the SPDIFRX\_CLK domain is reset. The PCLK1 domain is functional.
- STATE\_SYNC:  
The peripheral is synchronized to the stream, thresholds are updated regularly, user and channel status can be read via interrupt or DMA. The audio samples are not provided to receive buffer.
- STATE\_RCV:  
The peripheral is synchronized to the stream, thresholds are updated regularly, user, channel status and audio samples can be read via interrupt or DMA channels. When SPDIFRXEN goes to 0b11, the SPDIFRX waits for "B" preamble before starting saving audio samples.
- STOP\_STATE:  
The peripheral is no longer synchronized, the reception of the user, channel status and audio samples are stopped. It is expected that the software re-starts the SPDIFRX.

The [Figure 347](#) shows the possible states of the SPDIFRX, and how to transition from one state to the other. The bits under software control are followed by the mention "(SW)", the bits under IP control are followed by the mention "(HW)".

Figure 347. SPDIFRX States



MSv35985V2

When SPDIFRX is in STATE\_IDLE:

- The software can transition to STATE\_SYNC by setting SPDIFRXEN to 0b01 or 0b11

When SPDIFRX is in STATE\_SYNC:

- If the synchronization fails or if the received data are not properly decoded with no chance of recovery without a re-synchronization (FERR or SERR or TERR = 1), the SPDIFRX goes to STATE\_STOP, and waits for software acknowledge.
- When the synchronization phase is completed, if SPDIFRXEN = 0b01 the peripheral remains in this state.
- At any time the software can set SPDIFRXEN to 0, then SPDIFRX returns immediately to STATE\_IDLE. If a DMA transfer is on-going, it will be properly completed.
- The SPDIFRX goes to STATE\_RCV if SPDIFRXEN = 0b11 and if the SYNCD = 1

When SPDIFRX is in STATE\_RCV:

- If the received data are not properly decoded with no chance of recovery without a re-synchronization (FERR or SERR or TERR = 1), the SPDIFRX goes to STATE\_STOP, and waits for software acknowledge.
- At any time the software can set SPDIFRXEN to 0, then SPDIFRX returns immediately to STATE\_IDLE. If a DMA transfer is on-going, it will properly be completed.

When SPDIFRX is in STATE\_STOP:

- The SPDIFRX stops reception and synchronization, and waits for the software to set the bit SPDIFRXEN to 0, in order to clear the error flags.

When SPDIFRXEN is set to 0, the IP is disabled, meaning that all the state machines are reset, and RX\_BUF is flushed. Note as well that flags FERR, SERR and TERR are reset.

### 27.3.6 Data reception management

The SPDIFRX offers a double buffer for the audio sample reception. A 32-bit buffer located into the SPDIFRX\_CLK clock domain (RX\_BUF), and the SPDIFRX\_DR register. The valid data contained into the RX\_BUF will be immediately transferred into SPDIFRX\_DR if SPDIFRX\_DR is empty.

The valid data contained into the RX\_BUF will be transferred into SPDIFRX\_DR when the two following conditions are reached:

- The transition between the parity bit (P) and the next preamble is detected (this indicated that the word has been completely received).
- The SPDIFRX\_DR is empty.

Having a 2-word buffer gives more flexibility for the latency constraint.

The maximum latency allowed is  $T_{\text{SAMPLE}} - 2T_{\text{PCLK}} - 2T_{\text{SPDIFRX\_CLK}}$

Where  $T_{\text{SAMPLE}}$  is the audio sampling rate of the received stereo audio samples,  $T_{\text{PCLK}}$  is the period of PCLK1 clock, and  $T_{\text{SPDIFRX\_CLK}}$  is the period of SPDIFRX\_CLK clock.

The SPDIFRX offers the possibility to use either DMA (spdifrx\_dat\_dma and spdifrx\_ctrl\_dma) or interrupts for transferring the audio samples into the memory. The recommended option is DMA, refer to [Section 27.3.11: DMA Interface](#) for additional information.

The SPDIFRX offers several way on handling the received data. The user can either have a separate flow for control information and audio samples, or get them all together.

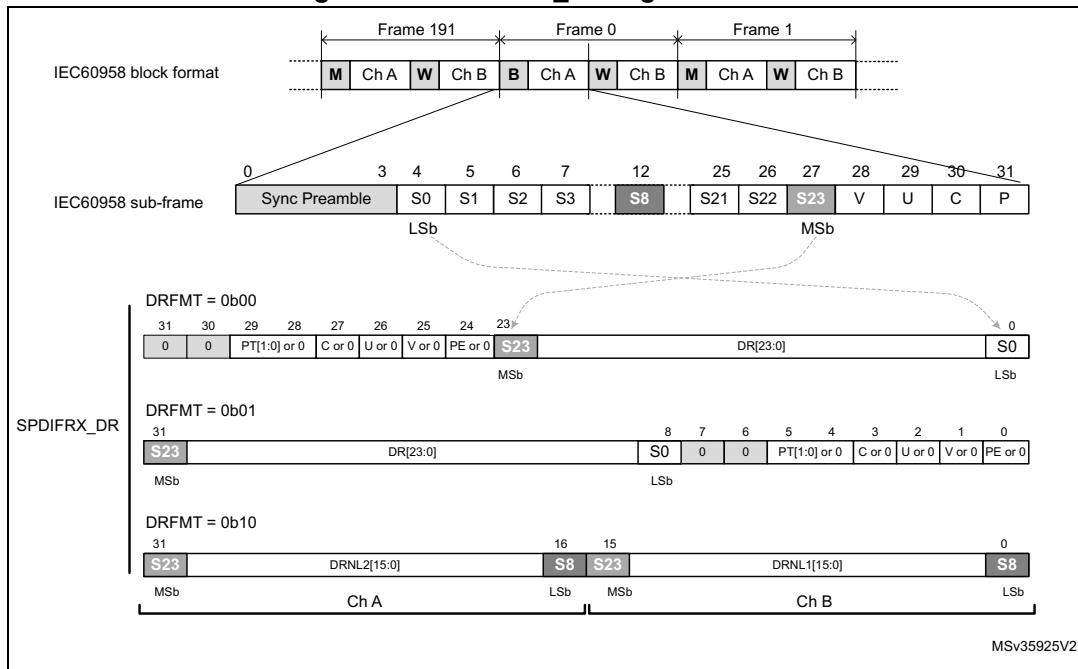
For each sub-frame, the data reception register SPDIFRX\_DR contains the 24 data bits, and optionally the V, U, C, PE status bits, and the PT (see [Mixing data and control flow](#)).

Note that PE bit stands for Parity Error bit, and will be set to 1 when a parity error is detected in the decoded sub-frame.

The PT field carries the preamble type (B, M or W).

V, U and C are a direct copy of the value received from the S/PDIF interface.

The bit DRFMT allows the selection between 3 audio formats as shown in [Figure 348](#).

**Figure 348. SPDIFRX\_DR register format**

Setting DRFMT to 0b00 or 0b01, offers the possibility to have the data either right or left aligned into the SPDIFRX\_DR register. The status information can be enabled or forced to zero according to the way the software wants to handle them.

The format given by DRFMT= 0b10 is interesting in non-linear mode, as only 16 bits per sub-frame are used. By using this format, the data of two consecutive sub-frames are stored into SPDIFRX\_DR, dividing by two the amount of memory footprint. Note that when RXSTEO = 1, there is no misalignment risks (i.e. data from ChA will be always stored into SPDIFRX\_DR[31:16]). If RXSTEO = 0, then there is a misalignment risk in case of overrun situation. In that case SPDIFRX\_DR[31:16] will always contain the oldest value and SPDIFRX\_DR[15:0] the more recent value (see [Figure 350](#)).

In this format the status information cannot be mixed with data, but the user can still get them through SPDIFRX\_CSR register, and use a dedicated DMA channel or interrupt to transfer them to memory (see [Section 27.3.7: Dedicated control flow](#))

### Mixing data and control flow

The user can choose to use this mode in order to get the full flexibility of the handling of the control flow. The user can select which field shall be kept into the data register (SPDIFRX\_DR).

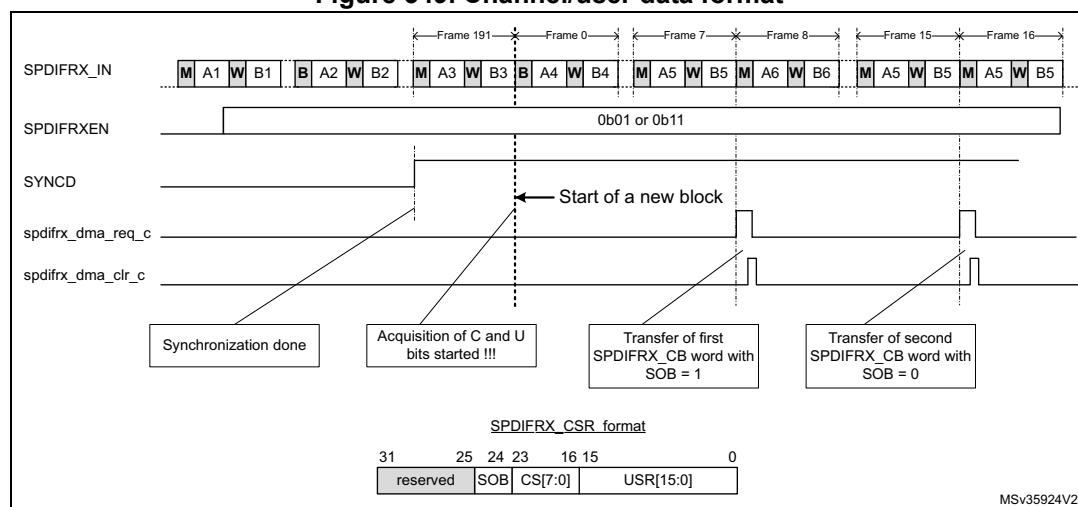
- When bit PMSK = 1, the Parity Error information is masked (set to 0), otherwise it is copied into SPDIFRX\_DR.
- When bit VMSK = 1, the Validity information is masked (set to 0), otherwise it is copied into SPDIFRX\_DR.
- When bit CUMSK = 1, the Channel Status, and Used data information are masked (set to 0), otherwise they are copied into SPDIFRX\_DR.
- When bit PTMSK = 1, the Preamble Type is masked (set to 0), otherwise it is copied into SPDIFRX\_DR.

### 27.3.7 Dedicated control flow

The SPDIFRX offers the possibility to catch both user data and channel status information via a dedicated DMA channel. This feature allows the SPDIFRX to acquire continuously the channel status and user information. The acquisition will start at the beginning of a IEC 60958 block. Two fields are available to control this path: CBDMAEN and SPDIFRXEN. When SPDIFRXEN is set to 0b01 or 0x11, the acquisition is started, after completion of the synchronization phase. When 8 channel status and 16 user data bits have been received, they are packed and stored into SPDIFRX\_CSR register. A DMA request is triggered if the bit CBDMAEN is set to 1 (see [Figure 349](#)).

If CS[0] corresponds to the first bit of a new block, the bit SOB will be set to 1. Refer to [Section 27.5.8: Channel status register \(SPDIFRX\\_CSR\)](#). A bit is available (CHSEL) in order to select if the user wants to select channel status information (C) from the channel A or B.

**Figure 349. Channel/user data format**



**Note:** Once the first start of block is detected (B preamble), the SPDIFRX is checking the preamble type every 8 frames.

**Note:** Overrun error on SPDIFRX\_DR register does not affect this path.

### 27.3.8 Reception errors

#### Frame structure and synchronization error

The SPDIFRX, detects errors, when one of the following condition occurs:

- The FERR bit is set to 1 on the following conditions:
  - For each of the 28 information bits, if one symbol transition sequence is not correct: for example if short pulses are not grouped by pairs.
  - If preambles occur to an unexpected place, or an expected preamble is not received.
- The SERR bit is set when the synchronization fails, because the number of re-tries exceeded the programmed value.
- The TERR bit is set when the counter used to estimate the width between two transitions overflows (TRCNT).

The overflow occurs when no transition is detected during 8192 periods of SPDIFRX\_CLK clock. It represents at most a time interval of 11.6 frames.

When one of those flags goes to 1, the traffic on selected SPDIFRX\_IN is then ignored, an interrupt is generated if the IFEIE bit of the SPDIFRX\_CR register is set.

The normal procedure when one of those errors occur is:

- Set SPDIFRXEN to 0 in order to clear the error flags
- Set SPDIFRXEN to 0b01 or 0b11 in order to restart the IP

Refer to [Figure 347](#) for additional information.

### Parity error

For each sub-frame, an even number of zeros and ones is expected inside the 28 information bits. If not, the parity error bit PERR is set in the SPDIFRX\_SR register and an interrupt is generated if the parity interrupt enable PERRIE bit is set in the SPDIFRX\_CR register. The reception of the incoming data is not paused, and the SPDIFRX continue to deliver data to SPDIFRX\_DR even if the interrupt is still pending.

The interrupt is acknowledged by clearing the PERR flag through PERRCF bit.

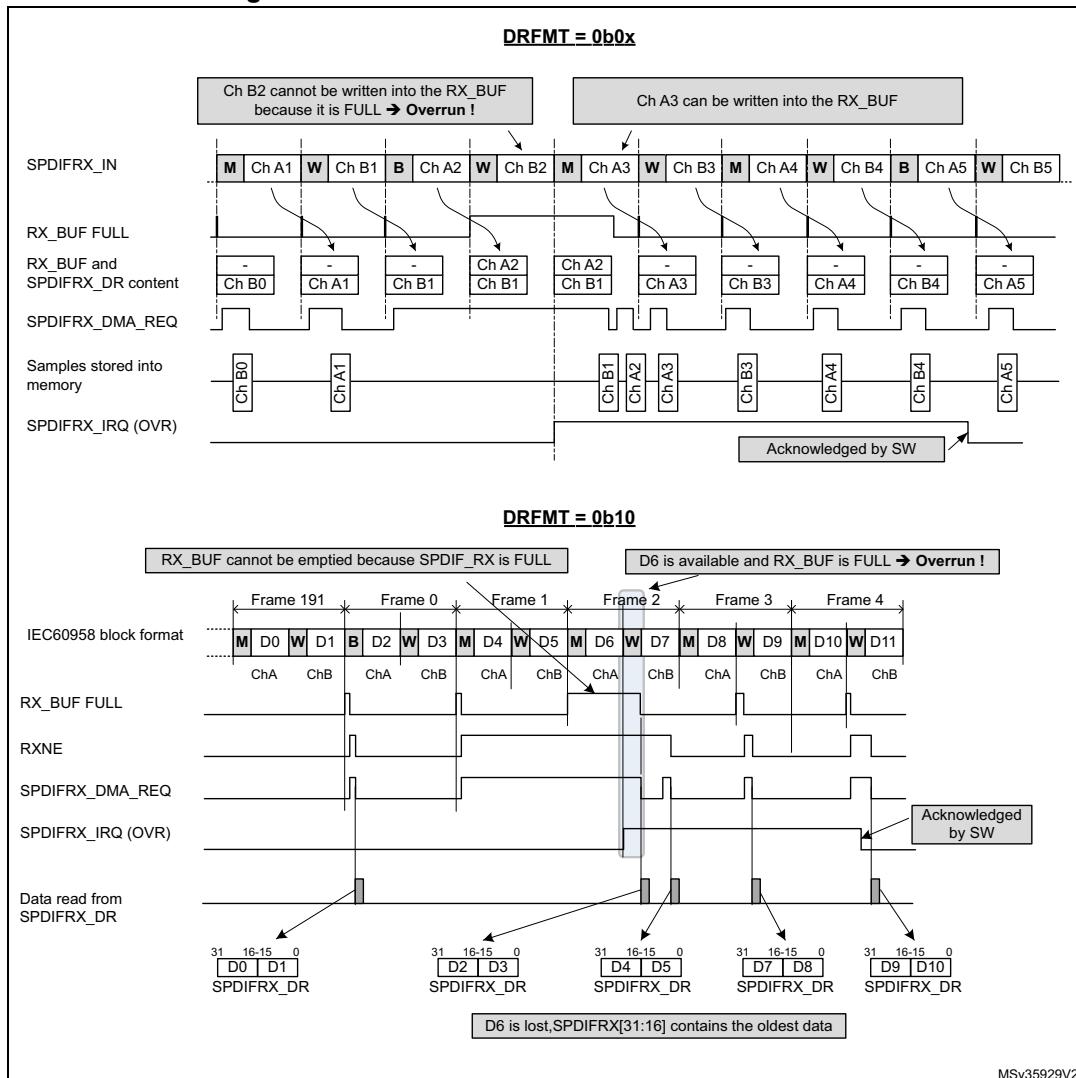
If the software wants to guarantee the coherency between the data read in the SPDIFRX\_DR register and the value of the bit PERR, the bit PMSK must be set to 0.

### Overrun error

If both SPDIFRX\_DR and RX\_BUF are full, while the SPDIFRX\_DC needs to write a new sample in RX\_BUF, this new sample is dropped, and an overrun condition is triggered. The overrun error flag OVR is set in the SPDIFRX\_SR register and an interrupt is generated if the OVRIE bit of the SPDIFRX\_CR register is set.

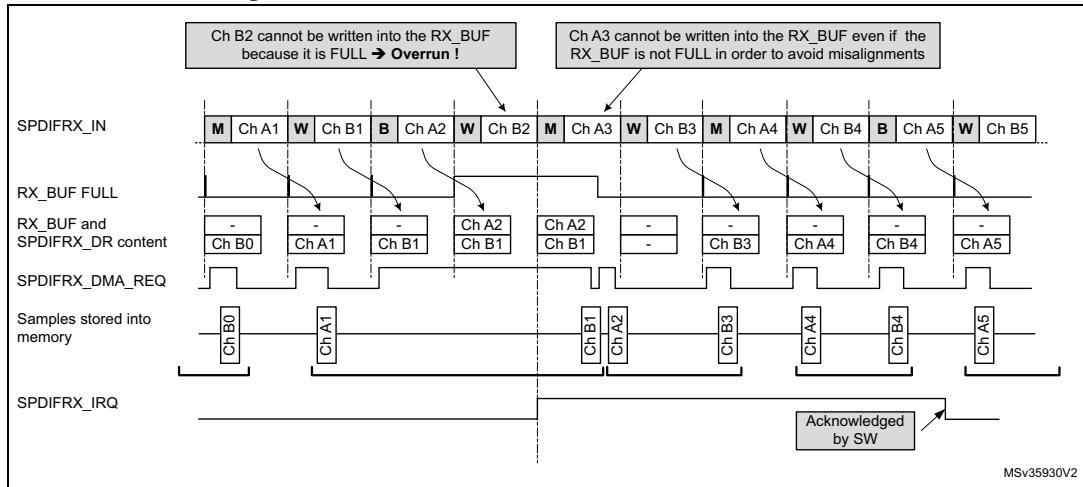
If the RXSTEO bit is set to 0, then as soon as the RX\_BUF is empty, the IP will store the next incoming data, even if the OVR flag is still pending. The main purpose is to reduce as much as possible the amount of lost samples. Note that the behavior is similar independently of DRFMT value. See [Figure 350](#).

Figure 350. S/PDIF overrun error when RXSTEO = 0



If the RXSTEO bit is set to 1, it means that stereo data are transported, then the SPDIFRX has to avoid misalignment between left and right channels. So the peripheral has to drop a second sample even if there is room inside the RX\_BUF in order to avoid misalignment. Then the incoming samples can be written normally into the RX\_BUF even if the OVR flag is still pending. Refer to [Figure 351](#).

The OVR flag is cleared by software, by setting the OVRCF bit to 1.

**Figure 351. S/PDIF overrun error when RXSTEO = 1**

MSv35930V2

### 27.3.9 Clocking strategy

The SPDIFRX block needs two different clocks:

- The APB1 clock (PCLK1), which is used for the register interface,
- The SPDIFRX\_CLK which is mainly used by the SPDIFRX\_DC part. Those clocks are not supposed to be phase locked, so all signals crossing those clock domains are re-synchronized (SYNC block on [Figure 337](#)).

In order to decode properly the incoming S/PDIF stream the SPDIFRX\_DC shall re-sample the received data with a clock at least 11 times higher than the maximum symbol rate, or 704 times higher than the audio sample rate. For example if the user expects to receive a symbol rate up to 12.288 MHz, the sample rate shall be at least 135.2 MHz. The clock used by the SPDIFRX\_DC is the SPDIFRX\_CLK.

The frequency of the PCLK1 must be at least equal to the symbol rate.

**Table 165. Minimum SPDIFRX\_CLK frequency versus audio sampling rate**

Symbol Rate	Minimum SPDIFRX_CLK frequency	Comments
3.072 MHz	33.8 MHz	For 48 kHz stream
6.144 MHz	67.6 MHz	For 96 kHz stream
12.288 MHz	135.2 MHz	For 192 kHz stream

### 27.3.10 Symbol clock generation

The SPDIFRX block provides a symbol clock on signal named **spdifrx\_symb\_ck**, which can be used as the reference kernel clock for another audio device such as SAI or SPI/I2S. It could be used for SPDIFRX to I2S bridge function.

The symbol clock is built using the values of WIDTH24, WIDTH40 and the symbol boundaries.

- During the reception of the sub-frame sync preambles, the falling and rising edges of the symbol clock are built from the WIDTH24 and WIDTH40 values. Note that WIDTH24 and WIDTH40 are also used for the generation of the symbol clock, when the SPDIFRX is STATE\_STOP or STATE\_IDLE. See [Table 166](#) for details.
- During the reception of the sub-frame payload, the SPDIFRX uses the symbols boundaries to generate the rising edge, the WIDTH24 and WIDTH40 values for the generation of the falling edge.

The duty cycle of the symbol clock is close to 50% during the reception of the sub-frame payload. However, the duty cycle can be altered when the SPDIFRX transitions from a symbol clock generated with WIDTH24 and WIDTH40 to a clock generated by the symbol clock boundaries or vice-versa.

The symbol clock will have an important jitter mainly due to:

- The re-sampling of the S/PDIF signal with SPDIFRX\_CLK clock
- The transition of the symbol clock generation mode

For that reason the application shall consider the quality degradation if the symbol clock is used as the reference clock for A/D or D/A converters.

The generation of this symbol clock is controlled by the CKSEN bit. When CKSEN = '1', the clock symbol is generated when the SPDIFRX completes successfully the first fine synchronization (SYNCD = 1), and when it is receiving correct data from the selected SPDIFRX input.

When the SPDIFRX goes to STATE\_STOP, or STATE\_IDLE, the symbol clock is gated if the bit CKSBKPEN = '0'. If the CKSBKPEN = '1', then a backup symbol clock is still generated if the SPDIFRX is properly synchronized (i.e. valid values available for WIDTH24 and WIDTH40). [Table 166](#) gives more details on the conditions controlling the generation of the symbol clock.

**Table 166. Conditions of spdifrx\_symb\_ck generation**

SPDIFRX states and conditions	CKSEN	CKSBKPEN	spdifrx_symb_ck state
Any state	0	X	Disabled
<ul style="list-style-type: none"> <li>– SPDIFRX in STATE_SYNC and completing successfully the fine synchronization (SYNCD = '1') or,</li> <li>– SPDIFRX in STATE_RCV, and valid data are received via the selected SPDIFRX input.</li> </ul>		0	Enabled
<ul style="list-style-type: none"> <li>– SPDIFRX in STATE_IDLE or,</li> <li>– SPDIFRX in STATE_STOP or,</li> <li>– SPDIFRX did not complete the fine synchronization (on-going)</li> <li>– SPDIFRX is in STATE_RCV, but no data (transitions) detected on the selected SPDIFRX input.</li> </ul>	1	0	Disabled
<ul style="list-style-type: none"> <li>– SPDIFRX in STATE_IDLE, but with valid values for WIDTH40 and WIDTH24 or</li> <li>– SPDIFRX in STATE_SYNC and completing successfully the fine synchronization (SYNCD = '1') or,</li> <li>– SPDIFRX in STATE_SYNC the on-going fine synchronization is not completed, but WIDTH40 and WIDTH24 contain the valid values from the previous synchronization or,</li> <li>– SPDIFRX in STATE_RCV, and valid data are received via the selected SPDIFRX input or,</li> <li>– SPDIFRX in STATE_STOP, but with valid values for WIDTH40 and WIDTH24.</li> </ul>		1	Enabled
<ul style="list-style-type: none"> <li>– SPDIFRX in IDLE, with invalid values for WIDTH40 and WIDTH24 or,</li> <li>– SPDIFRX in STOP with invalid values for WIDTH40 and WIDTH24 (SERR = '1') or,</li> <li>– SPDIFRX in STATE_SYNC with invalid values for WIDTH40 and WIDTH24, and did not completed the on-going fine synchronization or,</li> <li>– SPDIFRX in STATE_RCV and no transitions detected on the selected SPDIFRX input</li> </ul>			Disabled

Note that when the flag SERR is set to '1', neither the symbol clock nor the backup clock can be generated, since there is no synchronization.

Note that when both CKSEN and CKSBKPEN are set to '1', the symbol clock will loose some transitions when the SPDIFRX switches from STATE\_SYNC or STATE\_RCV to STATE\_STOP, or STATE\_IDLE.

The bits CKSEN and CKSBKPEN are located into [Control register \(SPDIFRX\\_CR\)](#).

### 27.3.11 DMA Interface

The SPDIFRX interface is able to perform communication using the DMA.

*Note:* The user should refer to product specifications for availability of the DMA controller.

The SPDIFRX offers two independent DMA channels:

- A DMA channel dedicated to the data transfer
- A DMA channel dedicated to the channel status and user data transfer

The DMA mode for the data can be enabled for reception by setting the RXDMAEN bit in the SPDIFRX\_CR register. In this case, as soon as the SPDIFRX\_DR is not empty, the

SPDIFRX interface sends a transfer request to the DMA. The DMA reads the data received through the SPDIFRX\_DR register without CPU intervention.

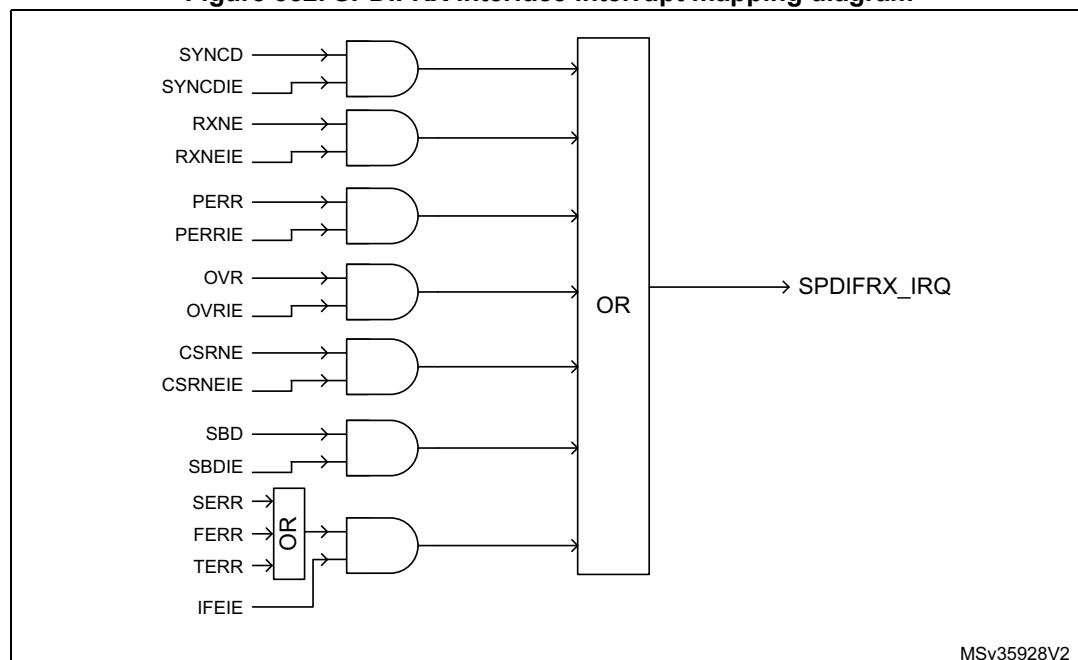
For the use of DMA for the control data refer to [Section 27.3.7: Dedicated control flow](#).

### 27.3.12 Interrupt Generation

An interrupt line is shared between:

- Reception events for data flow (RXNE)
- Reception event for control flow (CSRNE)
- Data corruption detection (PERR)
- Transfer flow interruption (OVR)
- Frame structure and synchronization errors (SERR, TERR and FERR)
- Start of new block interrupt (SBD)
- Synchronization done (SYNCD)

**Figure 352. SPDIFRX interface interrupt mapping diagram**



#### Clearing interrupt source

- RXNE is cleared when SPDIFRX\_DR register is read
- CSRNE is cleared when SPDIFRX\_CSR register is read
- FERR is cleared when SPDIFRXEN is set to 0
- SERR is cleared when SPDIFRXEN is set to 0
- TERR is cleared when SPDIFRXEN is set to 0
- Others are cleared through SPDIFRX\_IFCR register

**Note:** The SBD event can only occur when the SPDIFRX is synchronized to the input stream (SYNCD = 1).

*The SBD flag behavior is not guaranteed when the sub-frame which contains the B preamble is lost due to an overrun.*

### 27.3.13 Register protection

The SPDIFRX block embeds some hardware protection avoid erroneous use of control registers. The table hereafter shows the bit field properties according to the SPDIFRX state.

**Table 167. Bit field property versus SPDIFRX state**

Registers	Field	SPDIFRXEN		
		0b00 (STATE_IDLE)	0b01 (STATE_SYNC)	0b11 (STATE_RCV)
SPDIFRX_CR	INSEL	rw	r	r
	WFA	rw	r	r
	NBTR	rw	r	r
	CHSEL	rw	r	r
	CBDMAEN	rw	rw	rw
	PTMSK	rw	rw	rw
	CUMSK	rw	rw	rw
	VMSK	rw	rw	rw
	PMSK	rw	rw	rw
	DRFMT	rw	rw	r
	RXSTEO	rw	rw	r
	RXDMAEN	rw	rw	rw
SPDIFRX_IMR	All fields	rw	rw	rw

The table clearly shows that fields such as INSEL must be programmed when the IP is in STATE\_IDLE. In the others IP states, the hardware prevents writing to this field.

**Note:** *Even if the hardware allows the writing of CBDMAEN and RXDMAEN “on-the-fly”, it is not recommended to enable the DMA when the IP is already receiving data.*

**Note:** *Note that each of the mask bits (PMSK, VMSK, ...) can be changed “on-the-fly” at any IP state, but any change does not affect data which is already being held in SPDIFRX\_DR.*

## 27.4 Programming procedures

The following example illustrates a complete activation sequence of the SPDIFRX block. The data path and channel status & user information will both use a dedicated DMA channel. The activation sequence is then split into the following steps:

- Wait for valid data on the selected SPDIFRX\_IN input
- Synchronize to the S/PDIF stream
- Read the channel status and user information in order to setup the complete audio path
- Start data acquisition

A simple way to check if valid data are available into the SPDIFRX\_IN line is to switch the SPDIFRX into the STATE\_SYNC, with bit WFA set to 1. The description hereafter will focus on detection. It is also possible to implement this function as follow:

- The software has to check from time to time (i.e. every 100 ms for example) if the SPDIFRX can find synchronization. This can be done by checking if the bit TERR is set. When it is set it indicates that no activity has been found.
- Connect the SPDIFRX\_IN input to an external interrupt event block in order to detect transitions of SPDIFRX\_IN line. When activity is detected, then SPDIFRXEN can be set to 0b01 or 0b11.

For those two implementations, the bit WFA is set to 0.

#### 27.4.1 Initialization phase

- The initialization function will look like this:
- Configure the DMA transfer for both audio samples and IEC60958 channel status and user information (DMA channel selection and activation, priority, number of data to transfer, circular/no circular mode, DMA interrupts)
- Configure the destination address:
  - Configure the address of the SPDIFRX\_CSR register as source address for IEC60958 channel status and user information
  - Configure the address of the SPDIFRX\_DR register as source address for audio samples
  - Enable the generation of the SPDIFRX\_CLK. Refer to [Table 165](#) in order to define the minimum clock frequency versus supported audio sampling rate.  
Note that the audio sampling rate of the received stream is not known in advance. This means that the user has to select a SPDIFRX\_CLK frequency at least 704 times higher than the maximum audio sampling rate the application is supposed to handle: for example if the application is able to handle streams up to 96 kHz, then  $F_{SPDIFRX\_CLK}$  shall be at least  $704 \times 96 \text{ kHz} = 67.6 \text{ MHz}$
- Enable interrupt for errors and event signaling (IFEIE = SYNC DIE = OVRIE, PERRIE = 1, others set to 0). Note that SYNC DIE can be set to 0.
- Configure the SPDIFRX\_CR register:
  - INSEL shall select the wanted input
  - NBTR = 2, WFA = 1 (16 re-tries allowed, wait for activity before going to synchronization phase),
  - PTMSK = CUMSK = 1 (Preamble, C and U bits are not mixed with data)
  - VMSK = PMSK = 0 (Parity error and validity bit mixed with data)
  - CHSEL = 0 (channels status will be read from sub-frame A)
  - DRFMT = 0b01 (data aligned to the left)
  - RXSTEO = 1 (expected stereo mode linear)
  - CBDMAEN = RXDMAEN = 1 (enable DMA channels)
  - SPDIFRXEN = 0b01 (switch SPDIFRX to STATE\_SYNC)
- The CPU can enter in WFI mode

Then the CPU will receive interrupts coming either from DMA or SPDIFRX.

### 27.4.2 Handling of interrupts coming from SPDIFRX

When an interrupt from the SPDIFRX is received, then the software has to check what is the source of the interrupt by reading the SPDIFRX\_SR register.

- If SYNCDC is set to 1, then it means that the synchronization has been properly completed. No action has to be performed in our case as the DMA is already programmed. The software just needs to wait for DMA interrupt in order to read channel status information.  
The SYNCDC flag must be cleared by setting SYNCDCF bit of SPDIFRX\_IFCR register to 1.
- If TERR or SERR or FERR are set to 1, the software has to set SPDIFRXEN to 0, and re-start from the initialization phase.
  - TERR indicates that a time-out occurs either during synchronization phase or after.
  - SERR indicates that the synchronization fails because the maximum allowed retries have been reached.
  - FERR indicates that the reading of information after synchronization fails (unexpected preamble, bad data decoding...).
- If PERR is set to 1, it means that a parity error has been detected, so one of the received audio sample or the channel status or user data bits are corrupted. The action taken here depends on the application: one action could be to drop the current channel status block as it is not reliable. There is no need to re-start from the initialization phase, as the synchronization is not lost.  
The PERR flag must be cleared by setting PERRCF bit of SPDIFRX\_IFCR register to 1.

### 27.4.3 Handling of interrupts coming from DMA

If an interrupt is coming from the DMA channel used of the channel status (SPDIFRX\_CSR):

If no error occurred (i.e. PERR), the CPU can start the decoding of channel information. For example bit 1 of the channel status informs the user if the current stream is linear or not. This information is very important in order to set-up the proper processing chain. In the same way, bits 24 to 27 of the channel status give the sampling frequency of the stream incoming stream.

Thanks to that information, the user can then configure the RXSTEO bit and DRFMT field prior to start the data reception. For example if the current stream is non linear PCM then RXSTEO is set to 0, and DRFMT is set to 0b10. Then the user can enable the data reception by setting SPDIFRXEN to 0b11.

The bit SOB, when set to 1 indicates the start of a new block. This information will help the software to identify the bit 0 of the channel status. Note that if the DMA generates an interrupt every time 24 values are transferred into the memory, then the first word will always correspond to the start of a new block.

If an interrupt is coming from the DMA channel used of the audio samples (SPDIFRX\_DR):

The process performed here depends of the data type (linear or non-linear), and on the data format selected.

For example in linear mode, if PE or V bit is set a special processing can be performed locally in order to avoid spurs on output. In non-linear mode those bits are not important as data frame have their own checksum.

## 27.5 SPDIFRX interface registers

### 27.5.1 Control register (SPDIFRX\_CR)

Address offset: 0x00

Reset value: 0x00000000

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.	CKSBKPEN	CKSEN	Res.	INSEL[2:0] <sup>(1)</sup>		
										rw	rw		rw	rw	rw
Res.	WFA <sup>(1)</sup>	NBTR[1:0] <sup>(1)</sup>		CHSEL <sup>(1)</sup>	CBDMAEN <sup>(1)</sup>	PTMSK <sup>(1)</sup>	CUMSK <sup>(1)</sup>	VMSK <sup>(1)</sup>	PMSK <sup>(1)</sup>	DRFMT[1:0] <sup>(1)</sup>		RXSTEO <sup>(1)</sup>	RXDMAEN <sup>(1)</sup>	SPDIFRXEN[1:0] <sup>(1)</sup>	
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Refer to [Section 27.3.13: Register protection](#) for additional information on fields properties.

Bits 31:22 Reserved, forced by hardware to 0.

Bit 21 **CKSBKPEN**: Backup Symbol Clock Enable

This bit is set/reset by software

- 1: The SPDIFRX generates a backup symbol clock if CKSEN = '1'
- 0: The SPDIFRX does not generate a backup symbol clock

Bit 20 **CKSEN**: Symbol Clock Enable

This bit is set/reset by software

- 1: The SPDIFRX generates a symbol clock
- 0: The SPDIFRX does not generate a symbol clock

Bit 19 Reserved, forced by hardware to 0.par

Bits18:16 **INSEL[2:0]**: SPDIFRX input selection

0b000: SPDIFRX\_IN1 selected

0b001: SPDIFRX\_IN2 selected

0b010: SPDIFRX\_IN3 selected

0b011: SPDIFRX\_IN4 selected

others reserved

Bit 15 Reserved, forced by hardware to 0.

Bit 14 **WFA**: Wait For Activity

This bit is set/reset by software

- 1: The SPDIFRX waits for activity on SPDIFRX\_IN line (4 transitions) before performing the synchronization
- 0: The SPDIFRX does not wait for activity on SPDIFRX\_IN line before performing the synchronization

Bits 13:12 **NBTR[1:0]**: Maximum allowed re-tries during synchronization phase

- 0b00: No re-try is allowed (only one attempt)
- 0b01: 3 re-tries allowed
- 0b10: 15 re-tries allowed
- 0b11: 63 re-tries allowed

Bit 11 **CHSEL**: Channel Selection

- This bit is set/reset by software
- 1: The control flow will take the channel status from channel B
- 0: The control flow will take the channel status from channel A

Bit 10 **CBDMAEN**: Control Buffer DMA ENable for control flow

- This bit is set/reset by software
- 1: DMA mode is enabled for reception of channel status and used data information.
- 0: DMA mode is disabled for reception of channel status and used data information.

When this bit is set, the DMA request is made whenever the CSRNE flag is set.

Bit 9 **PTMSK**: Mask of Preamble Type bits

- This bit is set/reset by software
- 1: The preamble type bits are not copied into the SPDIFRX\_DR, zeros are written instead
- 0: The preamble type bits are copied into the SPDIFRX\_DR

Bit 8 **CUMSK**: Mask of channel status and user bits

- This bit is set/reset by software
- 1: The channel status and user bits are not copied into the SPDIFRX\_DR, zeros are written instead
- 0: The channel status and user bits are copied into the SPDIFRX\_DR

Bit 7 **VMSK**: Mask of Validity bit

- This bit is set/reset by software
- 1: The validity bit is not copied into the SPDIFRX\_DR, a zero is written instead
- 0: The validity bit is copied into the SPDIFRX\_DR

Bit 6 **PMSK**: Mask Parity error bit

- This bit is set/reset by software
- 1: The parity error bit is not copied into the SPDIFRX\_DR, a zero is written instead
- 0: The parity error bit is copied into the SPDIFRX\_DR

Bits 5:4 **DRFMT[1:0]**: RX Data format

- This bit is set/reset by software
- 0b11: reserved
- 0b10: Data sample are packed by setting two 16-bit sample into a 32-bit word
- 0b01: Data samples are aligned in the left (MSB)
- 0b00: Data samples are aligned in the right (LSB)

**Bit 3 RXSTEO:** STerEO Mode

This bit is set/reset by software

1: The peripheral is in STEREO mode

0: The peripheral is in MONO mode

This bit is used in case of overrun situation in order to handle misalignment

**Bit 2 RXDMAEN:** Receiver DMA ENable for data flow

This bit is set/reset by software

1: DMA mode is enabled for reception.

0: DMA mode is disabled for reception.

When this bit is set, the DMA request is made whenever the RXNE flag is set.

**Bits 1:0 SPDIFRXEN[1:0]:** Peripheral Block Enable

This field is modified by software.

It shall be used to change the peripheral phase among the three possible states: STATE\_IDLE, STATE\_SYNC and STATE\_RCV.

0b00: Disable SPDIFRX (STATE\_IDLE).

0b01: Enable SPDIFRX Synchronization only

0b10: Reserved

0b11: Enable SPDIF Receiver

- Note:*
- 1 *it is not possible to transition from STATE\_RCV to STATE\_SYNC, the user shall first go the STATE\_IDLE.*
  - 2 *it is possible to transition from STATE\_IDLE to STATE\_RCV: in that case the peripheral transitions from STATE\_IDLE to STATE\_SYNC and as soon as the synchronization is performed goes to STATE\_RCV.*

## 27.5.2 Interrupt mask register (SPDIFRX\_IMR)

Address offset: 0x04

Reset value: 0x00000000

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.	IFEIE	SYNCDIE	SBLKIE	OVRIE	PERRIE	CSRNEIE	RXNEIE								

Bits 31:7 Reserved, forced by hardware to 0.

Bit 6 **IFEIE**: Serial Interface Error Interrupt Enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A SPDIFRX interface interrupt is generated whenever SERR=1, TERR=1 or FERR=1 in the SPDIFRX\_SR register.

Bit 5 **SYNCDIE**: Synchronization Done

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A SPDIFRX interface interrupt is generated whenever SYNCD = 1 in the SPDIFRX\_SR register.

Bit 4 **SBLKIE**: Synchronization Block Detected Interrupt Enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A SPDIFRX interface interrupt is generated whenever SBD = 1 in the SPDIFRX\_SR register.

Bit 3 **OVRIE**: Overrun error Interrupt Enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A SPDIFRX interface interrupt is generated whenever OVR=1 in the SPDIFRX\_SR register

Bit 2 **PERRIE**: Parity error interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A SPDIFRX interface interrupt is generated whenever PERR=1 in the SPDIFRX\_SR register

Bit 1 **CSRNEIE**: Control Buffer Ready Interrupt Enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A SPDIFRX interface interrupt is generated whenever CSRNE = 1 in the SPDIFRX\_SR register.

Bit 0 **RXNEIE**: RXNE interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: A SPDIFRX interface interrupt is generated whenever RXNE=1 in the SPDIFRX\_SR register

### 27.5.3 Status register (SPDIFRX\_SR)

Address offset: 0x08

Reset value: 0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	WIDTH5[14:0]														
	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.	TERR	SERR	FERR	SYNCD	SBD	OVR	PERR	CSRNE	RXNE
							r	r	r	r	r	r	r	r	r

Bit 31 Reserved, forced by hardware to 0.

Bits 30:16 **WIDTH5[14:0]**: Duration of 5 symbols counted with SPDIFRX\_CLK

This value represents the amount of SPDIFRX\_CLK clock periods contained on a length of 5 consecutive symbols. This value can be used to estimate the S/PDIF symbol rate. Its accuracy is limited by the frequency of SPDIFRX\_CLK.

For example if the SPDIFRX\_CLK is fixed to 84 MHz, and WIDTH5 = 147d. The estimated sampling rate of the S/PDIF stream is:

$F_s = 5 \times F_{SPDIFRX\_CLK} / (\text{WIDTH5} \times 64) \sim 44.6 \text{ kHz}$ , so the closest standard sampling rate is 44.1 kHz.

Note that WIDTH5 is updated by the hardware when SYNCD goes high, and then every frame.

Bits 15:9 Reserved, forced by hardware to 0.

Bit 8 **TERR**: Time-out error

This bit is set by hardware when the counter TRCNT reaches its max value. It indicates that the time interval between two transitions is too long. It generally indicates that there is no valid signal on SPDIFRX\_IN input.

This flag is cleared by writing SPDIFRXEN to 0

An interrupt is generated if IFEIE=1 in the SPDIFRX\_IMR register

0: No sequence error is detected

1: Sequence error is detected

Bit 7 **SERR**: Synchronization error

This bit is set by hardware when the synchronization fails due to amount of re-tries for NBTR.

This flag is cleared by writing SPDIFRXEN to 0

An interrupt is generated if IFEIE=1 in the SPDIFRX\_IMR register.

0: No synchronization error is detected

1: Synchronization error is detected

Bit 6 **FERR**: Framing error

This bit is set by hardware when an error occurs during data reception: preamble not at the expected place, short transition not grouped by pairs...

This is set by the hardware only if the synchronization has been completed (SYNCD = 1).

This flag is cleared by writing SPDIFRXEN to 0

An interrupt is generated if IFEIE=1 in the SPDIFRX\_IMR register.

0: no Manchester Violation detected

1: Manchester Violation detected

**Bit 5 SYNCED:** Synchronization Done

This bit is set by hardware when the initial synchronization phase is properly completed.

This flag is cleared by writing a 1 to its corresponding bit on SPDIFRX\_CLR\_SR register.

An interrupt is generated if SYNC DIE = 1 in the SPDIFRX\_IMR register

0: Synchronization is pending

1: Synchronization is completed

**Bit 4 SBD:** Synchronization Block Detected

This bit is set by hardware when a “B” preamble is detected

This flag is cleared by writing a 1 to its corresponding bit on SPDIFRX\_CLR\_SR register.

An interrupt is generated if SBLKIE = 1 in the SPDIFRX\_IMR register

0: No “B” preamble detected

1: “B” preamble has been detected

**Bit 3 OVR:** Overrun error

This bit is set by hardware when a received data is ready to be transferred in the SPDIFRX\_DR register while RXNE = 1 and both SPDIFRX\_DR and RX\_BUF are full.

This flag is cleared by writing a 1 to its corresponding bit on SPDIFRX\_CLR\_SR register.

An interrupt is generated if OVRIE=1 in the SPDIFRX\_IMR register.

0: No Overrun error

1: Overrun error is detected

*Note: When this bit is set, the SPDIFRX\_DR register content will not be lost but the last data received will.*

**Bit 2 PERR:** Parity error

This bit is set by hardware when the data and status bits of the sub-frame received contain an odd number of 0 and 1.

This flag is cleared by writing a 1 to its corresponding bit on SPDIFRX\_CLR\_SR register.

An interrupt is generated if PIE = 1 in the SPDIFRX\_IMR register.

0: No parity error

1: Parity error

**Bit 1 CSRNE:** The Control Buffer register is not empty

This bit is set by hardware when a valid control information is ready.

This flag is cleared when reading SPDIFRX\_CSR register.

An interrupt is generated if CBRDYIE = 1 in the SPDIFRX\_IMR register

0: No control word available on SPDIFRX\_CSR register

1: A control word is available on SPDIFRX\_CSR register

**Bit 0 RXNE:** Read data register not empty

This bit is set by hardware when a valid data is available into SPDIFRX\_DR register.

This flag is cleared by reading the SPDIFRX\_DR register.

An interrupt is generated if RXNEIE=1 in the SPDIFRX\_IMR register.

0: Data is not received

1: Received data is ready to be read.

### 27.5.4 Interrupt flag clear register (SPDIFRX\_IFCR)

Address offset: 0x0C

Reset value: 0x00000000

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.	SYNCDCF	SBDCF	OVRCF	PERRCF	Res.	Res.									
										w	w	w	w		

Bits 31:6 Reserved, forced by hardware to 0.

Bit 5 **SYNCDCF**: Clears the Synchronization Done flag

Writing 1 in this bit clears the flag SYNC in the SPDIFRX\_SR register.

Reading this bit always returns the value 0.

Bit 4 **SBDCF**: Clears the Synchronization Block Detected flag

Writing 1 in this bit clears the flag SBD in the SPDIFRX\_SR register.

Reading this bit always returns the value 0.

Bit 3 **OVRCF**: Clears the Overrun error flag

Writing 1 in this bit clears the flag OVR in the SPDIFRX\_SR register.

Reading this bit always returns the value 0.

Bit 2 **PERRCF**: Clears the Parity error flag

Writing 1 in this bit clears the flag PERR in the SPDIFRX\_SR register.

Reading this bit always returns the value 0.

Bits 1:0 Reserved

### 27.5.5 Data input register (SPDIFRX\_DR)

Address offset: 0x10

Reset value: 0x00000000

This register can take 3 different formats according to DRFMT. Here is the format when DRFMT = 0b00:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	PT[1:0]		C	U	V	PE	DR[23:16]							
		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
DR[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:30 Reserved: forced by hardware to 0

Bits 29:28 **PT**: Preamble Type

These bits indicate the preamble received.

00: not used

01: Preamble B received

10: Preamble M received

11: Preamble W received

Note that if PTMSK = 1, this field is forced to zero

Bit 27 **C**: Channel Status bit

Contains the received channel status bit, if CUMSK = 0, otherwise it is forced to 0

Bit 26 **U**: User bit

Contains the received user bit, if CUMSK = 0, otherwise it is forced to 0

Bit 25 **V**: Validity bit

Contains the received validity bit if VMSK = 0, otherwise it is forced to 0

Bit 24 **PE**: Parity Error bit

Contains a copy of PERR bit if PMSK = 0, otherwise it is forced to 0

Bits 23:0 **DR**: Data value

Contains the 24 received data bits, aligned on D[23]

### 27.5.6 Data input register (SPDIFRX\_DR)

Address offset: 0x10

Reset value: 0x00000000

This register can take 3 different formats according to DRFMT. Here is the format when DRFMT = 0b01:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DR[23:8]															
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
DR[7:0]								Res.	Res.	PT[1:0]		C	U	V	PE
r	r	r	r	r	r	r	r			r	r	r	r	r	r

Bits 31:8 **DR**: Data value

Contains the 24 received data bits, aligned on D[23]

Bits 7:6 Reserved: forced by hardware to 0

Bits 5:4 **PT**: Preamble Type

These bits indicate the preamble received.

00: not used

01: Preamble B received

10: Preamble M received

11: Preamble W received

Note that if PTMSK = 1, this field is forced to zero

Bit 3 **C**: Channel Status bit

Contains the received channel status bit, if CUMSK = 0, otherwise it is forced to 0

Bit 2 **U**: User bit

Contains the received user bit, if CUMSK = 0, otherwise it is forced to 0

Bit 1 **V**: Validity bit

Contains the received validity bit if VMSK = 0, otherwise it is forced to 0

Bit 0 **PE**: Parity Error bit

Contains a copy of PERR bit if PMSK = 0, otherwise it is forced to 0

### 27.5.7 Data input register (SPDIFRX\_DR)

Address offset: 0x10

Reset value: 0x00000000

This register can take 3 different formats according to DRFMT.

The data format proposed when DRFMT = 0b10, is dedicated to non-linear mode, as only 16 bits are used (bits 23 to 8 from S/PDIF sub-frame).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DRNL2[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
DRNL1[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 **DRNL2**: Data value

This field contains the Channel A

Bits 15:0 **DRNL1**: Data value

This field contains the Channel B

## 27.5.8 Channel status register (SPDIFRX\_CSR)

Address offset: 0x14

Reset value: 0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	SOB	CS[7:0]							
							r	r	r	r	r	r	r	r	r
USR[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:25 Reserved

Bit 24 **SOB**: Start Of Block

This bit indicates if the bit CS[0] corresponds to the first bit of a new block

0: CS[0] is not the first bit of a new block

1: CS[0] is the first bit of a new block

Bits 23:16 **CS[7:0]**: Channel A status information

Bit CS[0] is the oldest value

Bits 15:0 **USR[15:0]**: User data information

Bit USR[0] is the oldest value, and comes from channel A, USR[1] comes channel B.

So USR[n] bits come from channel A if n is even, otherwise they come from channel B.

### 27.5.9 Debug Information register (SPDIFRX\_DIR)

Address offset: 0x18

Reset value: 0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Res.	Res.	Res.	TLO[12:0]													
			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.	THI[12:0]													
			r	r	r	r	r	r	r	r	r	r	r	r	r	

Bits 31:29 Reserved, forced by hardware to 0.

Bits 16:28 **TLO**: Threshold LOW ( $TLO = 1.5 \times UI / T_{SPDIFRX\_CLK}$ )

This field contains the current threshold LOW estimation. This value can be used to estimate the sampling rate of the received stream. The accuracy of TLO is limited to a period of the SPDIFRX\_CLK. The sampling rate can be estimated as follow:

Sampling Rate =  $[2 \times TLO \times T_{SPDIFRX\_CLK} +/- T_{SPDIFRX\_CLK}] \times 2/3$

Note that TLO is updated by the hardware when SYNC'D goes high, and then every frame.

Bits 15:13 Reserved, forced by hardware to 0.

Bits 12:0 **THI**: Threshold HIGH ( $THI = 2.5 \times UI / T_{SPDIFRX\_CLK}$ )

This field contains the current threshold HIGH estimation. This value can be used to estimate the sampling rate of the received stream. The accuracy of THI is limited to a period of the SPDIFRX\_CLK. The sampling rate can be estimated as follow:

Sampling Rate =  $[2 \times THI \times T_{SPDIFRX\_CLK} +/- T_{SPDIFRX\_CLK}] \times 2/5$

Note that THI is updated by the hardware when SYNC'D goes high, and then every frame.

### 27.5.10 SPDIFRX version register (SPDIFRX\_VERR)

Address offset: 0x03F4

Reset value: 0x0000 0012

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

Bits 31:8 Reserved, must be kept at reset value

Bits 7:4 **MAJREV[3:0]**: Major revision

These bits return the SPDIFRX major revision.

Major revision is 1.

Bits 3:0 **MINREV[3:0]**: Minor revision

These bits return the SPDIFRX minor revision.

Minor revision is 2.

### 27.5.11 SPDIFRX identification register (SPDIFRX\_IDR)

Address offset: 0x03F8

Reset value: 0x0013 0041

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

Bits 31:0 **ID[31:0]**: SPDIFRX identifier

These bits return the SPDIFRX identifier value.

### 27.5.12 SPDIFRX size identification register (SPDIFRX\_SIDR)

Address offset: 0x03FC

Reset value: 0xA3C5 DD01

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

Bits 31:0 **SID[31:0]**: Size identification

These bits return the size of the memory region allocated to SPDIFRX registers.

The size of this memory region is of 1 Kbyte.

### **27.5.13 SPDIFRX interface register map**

[Table 168](#) gives the SPDIFRX interface register map and reset values.

**Table 168. SPDIFRX interface register map and reset values**

## 28 Serial audio interface (SAI)

### 28.1 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.

## 28.2 SAI main features

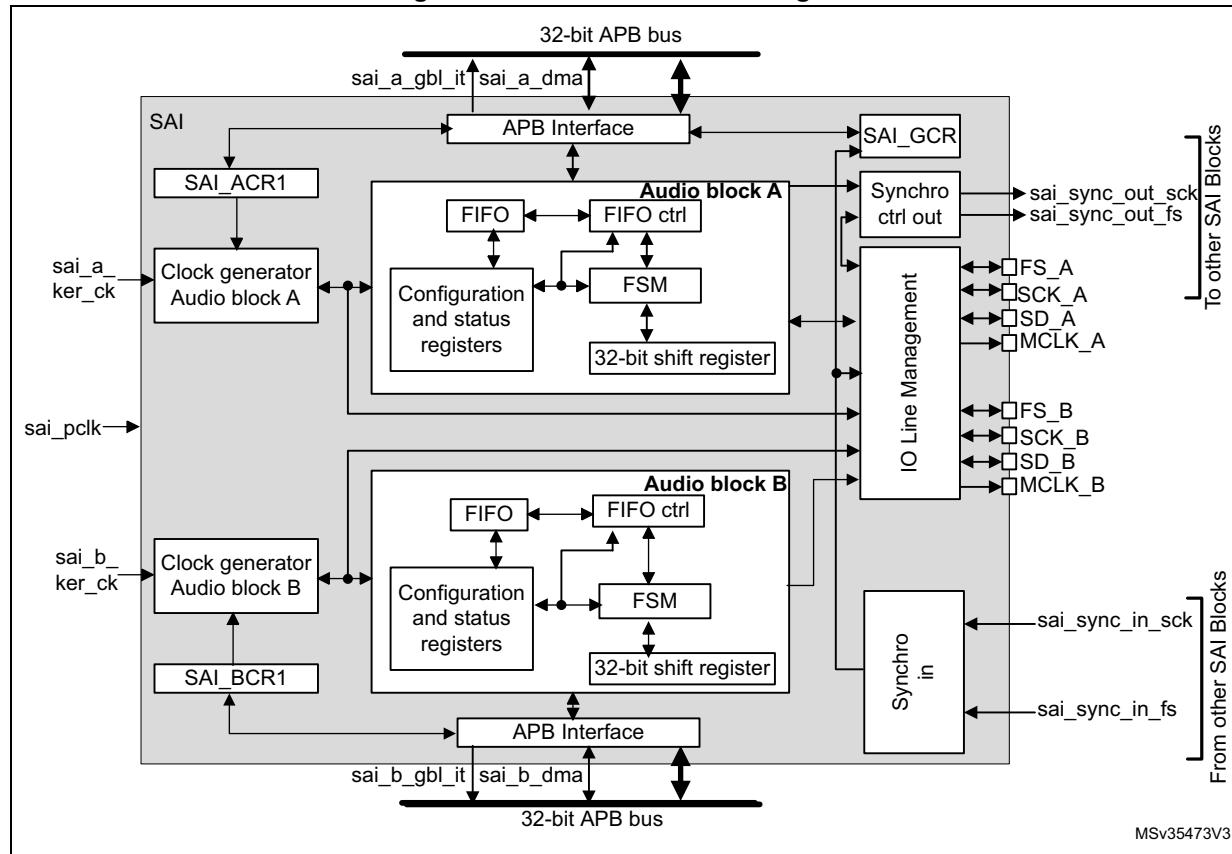
- Two independent audio sub-blocks which can be transmitters or receivers with their respective FIFO.
- 8-word integrated FIFOs for each audio sub-block.
- Synchronous or asynchronous mode between the audio sub-blocks.
- Possible synchronization between multiple SAIs.
- Master or slave configuration 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.

## 28.3 SAI functional description

### 28.3.1 SAI block diagram

*Figure 353* shows the SAI block diagram while *Table 169* and *Table 170* list SAI internal and external signals.

**Figure 353. 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.*

### 28.3.2 SAI pins and internal signals

**Table 169. SAI internal signals**

Internal signal name	Signal type	Description
sai_a_gbl_it/ sai_b_gbl_it	Digital output	Audio block A and B global interrupts.
sai_a_dma, sai_b_dma	Digital input/output	Audio block A and B DMA acknowledges and requests.
sai_sync_out_sck, sai_sync_out_fs	Digital output	Internal clock and frame synchronization output signals exchanged with other SAI blocks.
sai_sync_in_sck, sai_sync_in_fs	Digital input	Internal clock and frame synchronization input signals exchanged with other SAI blocks.
sai_a_ker_ck/ sai_b_ker_ck	Digital input	Audio block A/B kernel clock.
sai_pclk	Digital input	APB clock.

**Table 170. SAI pins**

Name	Signal type	Comments
SAI_SCK_A/B	Digital input/output	Audio block A/B bit clock.
SAI_MCLK_A/B	Digital output	Audio block A/B master clock.
SAI_SD_A/B	Digital input/output	Data line for block A/B.
SAI_FS_A/B	Digital input/output	Frame synchronization line for audio block A/B.

### 28.3.3 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 bit SAIXEN 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 000b (FIFO empty).
3. Enable the audio block in slave transmitter mode.

### 28.3.4 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 171. 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

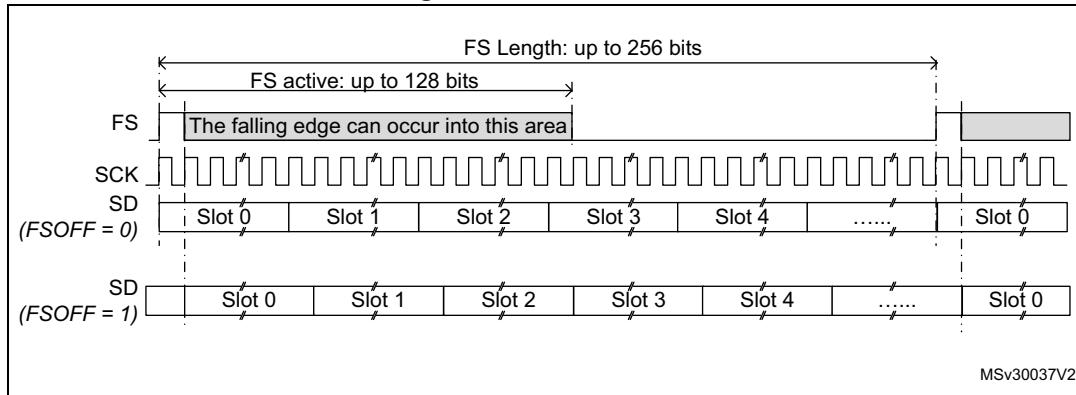
### 28.3.5 Audio data size

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.

### 28.3.6 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 354](#) illustrates this flexibility.

**Figure 354. Audio frame**



In AC'97 mode or in SPDIF mode (bit PRTC[1:0] = 10 or PRTC[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 [Section : 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 28.3.8: 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 28.3.13: 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 28.3.13: Error flags](#)).

In master mode, the frame synchronization is sent continuously each time an audio frame is complete until the SAIXEN 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 28.3.13: 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.

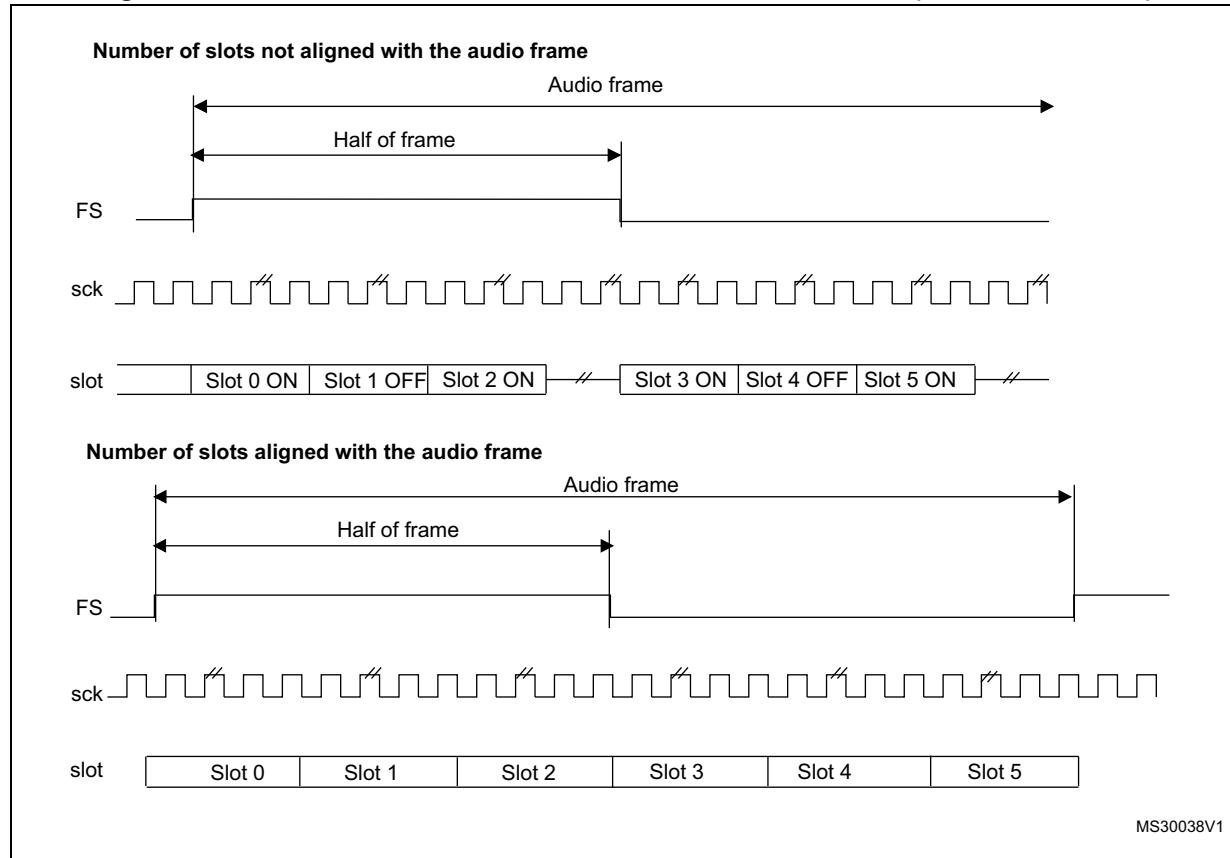
### FS signal role

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.

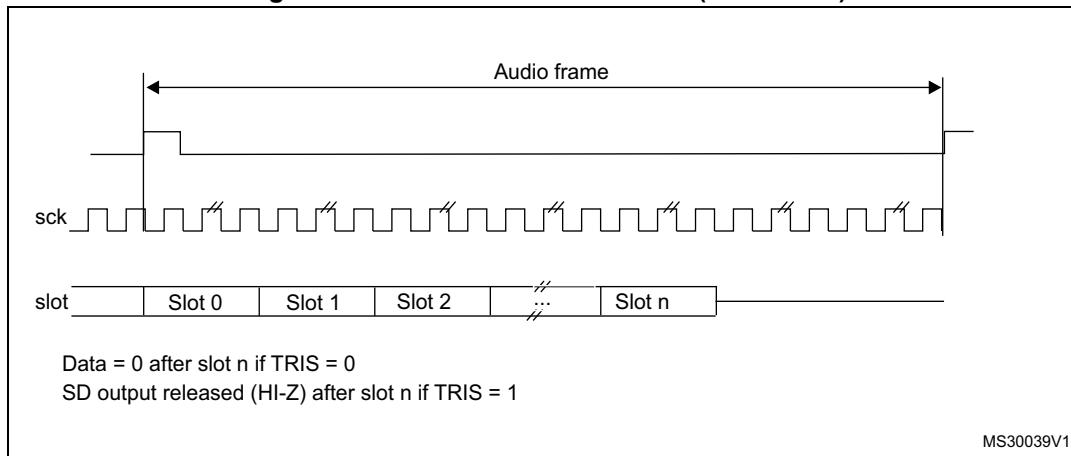
Otherwise if TRIS = 1, the SD line is released to HI-Z. In reception mode, the remaining bit clock cycles are not considered until the channel side changes.

**Figure 355. FS role is start of frame + channel side identification (FSDEF = TRIS = 1)**

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 356. 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.

### 28.3.7 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.

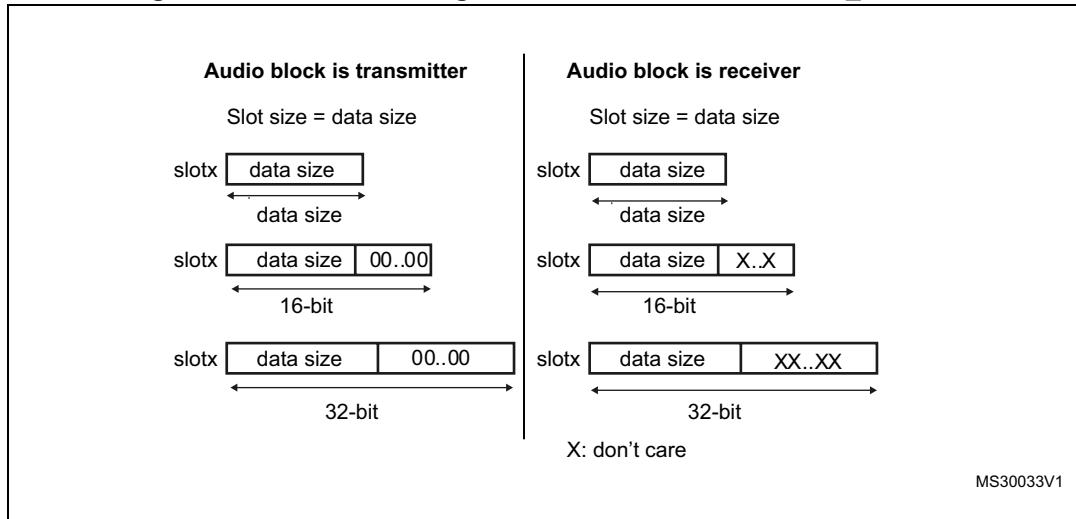
The maximum number of slots per audio frame is fixed at 16.

For AC'97 protocol or SPDIF (when bit PRTC<sub>FG</sub>[1:0] = 10 or PRTC<sub>FG</sub>[1:0] = 01), the number of slots is automatically set to target the protocol specification, and the value of NBSLOT[3:0] is ignored.

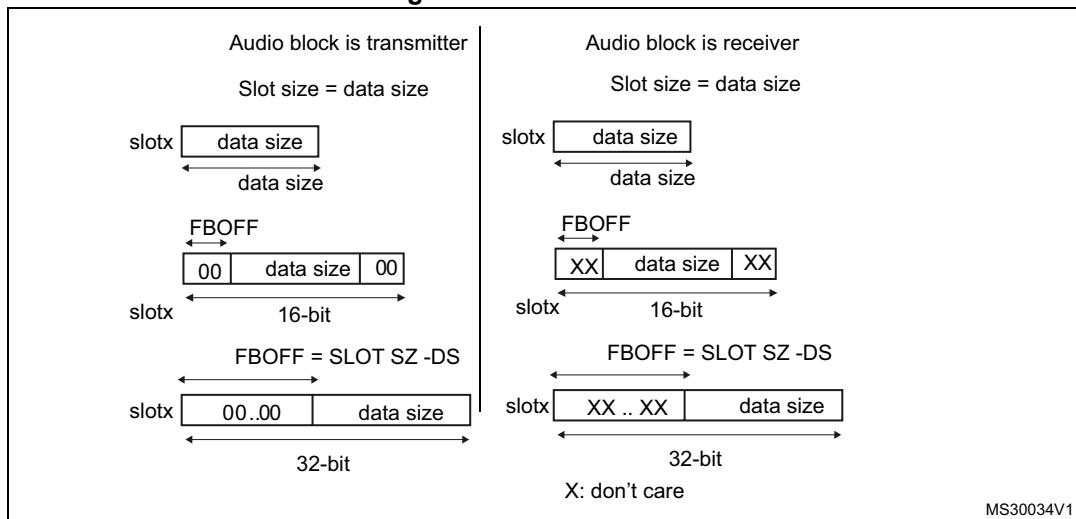
Each slot can be defined as a valid slot, or not, by setting SLOTEN[15:0] bits of the SAI\_xSLOTR register.

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 [Section : 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 357](#). 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 357. 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 358. First bit offset**

It is mandatory to respect the following conditions to avoid bad SAI behavior:

$$\text{FBOFF} \leq (\text{SLOTSZ} - \text{DS}),$$

$$\text{DS} \leq \text{SLOTSZ},$$

$$\text{NBSLOT} \times \text{SLOTSZ} \leq \text{FRL} \text{ (frame length)},$$

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 28.3.10: AC'97 link controller](#).

### 28.3.8 SAI clock generator

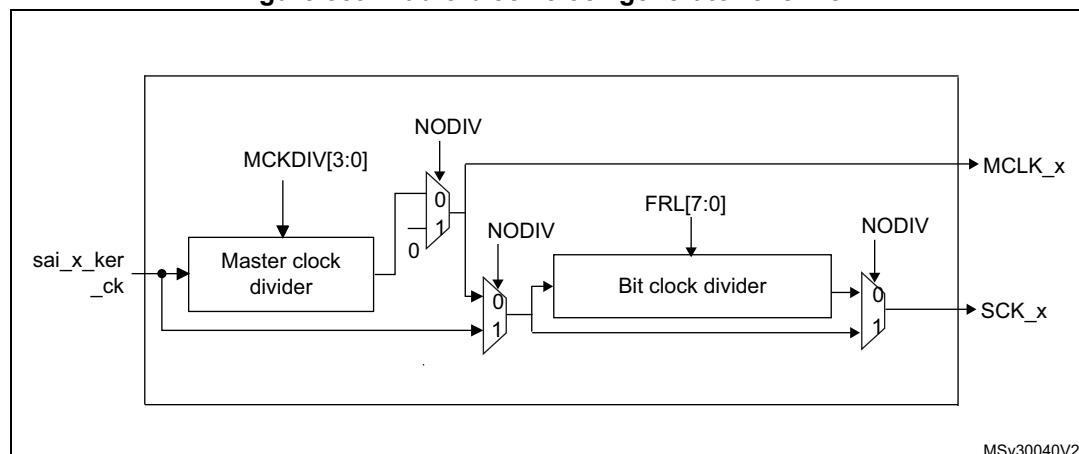
Each audio block has its own clock generator that makes these two blocks completely independent. There is no difference in terms of functionality between these two clock generators.

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 359* illustrates the architecture of the audio block clock generator.

**Figure 359. 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 172*.

**Table 172. Example of possible audio frequency sampling range**

<b>Input sai_x_ker_ck clock frequency</b>	<b>Most usual audio frequency sampling achievable</b>	<b>MCKDIV[3:0]</b>
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 <sup>(1)</sup>	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 28.3.6: 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 28.3.11: SPDIF output](#) for details on clock generator programming in SPDIF mode.

### 28.3.9 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 [Section : Interrupt generation in transmitter mode](#) and [Section : 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 000b), 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 000b) 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 010b). 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 010b).
- When the FIFO threshold bits in SAI\_xCR2 register are configured as FIFO half full (FTH[2:0] set to 010b), 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 100b). 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 100b).
- When the FIFO threshold bits in SAI\_xCR2 register are configured as FIFO full (FTH[2:0] set to 100b), 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 000b), 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 000b) 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 010b). 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 010b).
- When the FIFO threshold bits in SAI\_xCR2 register are configured as FIFO half fully (FTH[2:0] set to 010b 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 100b). 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 100b).
- When the FIFO threshold bits in SAI\_xCR2 register are configured as FIFO full(FTH[2:0] set to 100b), 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.

### 28.3.10 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.

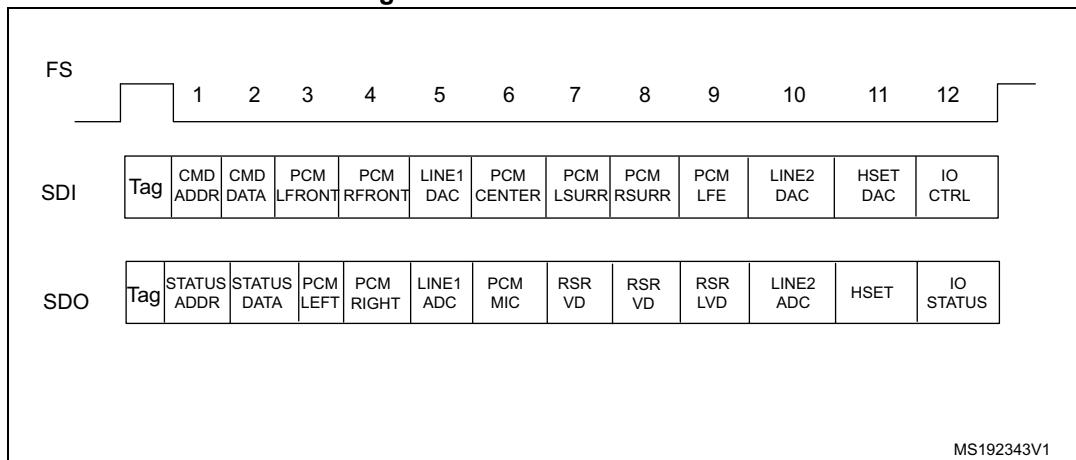
To select this protocol, set PRTC<sub>FG</sub>[1:0] bits in the SAI\_xCR1 register to 10. When AC'97 mode is selected, only data sizes of 16 or 20 bits can be used, otherwise the SAI behavior is not guaranteed.

- 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 360* shows an AC'97 audio frame structure.

**Figure 360. 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.

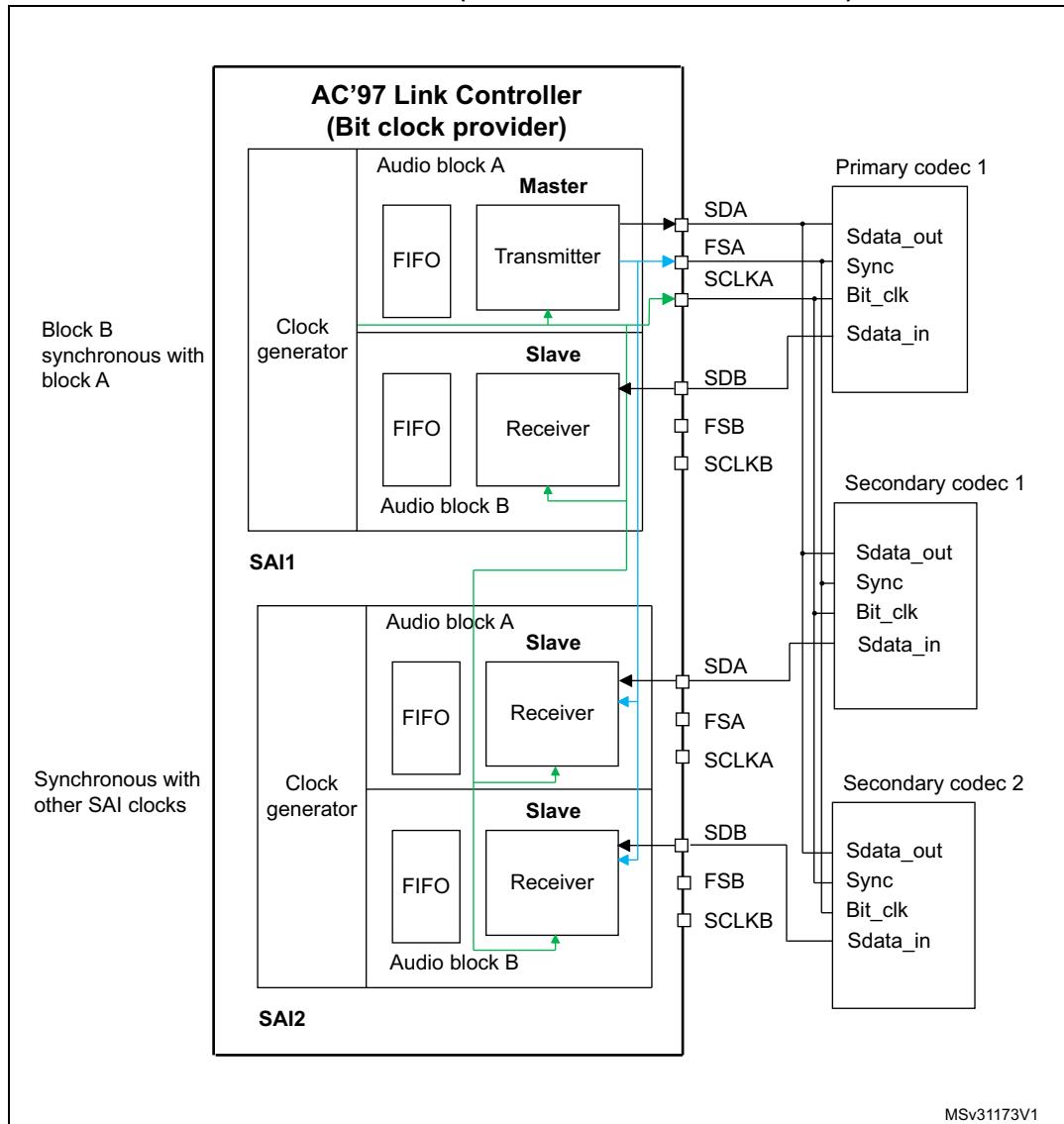
One SAI can be used to target an AC'97 point-to-point communication.

Using two SAIs (for devices featuring two embedded SAIs) allows controlling three external AC'97 decoders as illustrated in *Figure 361*.

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 361. Example of typical AC'97 configuration on devices featuring at least 2 embedded SAIs (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 28.3.8: SAI clock generator](#) shall be used with  $F_{FRL} = 255$ , in order to generate the proper frame rate ( $F_{FS\_x}$ ).

### 28.3.11 SPDIF output

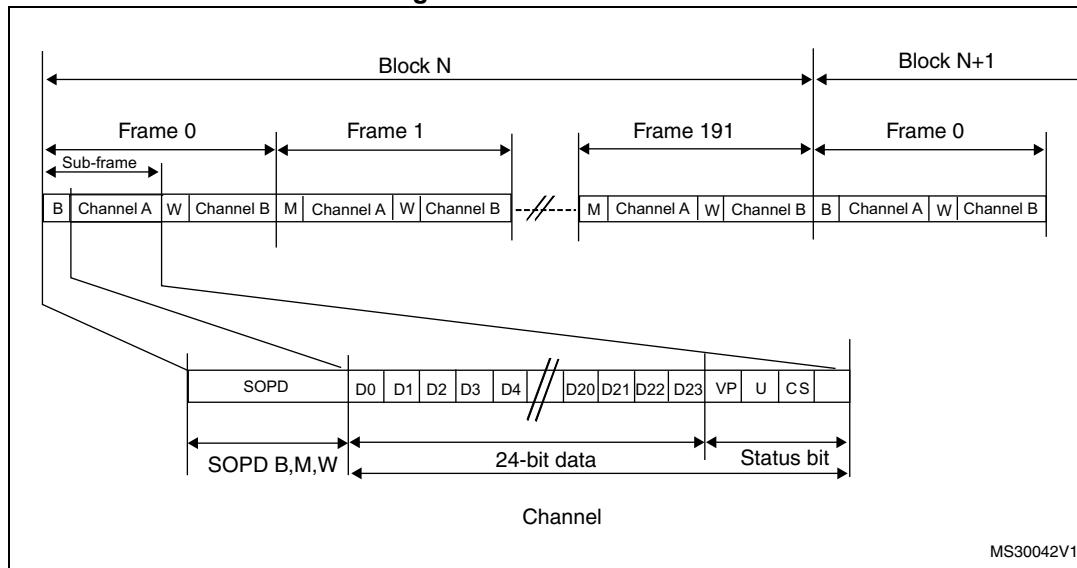
The SPDIF interface is available in transmitter mode only. It supports the audio IEC60958.

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 362](#).

**Figure 362. 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 173](#)). The next 28 bits of channel information are composed of 24 bits data + 4 status bits.

**Table 173. 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

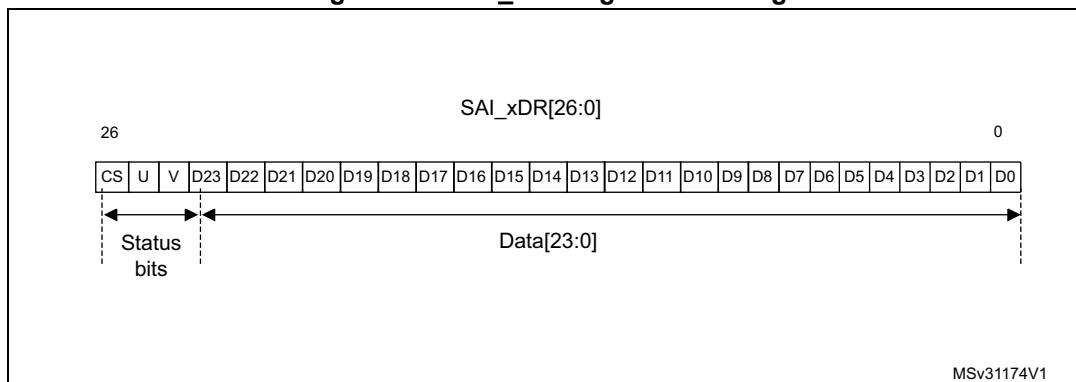
The data stored in SAI\_xDR has to be filled as follows:

- 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 363. 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 174](#).

**Table 174. 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 SAIXEN 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 SAIXEN bit in SAI\_xCR1 register.

### Clock generator programming in SPDIF generator mode

For the SPDIF generator, the SAI shall provide a bit clock equal to the symbol-rate. The table hereafter shows usual examples of symbol rates with respect to the audio sampling rate.

**Table 175. 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 rate ( $F_S$ ) and the bit-clock rate ( $F_{SCK\_X}$ ) is given by the formula:

#### 28.3.12 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 364](#). The two companding modes supported are the  $\mu$ -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  $\mu$ -Law. It supports 14 bits of dynamic range (COMP[1:0] = 10 in the SAI\_xCR2 register).

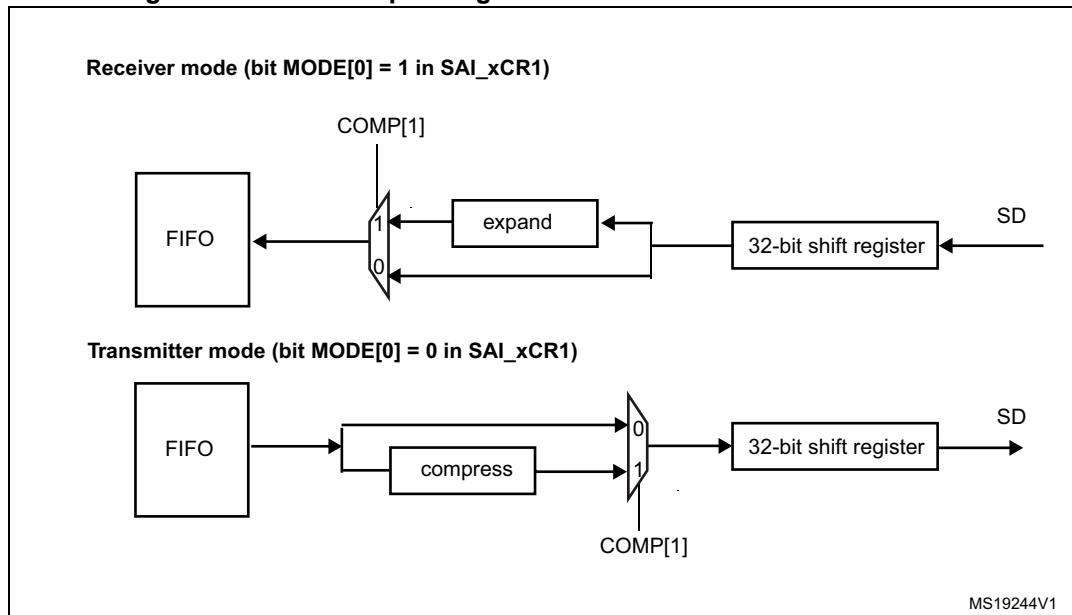
The European companding standard is A-Law and supports 13 bits of dynamic range (COMP[1:0] = 11 in the SAI\_xCR2 register).

Both  $\mu$ -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  $\mu$ -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 (bit SAIEN = 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 364. 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.

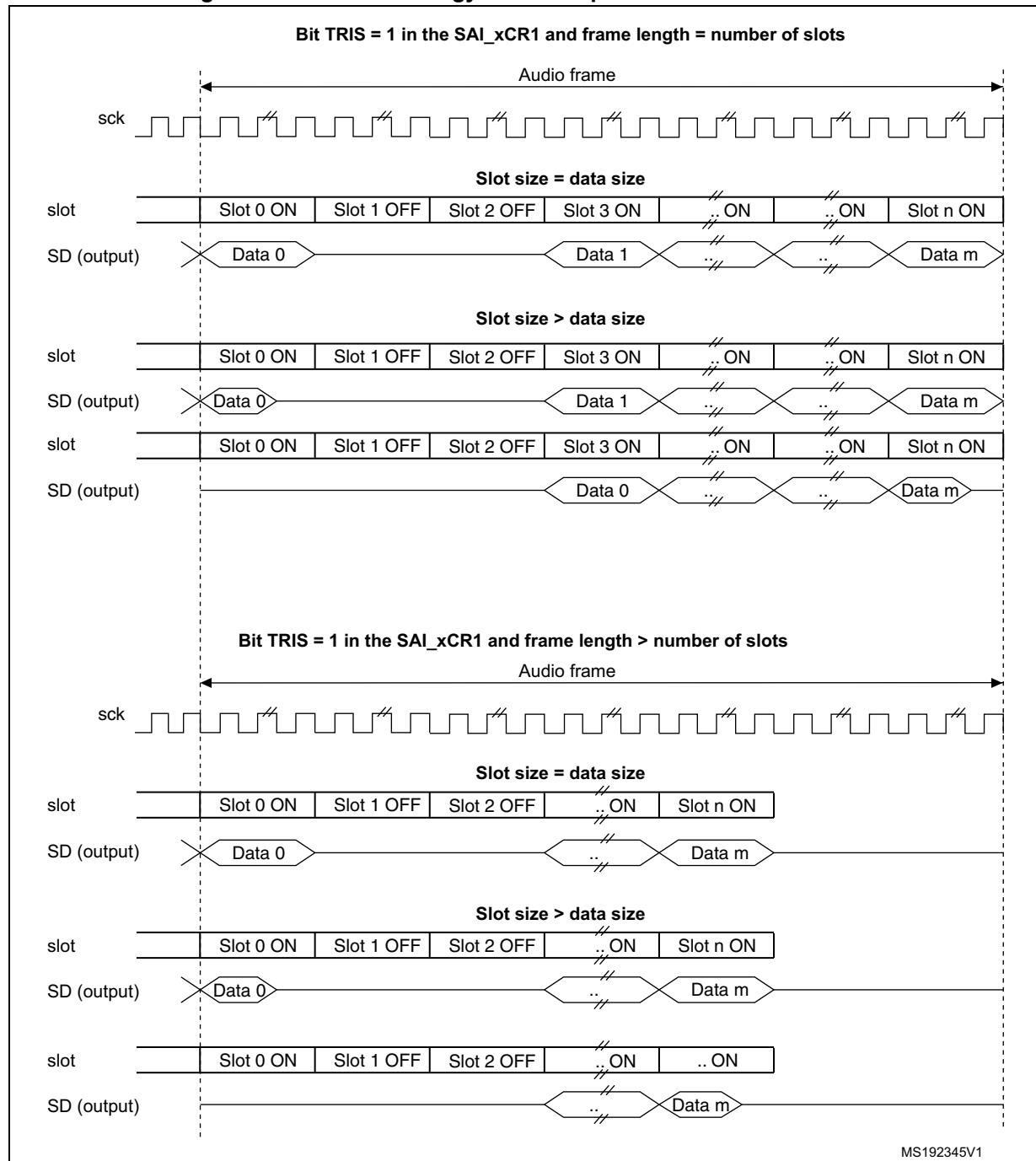
It is important to note that the two transmitters cannot attempt to drive the same SD output pin simultaneously, which could result in a short circuit. To ensure a gap between transmissions, if the data is lower than 32-bit, the data can be extended to 32-bit by setting bit SLOTSZ[1:0] = 10 in the SAI\_xSLOTR 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.

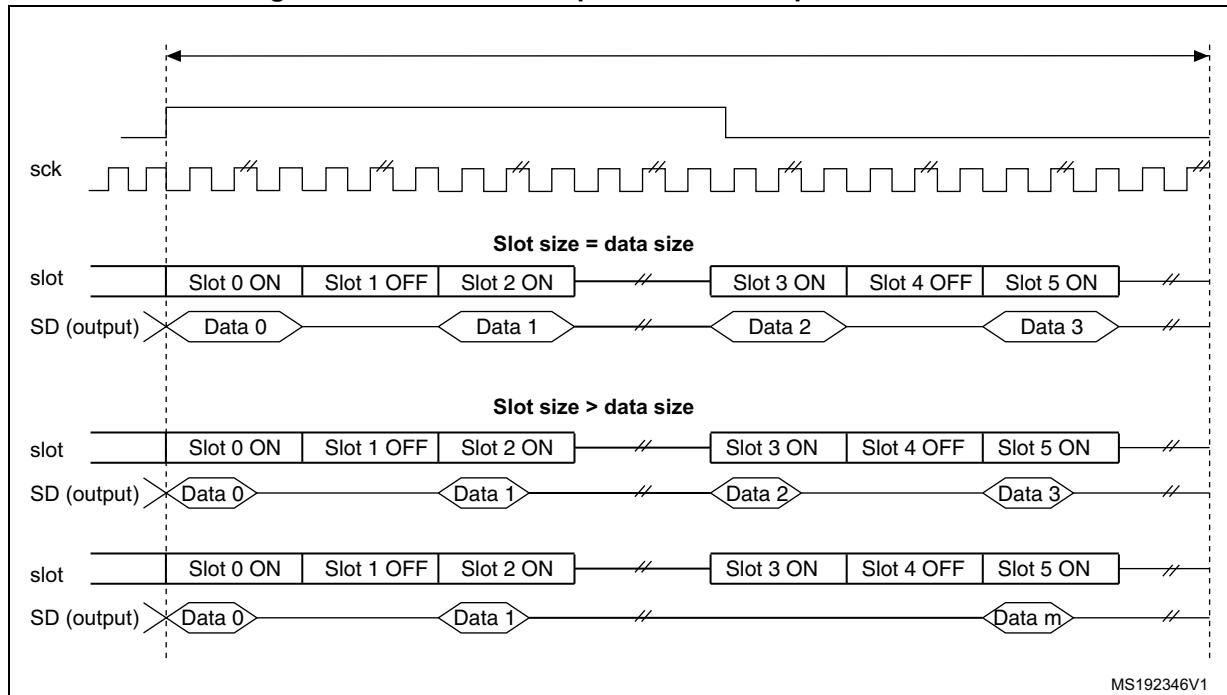
*Figure 365* illustrates these behaviors.

**Figure 365. 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 366](#) (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 366. 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 365](#) and [Figure 366](#) are replaced by a drive with a value of 0.

### 28.3.13 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.

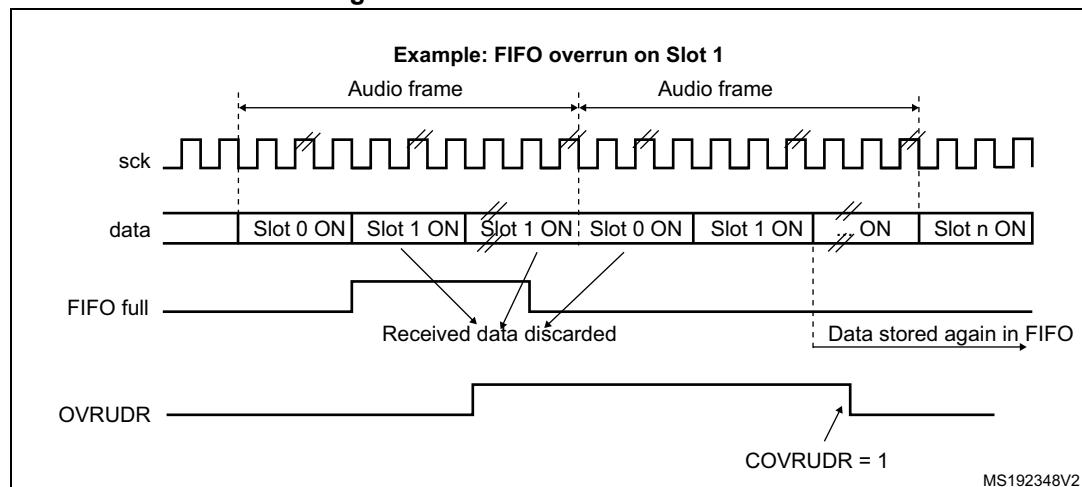
#### Overrun

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 367](#)).

The OVRUDR flag is cleared when COVRUDR bit is set in the SAI\_xCLRFR register.

**Figure 367. Overrun detection error**



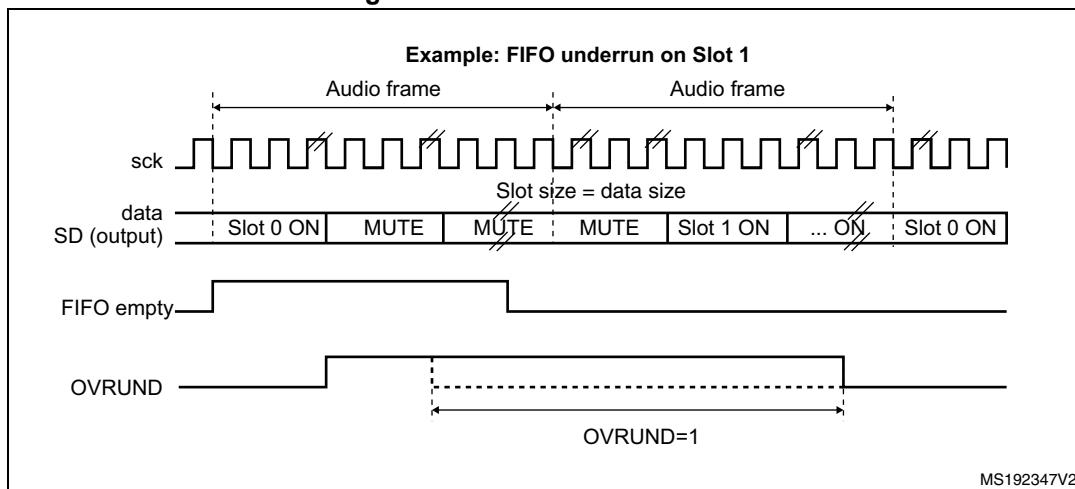
### Underrun

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 368](#)). 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 368. 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 SAIXEN bit in SAI\_xCR1 register. To make sure the SAI is disabled, read back the SAIXEN 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 (SAIXEN bit set to 1).
4. The SAI block will wait for the assertion on FS to restart the synchronization with master.

**Note:**

*The SAIXEN 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 [Section : 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 (PRTCFCFG[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 SAIXEN 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 SAIXEN bit.

### 28.3.14 Disabling the SAI

The SAI audio block can be disabled at any moment by clearing SAIXEN bit in the SAI\_xCR1 register. All the already started frames are automatically completed before the SAI stops working. SAIXEN 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.

### 28.3.15 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 28.3.9: 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 is 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.

## 28.4 SAI interrupts

The SAI supports 7 interrupt sources as shown in [Table 176](#).

**Table 176. 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 <a href="#">Section 28.3.9: Internal FIFOs</a>
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

**Table 176. SAI interrupt sources (continued)**

<b>Interrupt source</b>	<b>Interrupt group</b>	<b>Audio block mode</b>	<b>Interrupt enable</b>	<b>Interrupt clear</b>
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.

## 28.5 SAI registers

### 28.5.1 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 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).

### 28.5.2 Configuration register 1 (SAI\_ACR1 / SAI\_BCR1)

Address offset: Block A: 0x004

Address offset: Block B: 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	SAIX 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.	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 **SAIXEN**: Audio block enable where x is A or B

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 SAIx 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: SAIx audio block disabled

1: SAIx audio block enabled.

*Note: When SAIx block is configured in master mode, the clock must be present on the input of SAIx before setting SAIXEN 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 SAIXEN 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 PRTCFCG[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 **PRTCFG[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.*

### 28.5.3 Configuration register 2 (SAI\_ACR2 / SAI\_BCR2)

Address offset: Block A: 0x008

Address offset: Block B: 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]						MUTE VAL	MUTE	TRIS	F FLUSH	FTH			
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  $\mu$ -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:  $\mu$ -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  $\mu$ -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**: 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

#### 28.5.4 Frame configuration register (SAI\_AFRCR / SAI\_BFRCR)

Address offset: Block A: 0x00C

Address offset: Block B: 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.

### 28.5.5 Slot register (SAI\_ASLOTR / SAI\_BSLOTR)

Address offset: Block A: 0x010

Address offset: Block B: 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 1 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.

**28.5.6 Interrupt mask register 2 (SAI\_AIM / SAI\_BIM)**

Address offset: block A: 0x014

Address offset: block B: 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 operates 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.

## 28.5.7 Status register (SAI\_ASR / SAI\_BSR)

Address offset: block A: 0x018

Address offset: block B: 0x038

Reset value: 0x0000 0008

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	FLVL			
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.	AFSDET	CNRDY	FREQ	WCKCFG	MUTEDET	OVRUDR	r	r	r										
											r	r	r	r	r	r			

Bits 31:19 Reserved, must be kept at reset value

Bits 18:16 **FLVL**: 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 28.3.6: 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.

## 28.5.8 Clear flag register (SAI\_ACLRFR / SAI\_BCLRFR)

Address offset: block A: 0x01C

Address offset: block B: 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	CAFSDT	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.

## 28.5.9 Data register (SAI\_ADR / SAI\_BDR)

Address offset: block A: 0x020

Address offset: block B: 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.

### 28.5.10 SAI register map

The following table summarizes the SAI registers.

**Table 177. SAI register map and reset values**

Offset	Register and 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.
	Reset value	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.
	Reset value	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.
	Reset value	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.
	Reset value	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]															FBOFF[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
0x0014 or 0x0034	SAI_xIM	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
0x0018 or 0x0038	SAI_xSR	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
0x001C or 0x003C	SAI_xCLRFR	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

**Table 177. SAI register map and reset values (continued)**

Offset	Register and 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
0x0020 or 0x0040	SAI_xDR																																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Refer to [Section 2.2 on page 56](#) for the register boundary addresses.

## 29 Secure digital input/output interface (SDIO)

### 29.1 SDIO main features

The SD/SDIO MMC card host interface (SDIO) 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 SDIO 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 SDIO 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.

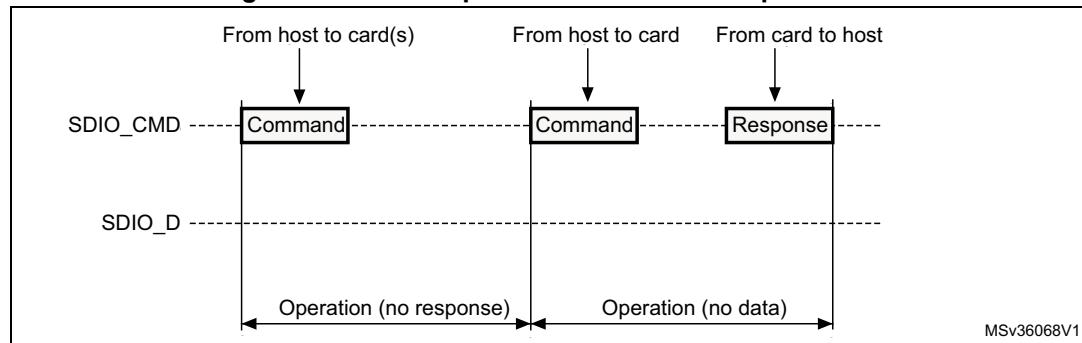
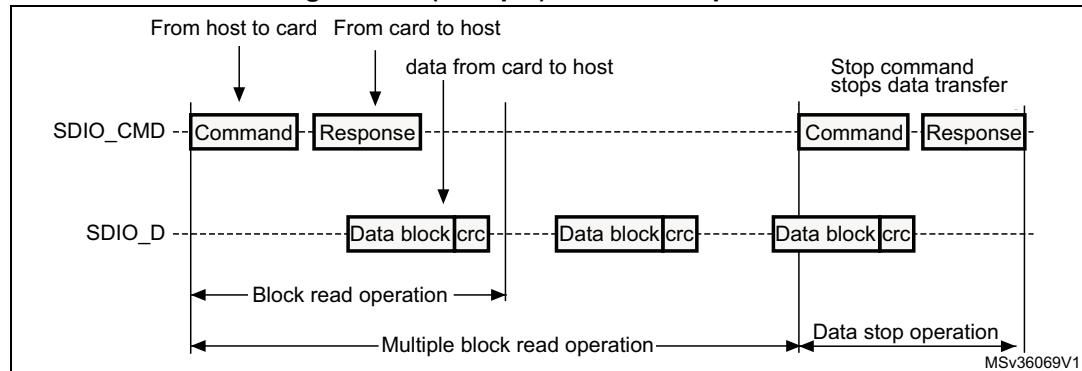
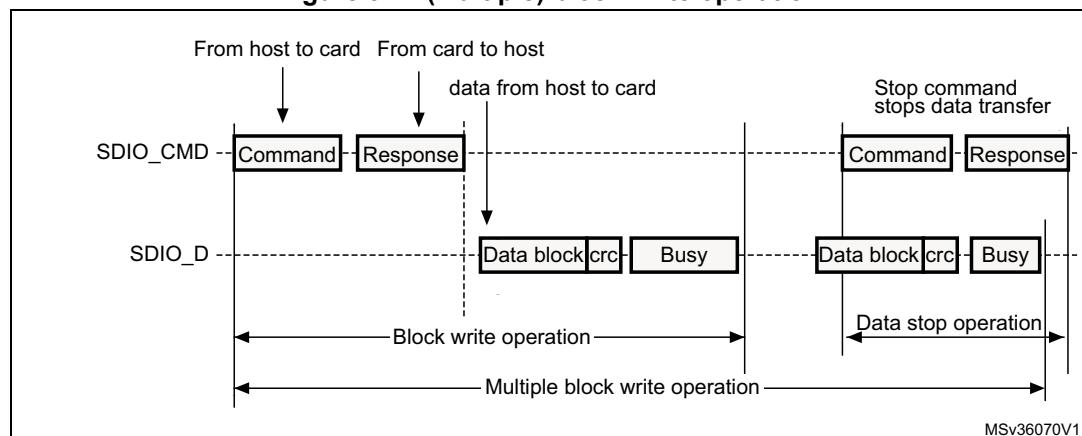
The current version of the SDIO supports only one SD/SDIO/MMC4.2 card at any one time and a stack of MMC4.1 or previous.

### 29.2 SDIO bus topology

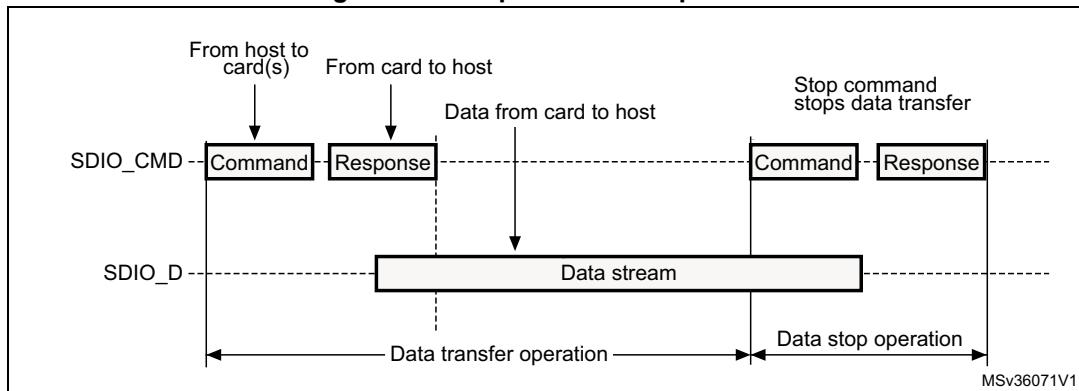
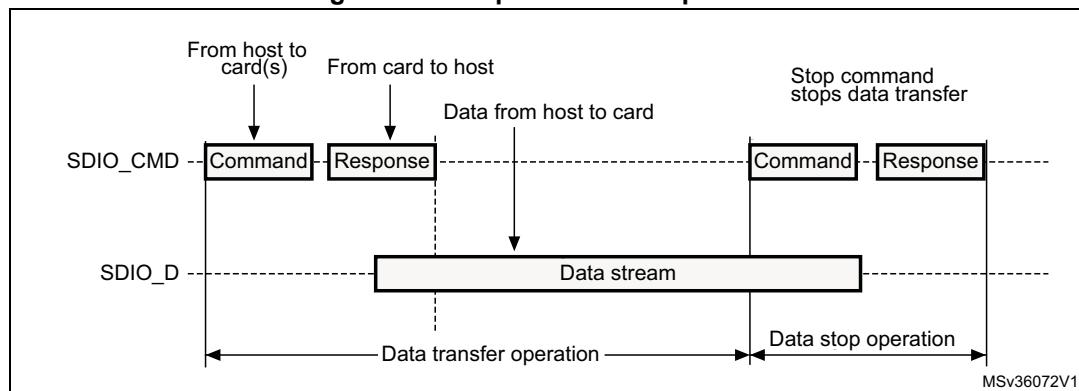
Communication over the bus is based on command and data transfers.

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 369. “No response” and “no data” operations****Figure 370. (Multiple) block read operation****Figure 371. (Multiple) block write operation**

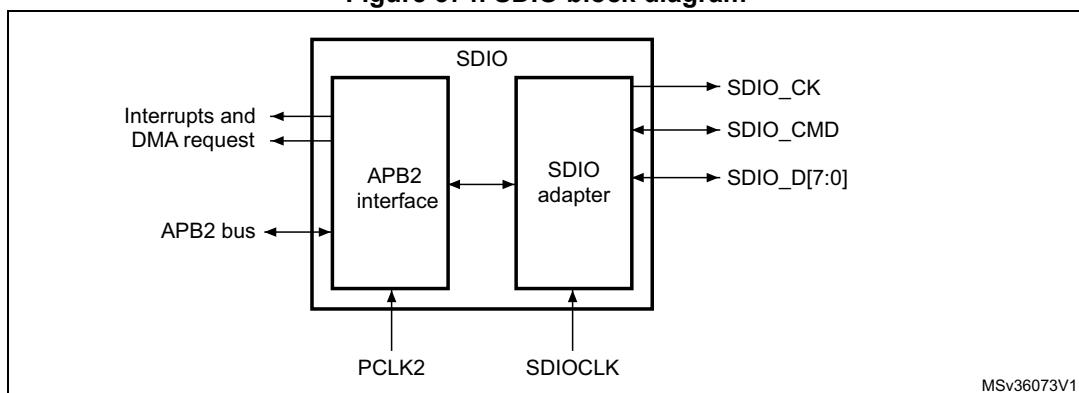
**Note:** The SDIO will not send any data as long as the Busy signal is asserted (SDIO\_D0 pulled low).

**Figure 372. Sequential read operation****Figure 373. Sequential write operation**

## 29.3 SDIO functional description

The SDIO consists of two parts:

- The SDIO 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 SDIO adapter registers, and generates interrupt and DMA request signals.

**Figure 374. SDIO block diagram**

By default SDIO\_D0 is used for data transfer. After initialization, the host can change the databus width.

If a MultiMediaCard is connected to the bus, SDIO\_D0, SDIO\_D[3:0] or SDIO\_D[7:0] can be used for data transfer. MMC V3.31 or previous, supports only 1 bit of data so only SDIO\_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 SDIO\_D0 or SDIO\_D[3:0]. All data lines are operating in push-pull mode.

**SDIO\_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)

**SDIO\_CK** is the clock to the card: one bit is transferred on both command and data lines with each clock cycle.

The SDIO uses two clock signals:

- SDIO adapter clock SDIOCLK = 50 MHz)
- APB2 bus clock (PCLK2)

PCLK2 and SDIO\_CK clock frequencies must respect the following condition:

$$\text{Frequenc(PCLK2)} > ((3 \times \text{Width}) / 32) \times \text{Frequency(SDIO_CK)}$$

The signals shown in [Table 178](#) are used on the MultiMediaCard/SD/SD I/O card bus.

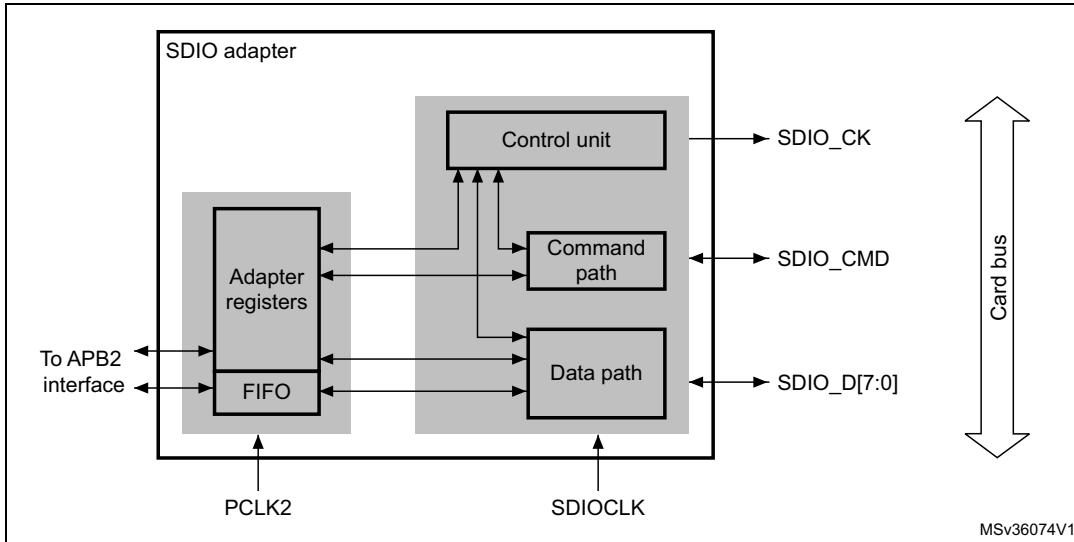
**Table 178. SDIO I/O definitions**

Pin	Direction	Description
SDIO_CK	Output	MultiMediaCard/SD/SDIO card clock. This pin is the clock from host to card.
SDIO_CMD	Bidirectional	MultiMediaCard/SD/SDIO card command. This pin is the bidirectional command/response signal.
SDIO_D[7:0]	Bidirectional	MultiMediaCard/SD/SDIO card data. These pins are the bidirectional databus.

### 29.3.1 SDIO adapter

*Figure 375* shows a simplified block diagram of an SDIO adapter.

**Figure 375. SDIO adapter**



The SDIO 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 SDIO adapter clock domain (SDIOCLK).

#### 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 SDIO Clear register.

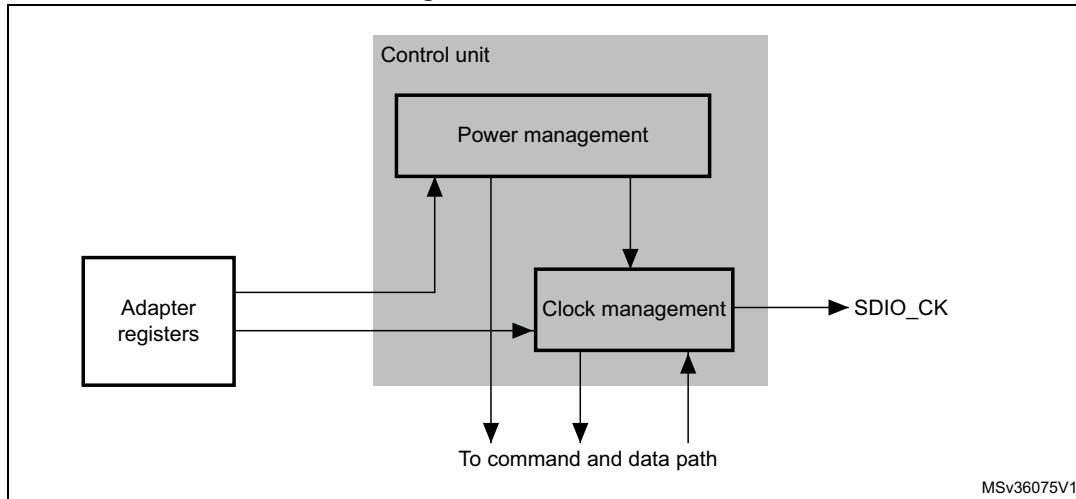
#### Control unit

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 376. Control unit



The control unit is illustrated in [Figure 376](#). 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 SDIO\_CK signal. The SDIO\_CK output can use either the clock divide or the clock bypass mode. The clock output is inactive:

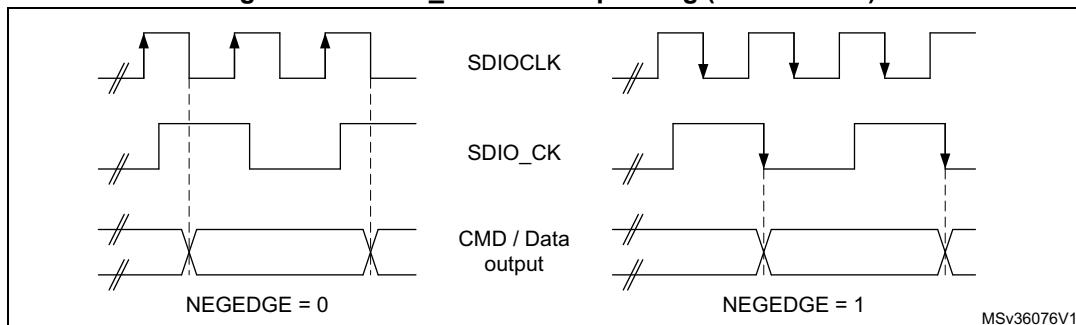
- after reset
- during the power-off or power-up phases
- if the power saving mode is enabled and the card bus is in the Idle state (eight clock periods after both the command and data path subunits enter the Idle phase)

The clock management subunit controls SDIO\_CK dephasing. When not in bypass mode the SDIO command and data output are generated on the SDIOCLK falling edge succeeding the rising edge of SDIO\_CK. (SDIO\_CK rising edge occurs on SDIOCLK rising edge) when SDIO\_CLKCR[13] bit is reset (NEGEDGE = 0). When SDIO\_CLKCR[13] bit is set (NEGEDGE = 1) SDIO command and data changed on the SDIO\_CK falling edge.

When SDIO\_CLKCR[10] is set (BYPASS = 1), SDIO\_CK rising edge occurs on SDIOCLK rising edge. The data and the command change on SDIOCLK falling edge whatever NEGEDGE value.

The data and command responses are latched using SDIO\_CK rising edge.

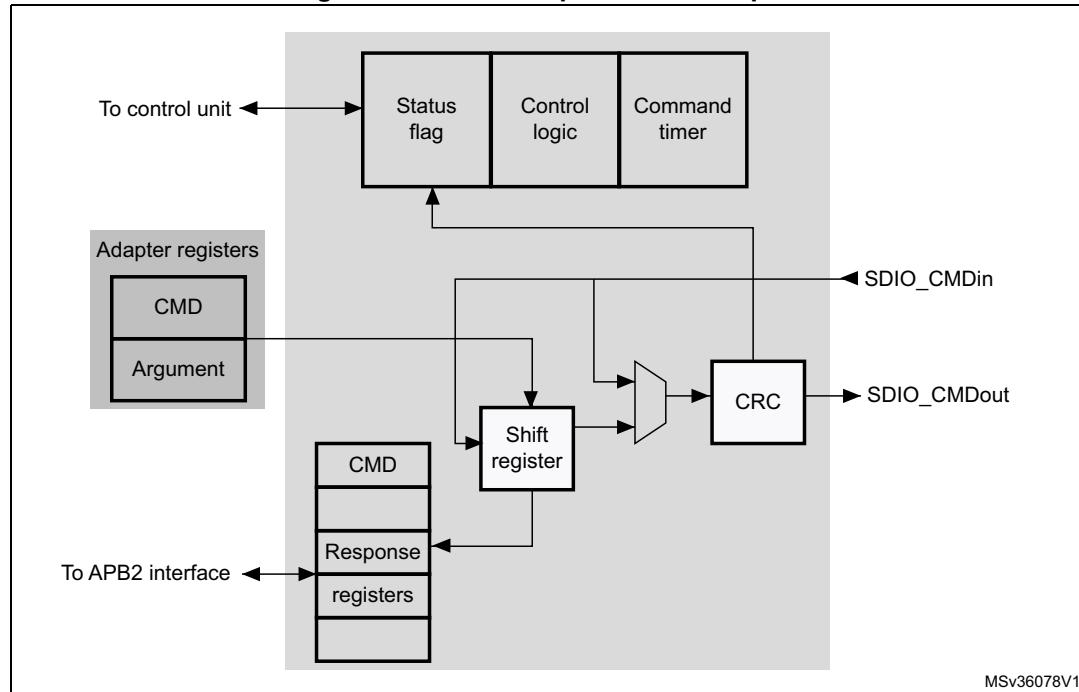
Figure 377. SDIO\_CK clock dephasing (BYPASS = 0)



## Command path

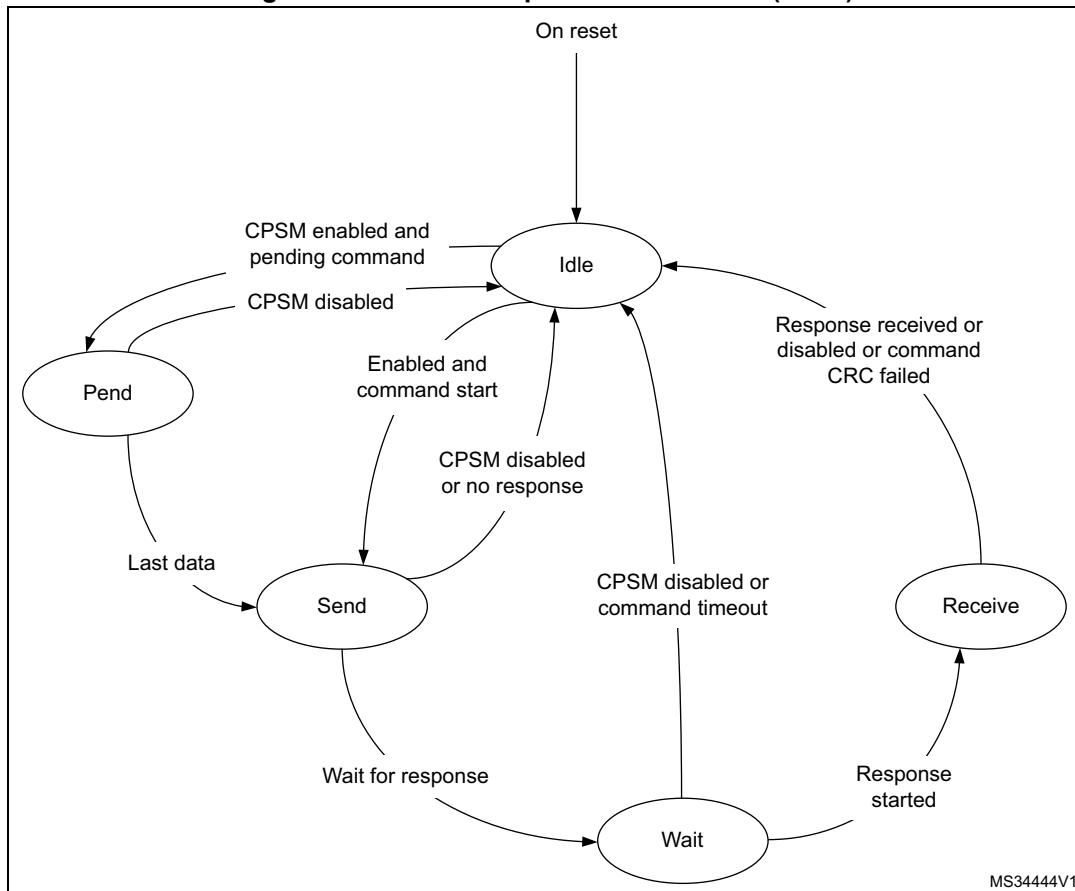
The command path unit sends commands to and receives responses from the cards.

**Figure 378. SDIO adapter command path**



- 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 379 on page 986](#)). When the response is received, the received CRC code and the internally generated code are compared, and the appropriate status flags are set.

Figure 379. Command path state machine (SDIO)



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:**

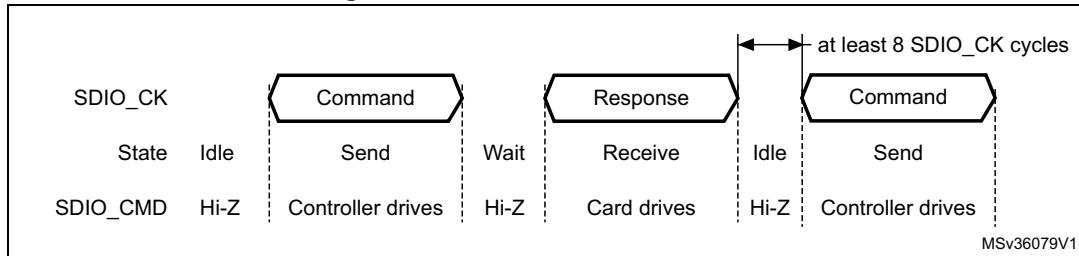
*The command timeout has a fixed value of 64 SDIO\_CK clock periods.*

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 SDIO\_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 380. SDIO command transfer



- 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 179](#).

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 SDIO\_CMD output is in the Hi-Z state, as shown in [Figure 380 on page 987](#). Data on SDIO\_CMD are synchronous with the rising edge of SDIO\_CK. [Table 179](#) shows the command format.

Table 179. 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 SDIO 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 180. 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 181. 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 29.8.4 on page 1023](#)). The command path implements the status flags shown in [Table 182](#):

**Table 182. 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:

$$\text{CRC}[6:0] = \text{Remainder } [(M(x) * x^7) / G(x)]$$

$$G(x) = x^7 + x^3 + 1$$

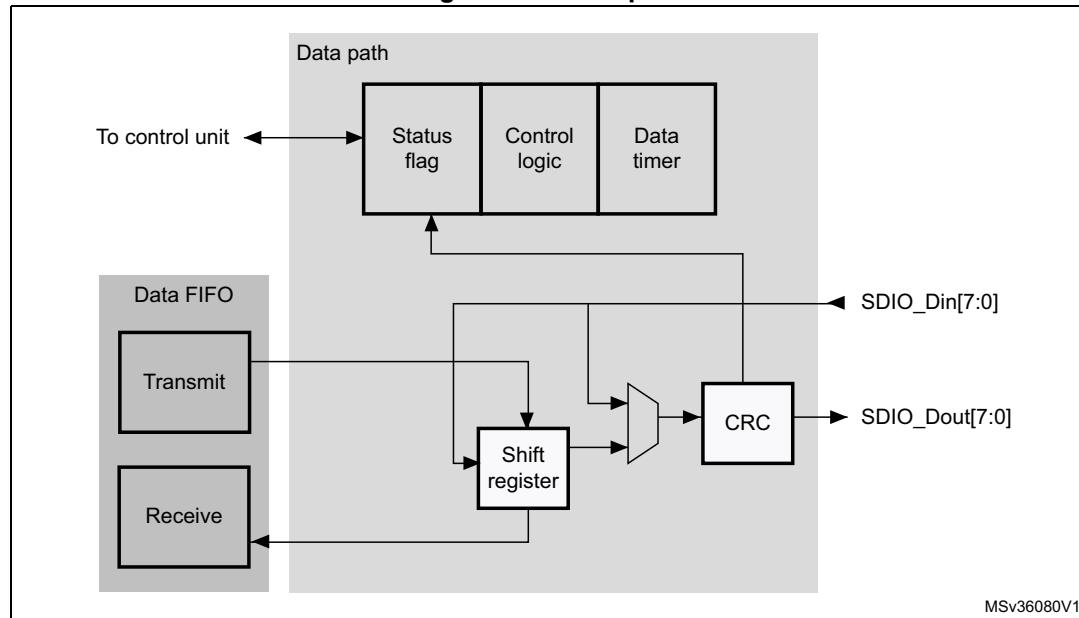
$$M(x) = (\text{start bit}) * x^{39} + \dots + (\text{last bit before CRC}) * x^0, \text{ or}$$

$$M(x) = (\text{start bit}) * x^{119} + \dots + (\text{last bit before CRC}) * x^0$$

## Data path

The data path subunit transfers data to and from cards. [Figure 381](#) shows a block diagram of the data path.

**Figure 381. 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 (SDIO\_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 (SDIO\_D[7:0]). If the wide bus mode is not enabled, only one bit per clock cycle is transferred over SDIO\_D0.

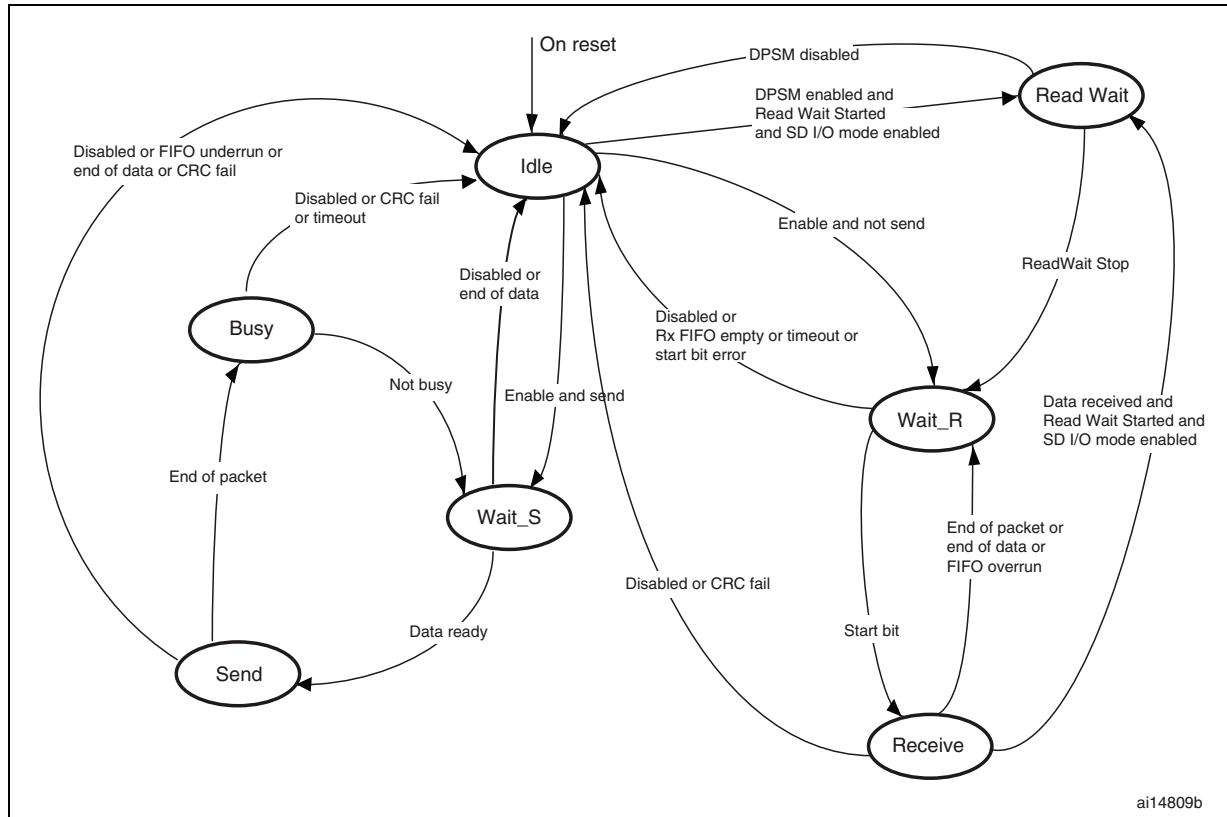
Depending on the transfer direction (send or receive), the data path state machine (DPSM) moves to the Wait\_S or Wait\_R state when it is enabled:

- 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 SDIO\_CK frequency. Data on the card bus signals is synchronous to the rising edge of SDIO\_CK. The DPSM has six states, as shown in [Figure 382: Data path state machine \(DPSM\)](#).

Figure 382. Data path state machine (DPSM)



- **Idle:** the data path is inactive, and the SDIO\_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 SDIO\_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 SDIO\_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 183. 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 184. 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 SDIO_DCOUNT register decrements and reaches 0. DATAEND indicates the end of a transfer on SDIO 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 SDIO clock domain (SDIOCLK) 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 SDIO 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 185. 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 SDIO Clear register. <i>Note:</i> In case of TXUNDERR, and DMA is used to fill SDIO FIFO, user software should disable DMA stream, and then write DMAEN bit in SDIO_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 186](#) lists the receive FIFO status flags. The receive FIFO is accessible via 32 sequential addresses.

**Table 186. 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 SDIO Clear register. <i>Note:</i> In case of RXOVERR, and DMA is used to read SDIO FIFO, user software should disable DMA stream, and then write DMAEN bit in SDIO_DCTRL with '0' (to disable DMA request generation).

### 29.3.2 SDIO APB2 interface

The APB2 interface generates the interrupt and DMA requests, and accesses the SDIO adapter registers and the data FIFO. It consists of a data path, register decoder, and interrupt/DMA logic.

#### SDIO 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.

#### SDIO/DMA interface

SDIO 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 SDIO data length register (SDIO data timer register should be already programmed before the card identification process)
- b) Program DMA channel (refer to [DMA configuration for SDIO controller](#))
- c) Program the SDIO data control register: DTEN with '1' (SDIO 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 SDIO argument register with the address location of the card from where data is to be transferred
- e) Program the SDIO command register: CmdIndex with 17(READ\_BLOCK); WaitResp with '1' (SDIO card host waits for a response); CPSMEN with '1' (SDIO card host enabled to send a command). Other fields are at their reset value.
- f) Wait for SDIO\_STA[6] = CMDREND interrupt, (CMDREND is set if there is no error on command path).
- g) Wait for SDIO\_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 SDIO\_STA[5] = RXOVERR 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 RXOVERR flag is set 2 APB clock cycles after DATAEND flag is set. To guarantee success of read operation RXOVERR must be checked after FIFO is empty.

### Example of write procedure using DMA

Send CMD24 (WRITE\_BLOCK) as follows:

- a) Program the SDIO data length register (SDIO data timer register should be already programmed before the card identification process)
- b) Program DMA channel (refer to [DMA configuration for SDIO controller](#))
- c) Program the SDIO argument register with the address location of the card from where data is to be transferred
- d) Program the SDIO command register: CmdIndex with 24(WRITE\_BLOCK); WaitResp with '1' (SDIO card host waits for a response); CPSMEN with '1' (SDIO card host enabled to send a command). Other fields are at their reset value.
- e) Wait for SDIO\_STA[6] = CMDREND interrupt, then Program the SDIO data control register: DTEN with '1' (SDIO 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 SDIO\_STA[10] = DBCKEND, (DBCKEND is set in case of no errors)

### DMA configuration for SDIO 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 SDIO\_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: SDIO host allows only to use the DMA in peripheral flow controller mode. DMA stream used to serve SDIO must be configured in peripheral flow controller mode*

*SDIO generates only DMA burst requests to DMA controller. DMA must be configured in incremental burst mode on peripheral side.*

## 29.4 Card functional description

### 29.4.1 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.

#### 29.4.2 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).

#### 29.4.3 Operating voltage range validation

All cards can communicate with the SDIO 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 SDIO 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 SDIO card host. The SDIO 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 SDIO 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 SDIO card host is able to select a common voltage range or when the user requires notification that cards are not usable.

#### 29.4.4 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 SDIO\_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 SDIO 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 SDIO 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 SDIO card host and enters the Identification state.
7. The SDIO 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 SDIO 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 SDIO\_CMD line output drives are push-pull drivers instead of open-drain. The registration process is accomplished as follows:

1. The bus is activated.
2. The SDIO 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 SDIO 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 SDIO 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 SDIO card host can reissue this command to change the RCA. The RCA of the card is the last assigned value.
8. The SDIO 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 SDIO 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 SDIO 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 SDIO card host can reissue this command to change the RCA. The RCA of the card is the last assigned value.

#### 29.4.5 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 SDIO\_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 SDIO\_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 SDIO\_D line(s) without interrupting the write operation. When reselecting the card, it will reactivate busy indication by pulling SDIO\_D to low if programming is still in progress and the write buffer is unavailable.

#### 29.4.6 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).

#### 29.4.7 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 SDIO card host to the card, beginning at the specified address and continuing until the SDIO 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 SDIO 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 SDIO 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 SDIO 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.

#### 29.4.8 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 SDIO\_D low. The actual erase time may be quite long, and the host may issue CMD7 to deselect the card.

#### 29.4.9 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).

#### 29.4.10 Protection management

Three write protection methods for the cards are supported in the SDIO card host module:

1. internal card write protection (card responsibility)
2. mechanical write protection switch (SDIO 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 SDIO card host module that the card is write-protected. The SDIO 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 SDIO 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 SDIO 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 SDIO card host module before it sends the card lock/unlock command, and has the structure shown in [Table 200](#).

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 200](#)), 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 SDIO card host module performs all the required steps for setting the password (see [Setting the password on page 1001](#)), 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 200](#)), 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 200](#)) 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.

### 29.4.11 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 187* 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 SDIO 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 187. 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 187. 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	<b>Set when only partial address space was erased due to existing write</b>	C
14	CARD_ECC_DISABLED	S X	'0'= enabled '1'= disabled	The command has been executed without using the internal ECC.	A

Table 187. 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					

### 29.4.12 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 SDIO 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 188* 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 SDIO 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 188. 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 188. 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 189. 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 190. 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 191. 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 192](#). The card can be set to any AU size between RU size and maximum AU size.

**Table 192. 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 193. 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 194. 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 195. Erase offset field**

ERASE_OFFSET	Value definition
0h	0 [sec]
1h	1 [sec]

**Table 195. Erase offset field (continued)**

ERASE_OFFSET	Value definition
2h	2 [sec]
3h	3 [sec]

### 29.4.13 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 SDIO\_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 (SDIO\_D[3:0]). The MultiMediaCard/SD module samples the level of pin 8 (SDIO\_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 SDIO\_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.

#### 29.4.14 Commands and responses

##### Application-specific and general commands

The SDIO 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 SDIO\_D line(s).
- **addressed (point-to-point) data transfer command (ADTC)**: sent to the card that is selected; includes a data transfer on the SDIO\_D line(s).

## Command formats

See [Table 179 on page 987](#) for command formats.

## Commands for the MultiMediaCard/SD module

**Table 196. 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 197. 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 198. 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 199. 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 199. 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 200. 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 201. 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.				

## 29.5 Response formats

All responses are sent via the SDIO command line SDIO\_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:

### 29.5.1 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 202. 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

### 29.5.2 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.

### 29.5.3 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 SDIO\_D0 low. The actual erase time may be quite long, and the host may issue CMD7 to deselect the card.

**Table 203. 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

### 29.5.4 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 204. 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

### 29.5.5 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 205. 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

### 29.5.6 R4b

For SD I/O only: an SDIO card receiving the CMD5 will respond with a unique SDIO response R4. The format is:

**Table 206. R4b response**

Bit position	Width (bits)	Value	Description
47	1	0	Start bit
46	1	0	Transmission bit
[45:40]	6	X	Reserved

**Table 206. 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.

### 29.5.7 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 207. 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

### 29.5.8 R6

Only for SD I/O. The normal response to CMD3 by a memory device. It is shown in [Table 208](#).

**Table 208. 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] [15:0]	16 X	RCA [31:16] of winning card or of the host 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

## 29.6 SDIO I/O card-specific operations

The following features are SD I/O-specific operations:

- SDIO read wait operation by SDIO\_D2 signalling
- SDIO read wait operation by stopping the clock
- SDIO suspend/resume operation (write and read suspend)
- SDIO interrupts

The SDIO supports these operations only if the SDIO\_DCTRL[11] bit is set, except for read suspend that does not need specific hardware implementation.

### 29.6.1 SDIO I/O read wait operation by SDIO\_D2 signalling

It is possible to start the readwait interval before the first block is received: when the data path is enabled (SDIO\_DCTRL[0] bit set), the SDIO-specific operation is enabled (SDIO\_DCTRL[11] bit set), read wait starts (SDIO\_DCTRL[10] =0 and SDIO\_DCTRL[8] =1) and data direction is from card to SDIO (SDIO\_DCTRL[1] = 1), the DPSM directly moves from Idle to Readwait. In Readwait the DPSM drives SDIO\_D2 to 0 after 2 SDIO\_CK clock cycles. In this state, when you set the RWSTOP bit (SDIO\_DCTRL[9]), the DPSM remains in Wait for two more SDIO\_CK clock cycles to drive SDIO\_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 SDIO can detect SDIO interrupts on SDIO\_D1.

### 29.6.2 SDIO read wait operation by stopping SDIO\_CK

If the SDIO card does not support the previous read wait method, the SDIO can perform a read wait by stopping SDIO\_CK (SDIO\_DCTRL is set just like in the method presented in [Section 29.6.1](#), but SDIO\_DCTRL[10] =1): DPSM stops the clock two SDIO\_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 SDIO\_CK is stopped, any command can be issued to the card. During a read/wait interval, the SDIO can detect SDIO interrupts on SDIO\_D1.

### 29.6.3 SDIO suspend/resume operation

While sending data to the card, the SDIO can suspend the write operation. the SDIO\_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 (SDIO\_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.

### 29.6.4 SDIO interrupts

SDIO interrupts are detected on the SDIO\_D1 line once the SDIO\_DCTRL[11] bit is set.

When SDIO interrupt is detected, SDIO\_STA[22] (SDIOIT) bit is set. This static bit can be cleared with clear bit SDIO\_ICR[22] (SDIOITC). An interrupt can be generated when SDIOIT status bit is set. Separated interrupt enable SDIO\_MASK[22] bit (SDIOITE) is available to enable and disable interrupt request.

When SD card interrupt occurs (SDIO\_STA[22] bit set), host software follows below steps to handle it.

1. Disable SDIOIT interrupt signaling by clearing SDIOITE bit (SDIO\_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 (SDIO\_ICR[22] = '1'),
4. Enable SDIOIT interrupt signaling by writing '1' to SDIOITE bit (SDIO\_MASK[22] = '1').

*Steps 2 to 4 can be executed out of the SDIO interrupt service routine.*

## 29.7 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 SDIO\_CK and freeze SDIO state machines. The data transfer is stalled while the FIFO is unable to transmit or receive data. Only state machines clocked by

SDIOCLK 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 SDIO\_CLKCR[14] register bit must be set to 1. After reset Flow Control is disabled.

## 29.8 SDIO registers

The device communicates to the system via 32-bit-wide control registers accessible via APB2.

### 29.8.1 SDIO power control register (SDIO\_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.	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 SDIOCLK clock periods plus two PCLK2 clock periods.

### 29.8.2 SDIO clock control register (SDIO\_CLKCR)

Address offset: 0x04

Reset value: 0x0000 0000

The SDIO\_CLKCR register controls the SDIO\_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	NEGE DGE	WID BUS		BYPAS S	PWRS AV	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 SDIO Status register definition in [Section 29.8.11](#).

Bit 13 **NEGEDGE:** SDIO\_CK dephasing selection bit

0b: Command and Data changed on the SDIOCLK falling edge succeeding the rising edge of SDIO\_CK. (SDIO\_CK rising edge occurs on SDIOCLK rising edge).

1b: Command and Data changed on the SDIO\_CK falling edge.

When BYPASS is active, the data and the command change on SDIOCLK falling edge whatever NEGEDGE value.

Bits 12:11 **WIDBUS:** Wide bus mode enable bit

00: Default bus mode: SDIO\_D0 used

01: 4-wide bus mode: SDIO\_D[3:0] used

10: 8-wide bus mode: SDIO\_D[7:0] used

Bit 10 **BYPASS:** Clock divider bypass enable bit

0: Disable bypass: SDIOCLK is divided according to the CLKDIV value before driving the SDIO\_CK output signal.

1: Enable bypass: SDIOCLK directly drives the SDIO\_CK output signal.

Bit 9 **PWRSAV:** Power saving configuration bit

For power saving, the SDIO\_CK clock output can be disabled when the bus is idle by setting PWRSAV:

0: SDIO\_CK clock is always enabled

1: SDIO\_CK is only enabled when the bus is active

Bit 8 **CLKEN:** Clock enable bit

0: SDIO\_CK is disabled

1: SDIO\_CK is enabled

Bits 7:0 **CLKDIV:** Clock divide factor

This field defines the divide factor between the input clock (SDIOCLK) and the output clock (SDIO\_CK): SDIO\_CK frequency = SDIOCLK / [CLKDIV + 2].

- Note:**
- 1 While the SD/SDIO card or MultiMediaCard is in identification mode, the SDIO\_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 SDIOCLK clock periods plus two PCLK2 clock periods. SDIO\_CK can also be stopped during the read wait interval for SD I/O cards: in this case the SDIO\_CLKCR register does not control SDIO\_CK.

### 29.8.3 SDIO argument register (SDIO\_ARG)

Address offset: 0x08

Reset value: 0x0000 0000

The SDIO\_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.

### 29.8.4 SDIO command register (SDIO\_CMD)

Address offset: 0x0C

Reset value: 0x0000 0000

The SDIO\_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 SDIO\_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 SDIOCLK 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.*

### 29.8.5 SDIO command response register (SDIO\_RESPCMD)

Address offset: 0x10

Reset value: 0x0000 0000

The SDIO\_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.

### 29.8.6 SDIO response 1..4 register (SDIO\_RESPx)

Address offset: (0x10 + (4 × x)); x = 1..4

Reset value: 0x0000 0000

The SDIO\_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 209](#).

The Card Status size is 32 or 127 bits, depending on the response type.

**Table 209. Response type and SDIO\_RESPx registers**

Register	Short response	Long response
SDIO_RESP1	Card Status[31:0]	Card Status [127:96]
SDIO_RESP2	Unused	Card Status [95:64]
SDIO_RESP3	Unused	Card Status [63:32]
SDIO_RESP4	Unused	Card Status [31:1]0b

The most significant bit of the card status is received first. The SDIO\_RESP4 register LSB is always 0b.

### 29.8.7 SDIO data timer register (SDIO\_DTIMER)

Address offset: 0x24

Reset value: 0x0000 0000

The SDIO\_DTIMER register contains the data timeout period, in card bus clock periods.

A counter loads the value from the SDIO\_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
DATATIME[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
DATATIME[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **DATATIME**: 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.

### 29.8.8 SDIO data length register (SDIO\_DLEN)

Address offset: 0x28

Reset value: 0x0000 0000

The SDIO\_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 SDIO\_DCTRL). A data transfer must be written to the data timer register and the data length register before being written to the data control register.

For an SDIO multibyte transfer the value in the data length register must be between 1 and 512.

### 29.8.9 SDIO data control register (SDIO\_DCTRL)

Address offset: 0x2C

Reset value: 0x0000 0000

The SDIO\_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	DTDIR	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 SDIO\_D2
- 1: Read Wait control using SDIO\_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 SDIO\_DCTRL must be updated to enable a new data transfer

**Note:** After a data write, data cannot be written to this register for three SDIOCLK (48 MHz) 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.

### 29.8.10 SDIO data counter register (SDIO\_DCOUNT)

Address offset: 0x30

Reset value: 0x0000 0000

The SDIO\_DCOUNT register loads the value from the data length register (see SDIO\_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]														
							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						
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.

### 29.8.11 SDIO status register (SDIO\_STA)

Address offset: 0x34

Reset value: 0x0000 0000

The SDIO\_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 SDIO Interrupt Clear register (see SDIO\_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
									r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RX FIFO HF	TX FIFO HE	RXACT	TXACT	CMD ACT	DBCK END	Res.	DATA END	CMDS ENT	CMDR END	RX OVERR	TXUND ERR	DTIME OUT	CTIME OUT	DCRC FAIL	CCRC FAIL
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 SDIO FIFO (DMAEN bit is set in SDIO\_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 SDIO FIFO (DMAEN bit is set in SDIO\_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 SDIO\_CK clock periods.
- Bit 1 **DCRCFAIL:** Data block sent/received (CRC check failed)
- Bit 0 **CCRCFAIL:** Command response received (CRC check failed)

### 29.8.12 SDIO interrupt clear register (SDIO\_ICR)

Address offset: 0x38

Reset value: 0x0000 0000

The SDIO\_ICR register is a write-only register. Writing a bit with 1b clears the corresponding bit in the SDIO\_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

**29.8.13 SDIO mask register (SDIO\_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

### 29.8.14 SDIO FIFO counter register (SDIO\_FIFOCNT)

Address offset: 0x48

Reset value: 0x0000 0000

The SDIO\_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 SDIO\_DLEN) when the data transfer enable bit, DTEN, is set in the data control register (SDIO\_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.

### 29.8.15 SDIO data FIFO register (SDIO\_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:  
SDIO base + 0x080 to SDIO base + 0xFC.

### 29.8.16 SDIO register map

The following table summarizes the SDIO registers.

**Table 210. SDIO 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	<b>SDIO_POWER</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	
		Reset value																													0	0	
0x04	<b>SDIO_CLKCR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	
		Reset value																														0	
0x08	<b>SDIO_ARG</b>																																
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x0C	<b>SDIO_CMD</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	
		Reset value																															0
0x10	<b>SDIO_RESPCMD</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	
		Reset value																														0	
0x14	<b>SDIO_RESP1</b>																																
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x18	<b>SDIO_RESP2</b>																																
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x1C	<b>SDIO_RESP3</b>																																
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x20	<b>SDIO_RESP4</b>																																
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x24	<b>SDIO_DTIME</b>																																
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x28	<b>SDIO_DLEN</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	
		Reset value																															
0x2C	<b>SDIO_DCTRL</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	
		Reset value																															
0x30	<b>SDIO_DCOUNT</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	0	
		Reset value																															

Table 210. SDIO 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											
0x34	<b>SDIO_STA</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SDIOITC	0	RXDAVL	0	RXDAVL	0	RXFIFOE	0	RXFIFOE	0	RXFIFOE	0	RXFIFOH	0	CMDACT	0	DBCKENDC	0	DATAENDC	0	TXUNDERRC	0	TXUNDERRC	0										
		Reset value																																										
0x38	<b>SDIO_ICR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SDIOIE	0	RXDAVIE	0	TXDAVIE	0	RXFIFOIE	0	TXFIFOIE	0	RXFIFOIE	0	RXFIFOHF	0	CMDRENDC	0	DBCKENDC	0	DATAENDC	0	TXUNDERTC	0	TXUNDERTC	0										
		Reset value																																										
0x3C	<b>SDIO_MASK</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SDIOITIE	0	RXDAVIE	0	TXDAVIE	0	RXFIFOIE	0	TXFIFOIE	0	RXFIFOIE	0	RXFIFOHF	0	CMDACT	0	DBCKENDC	0	DATAENDC	0	TXUNDERTC	0	TXUNDERTC	0										
		Reset value																																										
0x48	<b>SDIO_FIFOCNT</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	FIFOCOUNT	0	0	0	0	0	0	0	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																																										
0x80	<b>SDIO_FIFO</b>										FIFOData	0	0	0	0	0	0	0	0	0	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																																		

Refer to [Section 2.2.2: Memory map and register boundary addresses](#) for the register boundary addresses.

## 30 Controller area network (bxCAN)

### 30.1 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.

### 30.2 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:
  - 28 filter banks shared between CAN1 and CAN2
- 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

#### Dual CAN peripheral configuration

- CAN1: Master bxCAN for managing the communication between a Slave bxCAN and the 512-byte SRAM memory
- CAN2: Slave bxCAN, with no direct access to the SRAM memory.
- The two bxCAN cells share the 512-byte SRAM memory (see *Figure 384: Dual CAN block diagram*)

### 30.3 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.

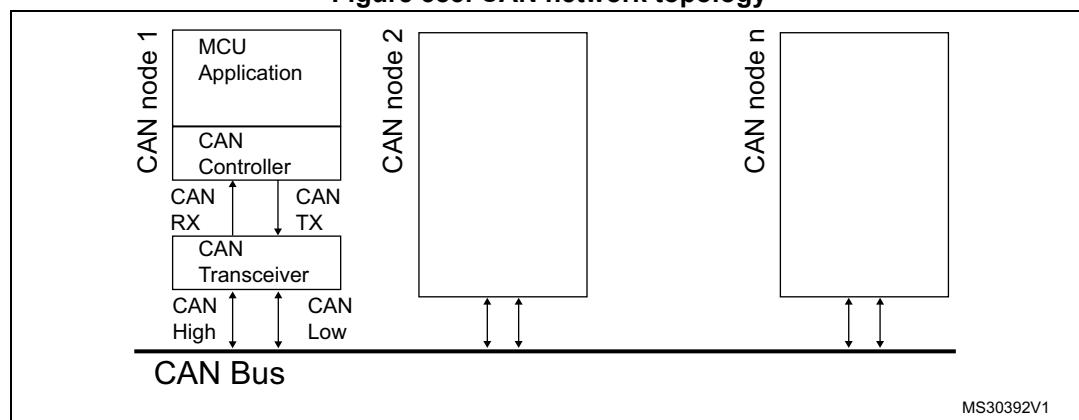
- An enhanced filtering mechanism is required to handle each type of message.

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 383. CAN network topology**



#### 30.3.1 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.

#### 30.3.2 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

#### 30.3.3 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.

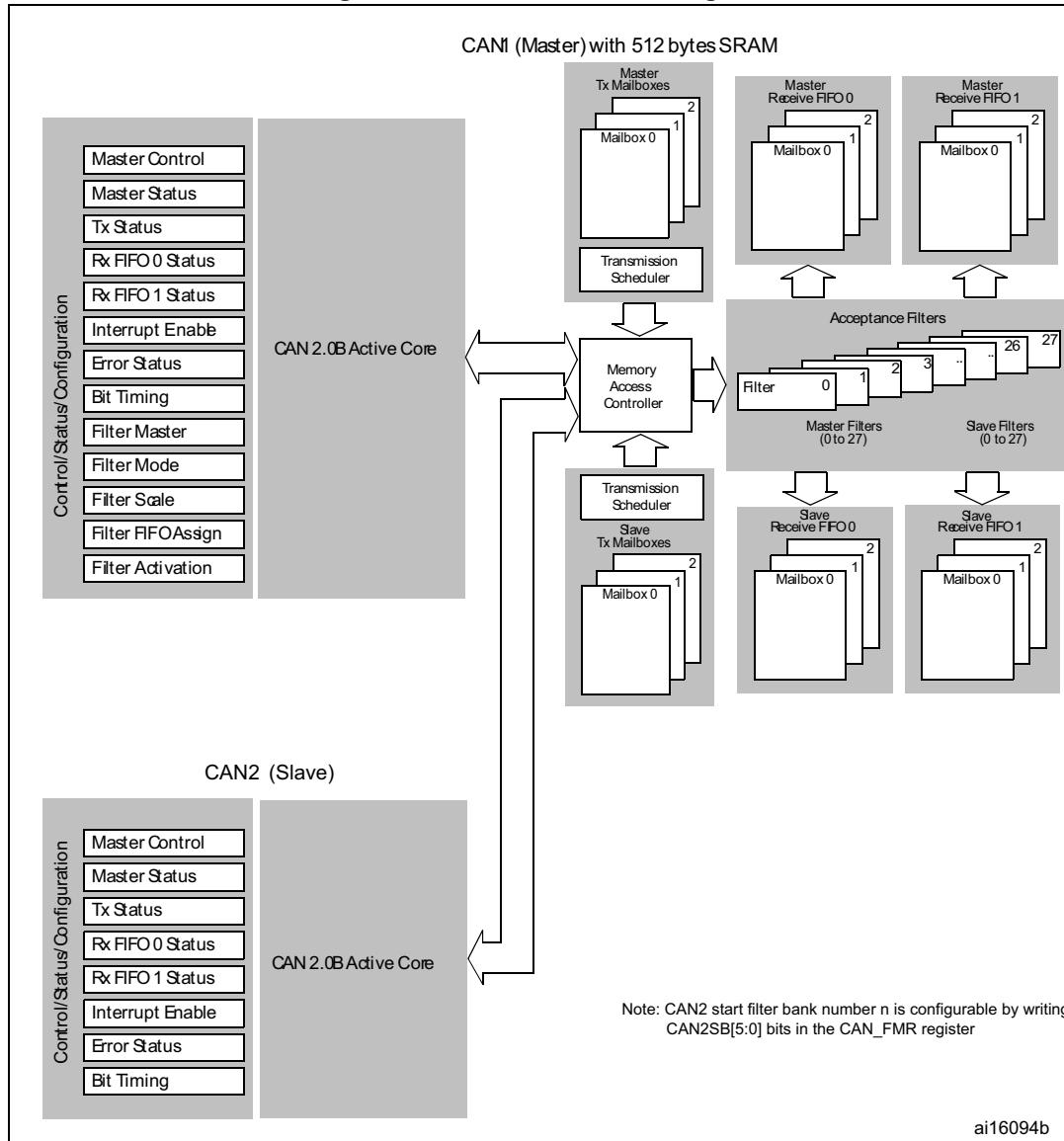
### 30.3.4 Acceptance filters

The bxCAN provides up to 28 scalable/configurable identifier filter banks, 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 384. Dual CAN block diagram**



### 30.4 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.

#### 30.4.1 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).*

#### 30.4.2 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.

#### 30.4.3 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.

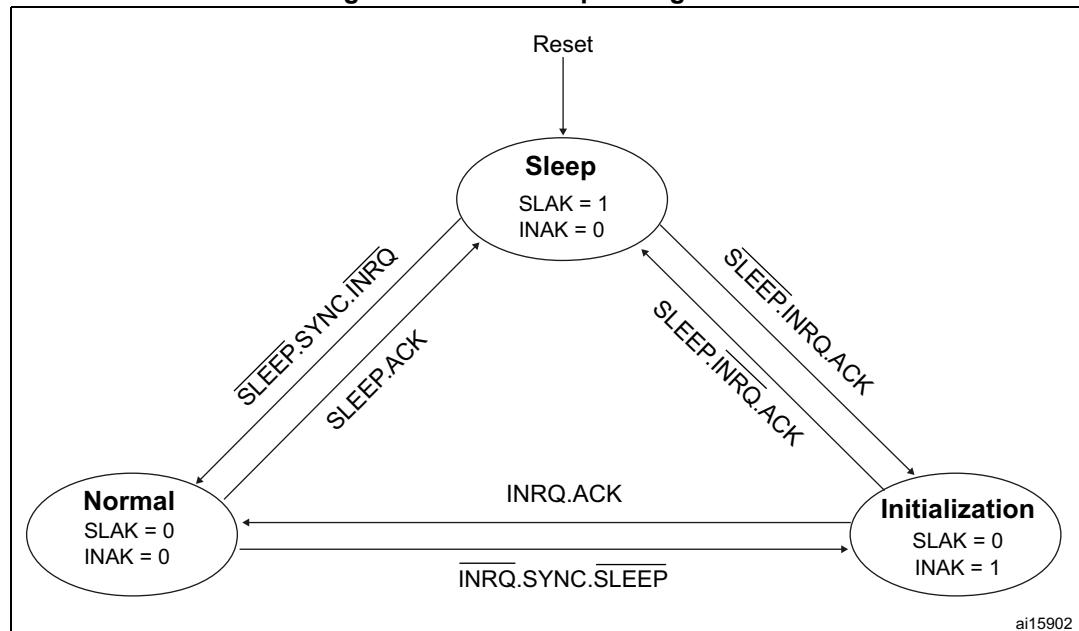
bxCAN can be woken up (exit Sleep mode) either by software clearing the SLEEP bit or on detection of CAN bus activity.

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 385: bxCAN operating modes](#). The Sleep mode is exited once the SLAK bit has been cleared by hardware.

**Figure 385. 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

## 30.5 Test mode

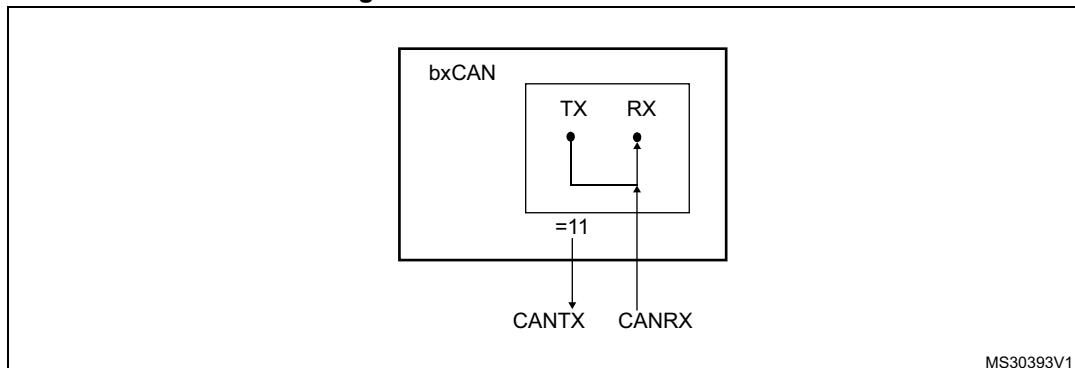
Test mode can be selected by the SILM and LBKM bits in the CAN\_BTR register. These bits must be configured while bxCAN is in Initialization mode. Once test mode has been selected, the INRQ bit in the CAN\_MCR register must be reset to enter Normal mode.

### 30.5.1 Silent mode

The bxCAN can be put in Silent mode by setting the SILEM 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).

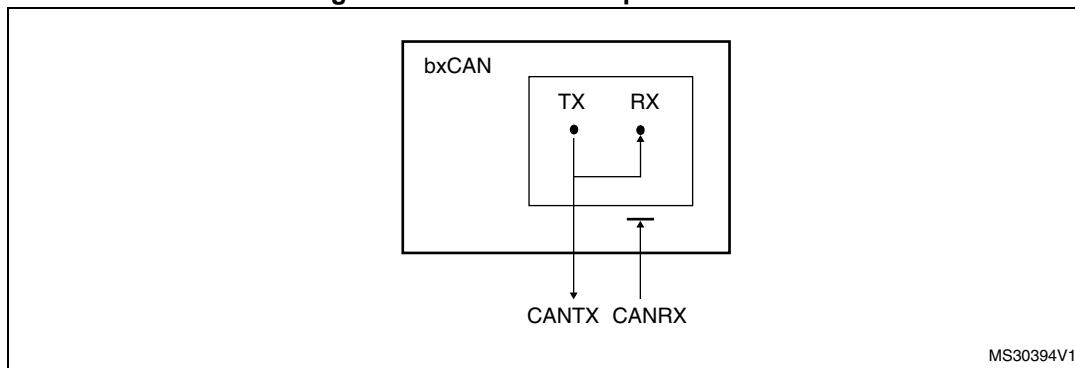
**Figure 386. bxCAN in silent mode**



### 30.5.2 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 387. bxCAN in loop back mode**



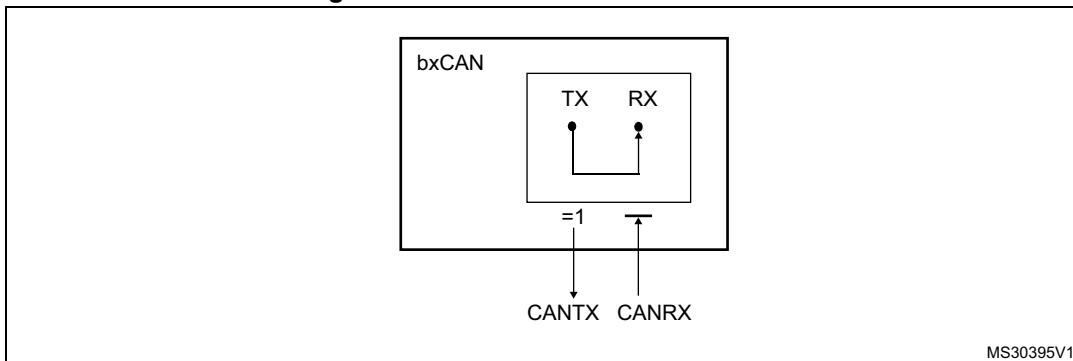
This mode is provided for self-test functions. To be independent of external events, the CAN Core ignores acknowledge errors (no dominant bit sampled in the acknowledge slot of a data / remote frame) in Loop Back Mode. In this mode, the bxCAN performs an internal feedback from its Tx output to its Rx input. The actual value of the CANRX input pin is disregarded by the bxCAN. The transmitted messages can be monitored on the CANTX pin.

### 30.5.3 Loop back combined with silent mode

It is also possible to combine Loop Back mode and Silent mode by setting the LBKM and SILEM bits in the CAN\_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 388. bxCAN in combined mode**



## 30.6 Behavior in debug mode

When the microcontroller enters the debug mode (Cortex®-M4 with FPU core halted), the bxCAN continues to work normally or stops, depending on:

- the DBG\_CAN1\_STOP bit for CAN1 or the DBG\_CAN2\_STOP bit for CAN2 in the DBG module.
- the DBF bit in CAN\_MCR. For more details, refer to [Section 30.9.2: CAN control and status registers](#).

## 30.7 bxCAN functional description

### 30.7.1 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\_TlRxR 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.

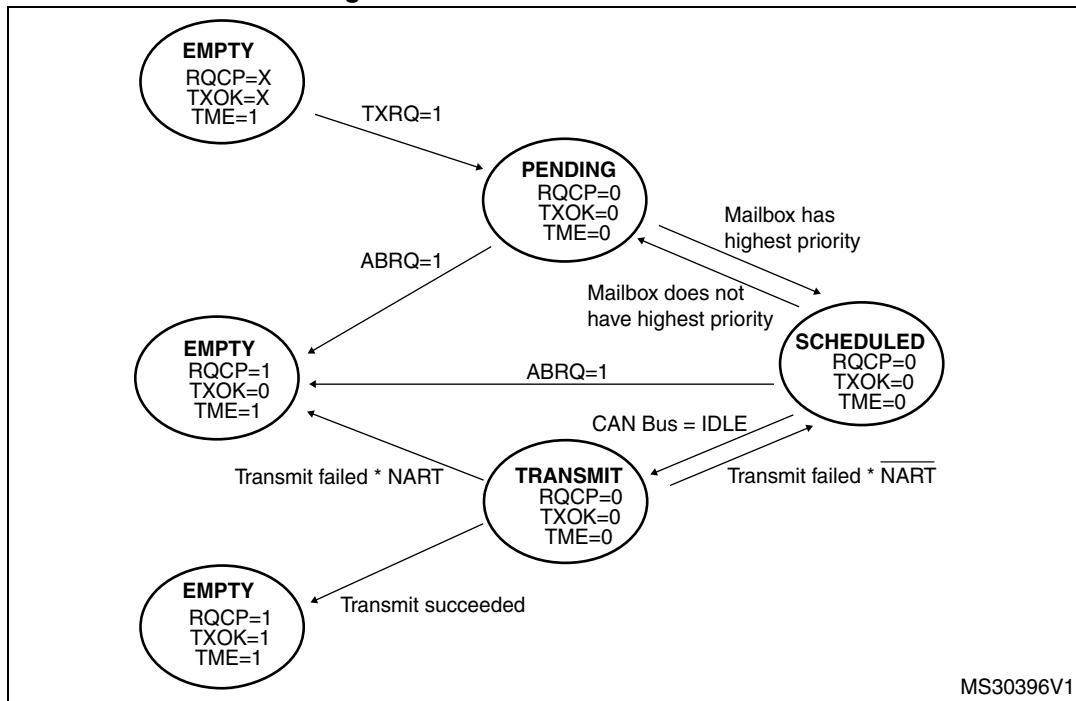
### Non automatic retransmission mode

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 389. Transmit mailbox states



### 30.7.2 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 30.7.7: Bit timing](#)). The internal counter is captured on the sample point of the Start Of Frame bit in both reception and transmission.

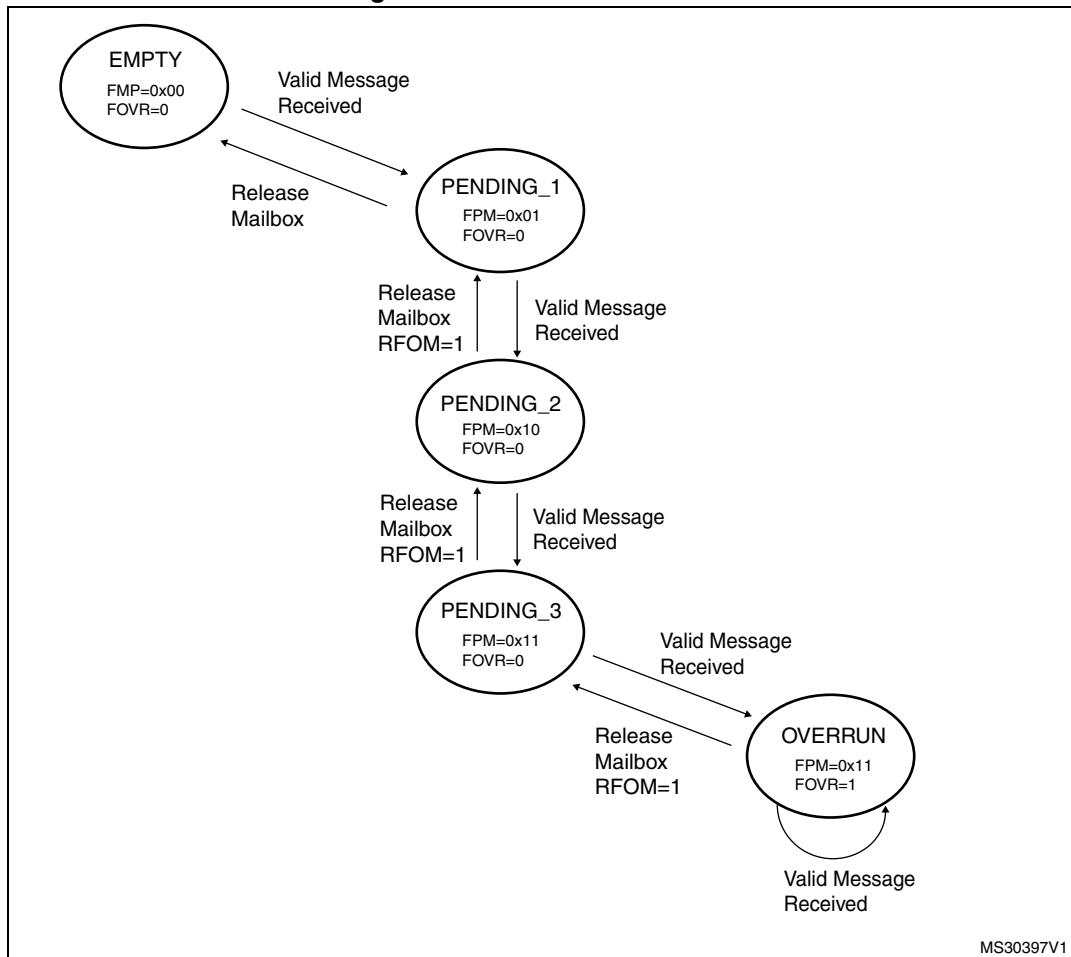
### 30.7.3 Reception handling

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 30.7.4: Identifier filtering](#).

Figure 390. 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 30.7.5: 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.

## 30.7.4 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 28 configurable and scalable filter banks (27-0) 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 391](#).

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 391](#). 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.

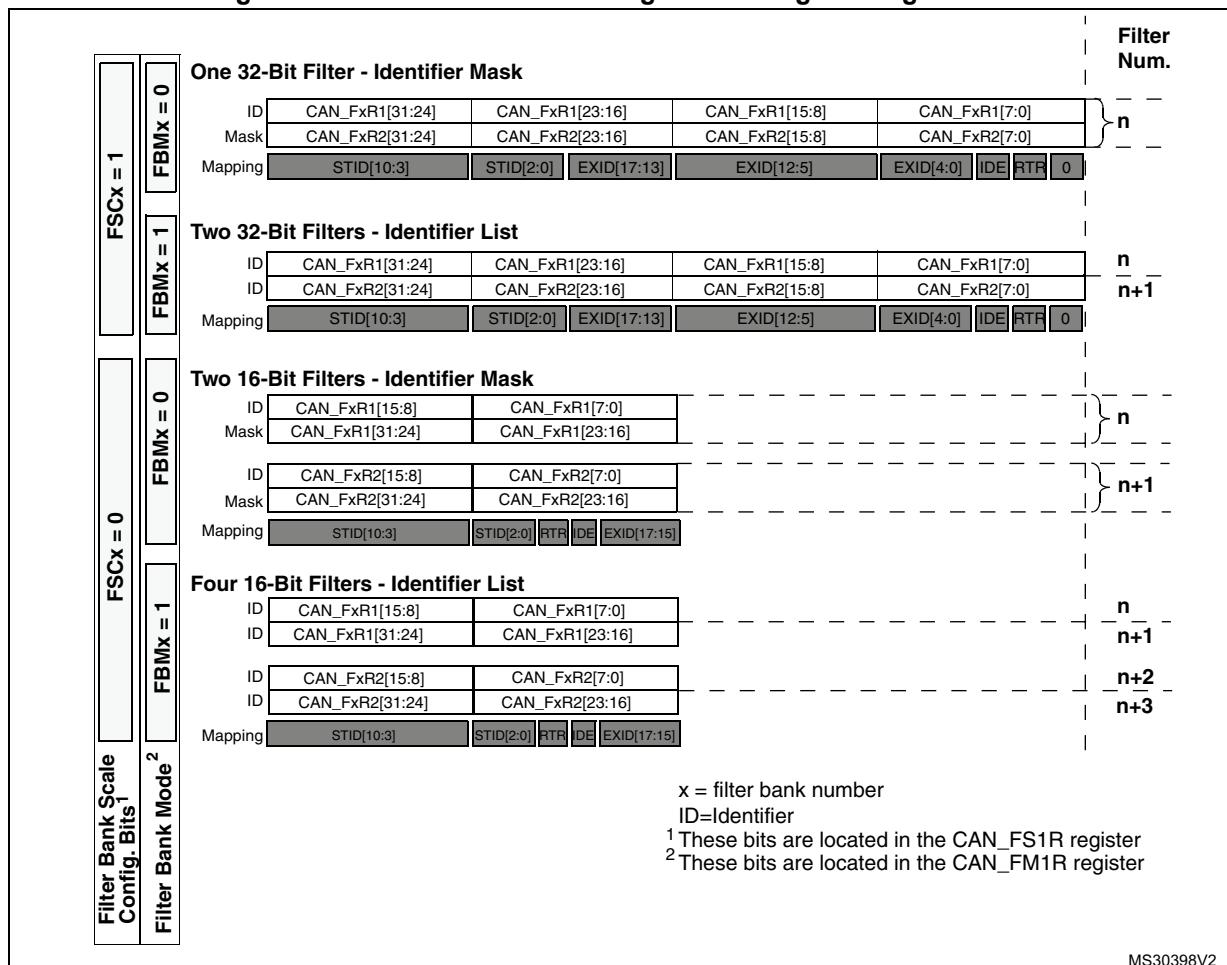
To select single identifiers, configure the Mask/Identifier registers in identifier list mode.

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 391](#).

Figure 391. Filter bank scale configuration - register organization



MS30398V2

## 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 392](#) for an example.

Figure 392. 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 List (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

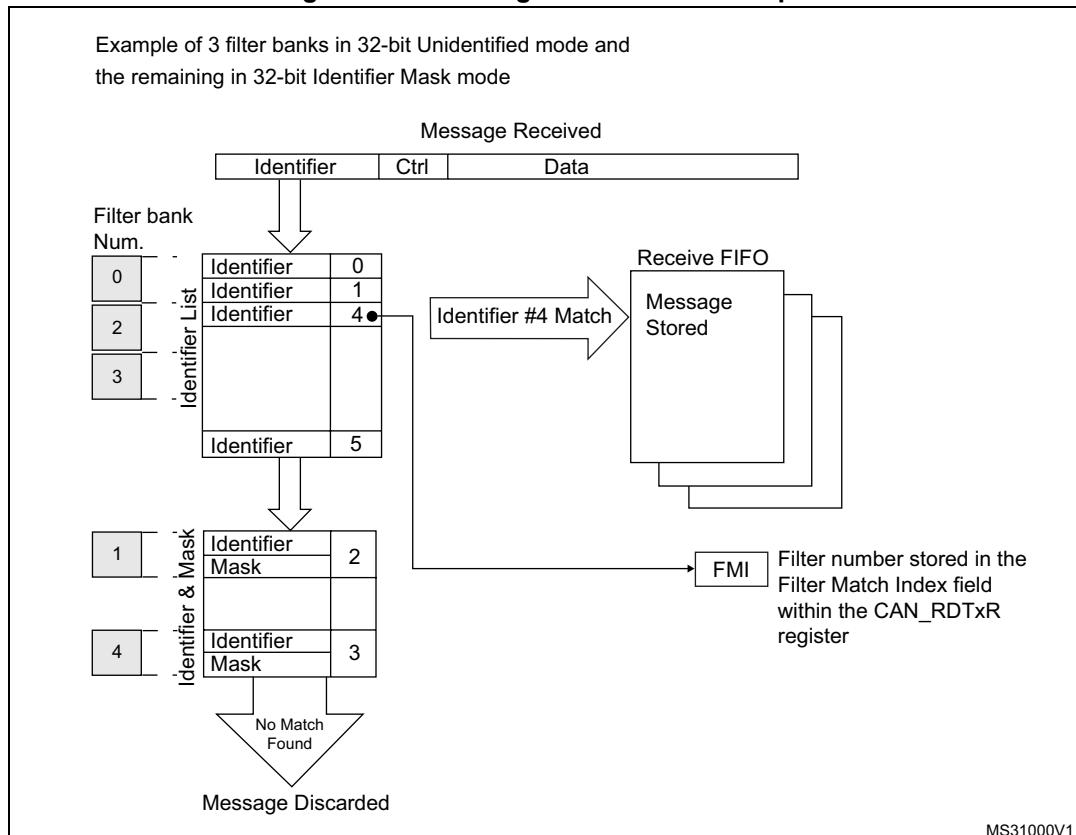
ID=Identifier

MS30399V1

### 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 393. 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.

### 30.7.5 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 211. Transmit mailbox mapping**

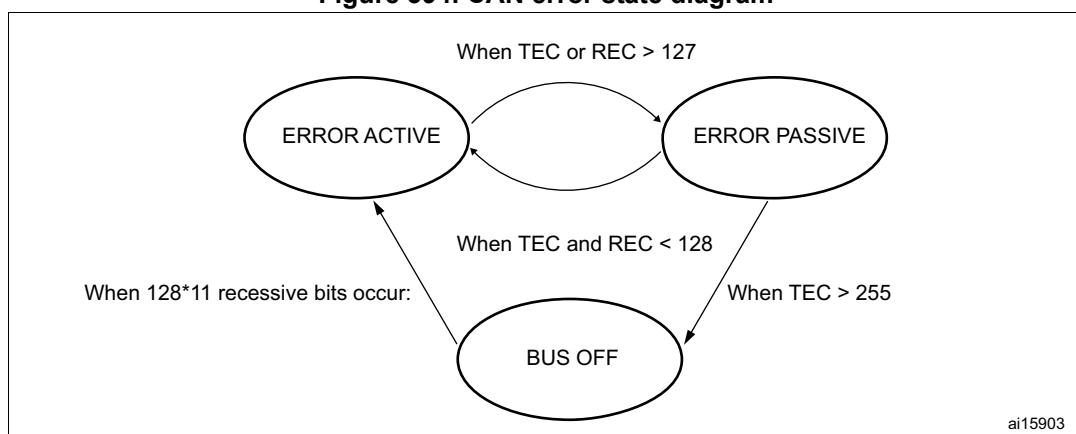
Offset to transmit mailbox base address	Register name
0	CAN_TIxR
4	CAN_TDTxR
8	CAN_TDLxR
12	CAN_TDhxR

**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 212. 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 394. CAN error state diagram**

### 30.7.6 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.*

### 30.7.7 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 395. Bit timing**

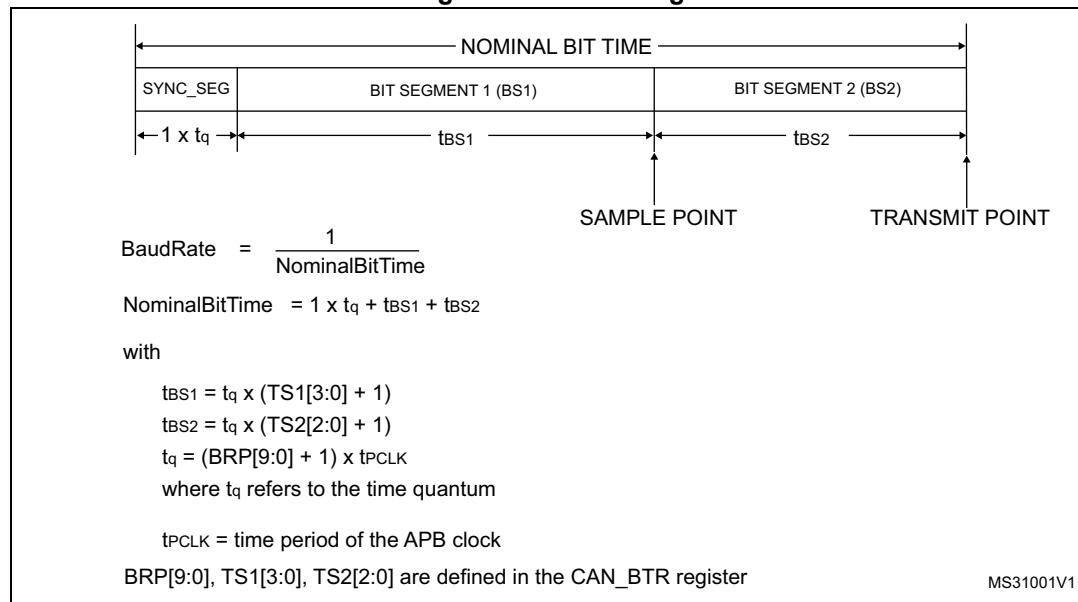
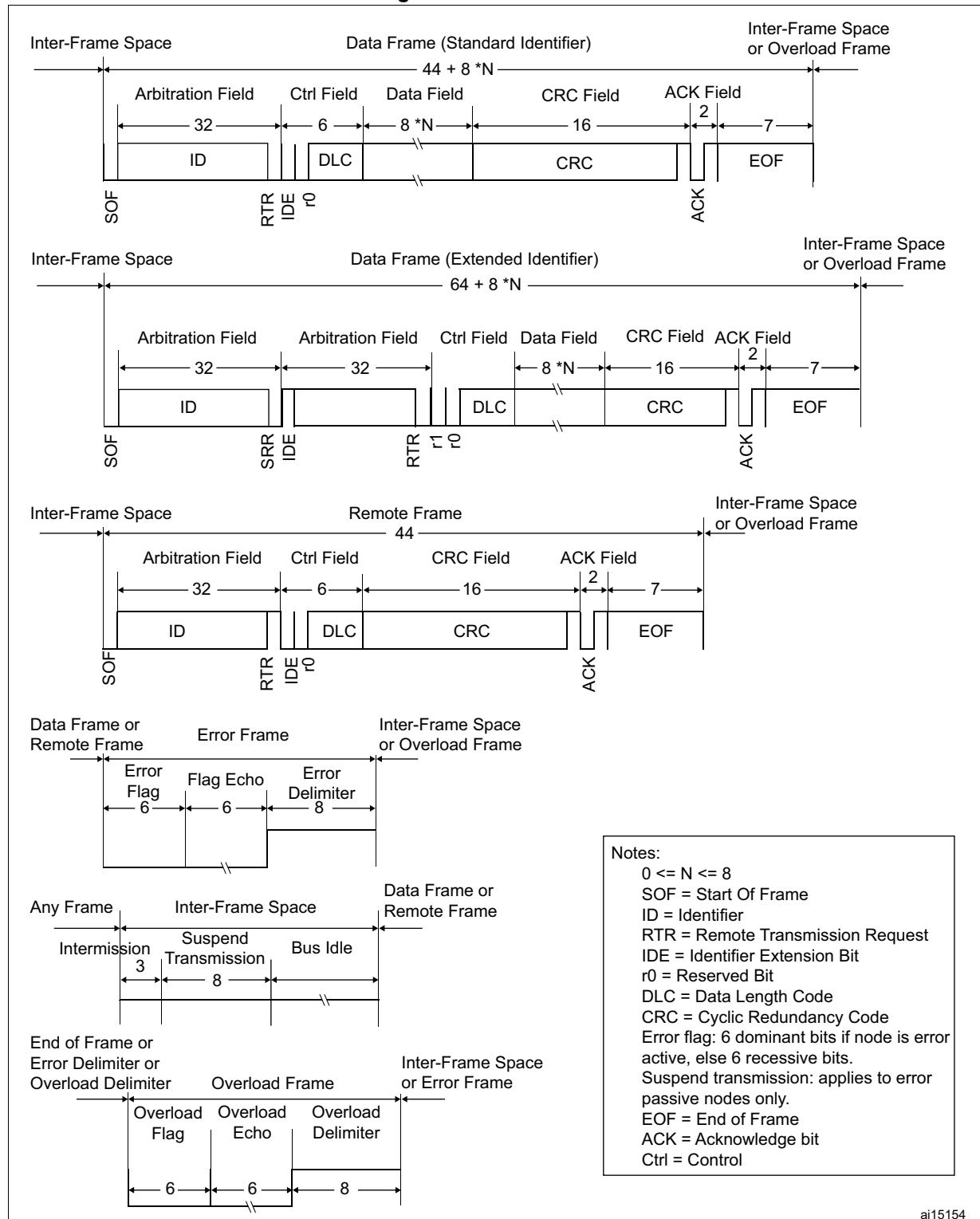


Figure 396. CAN frames

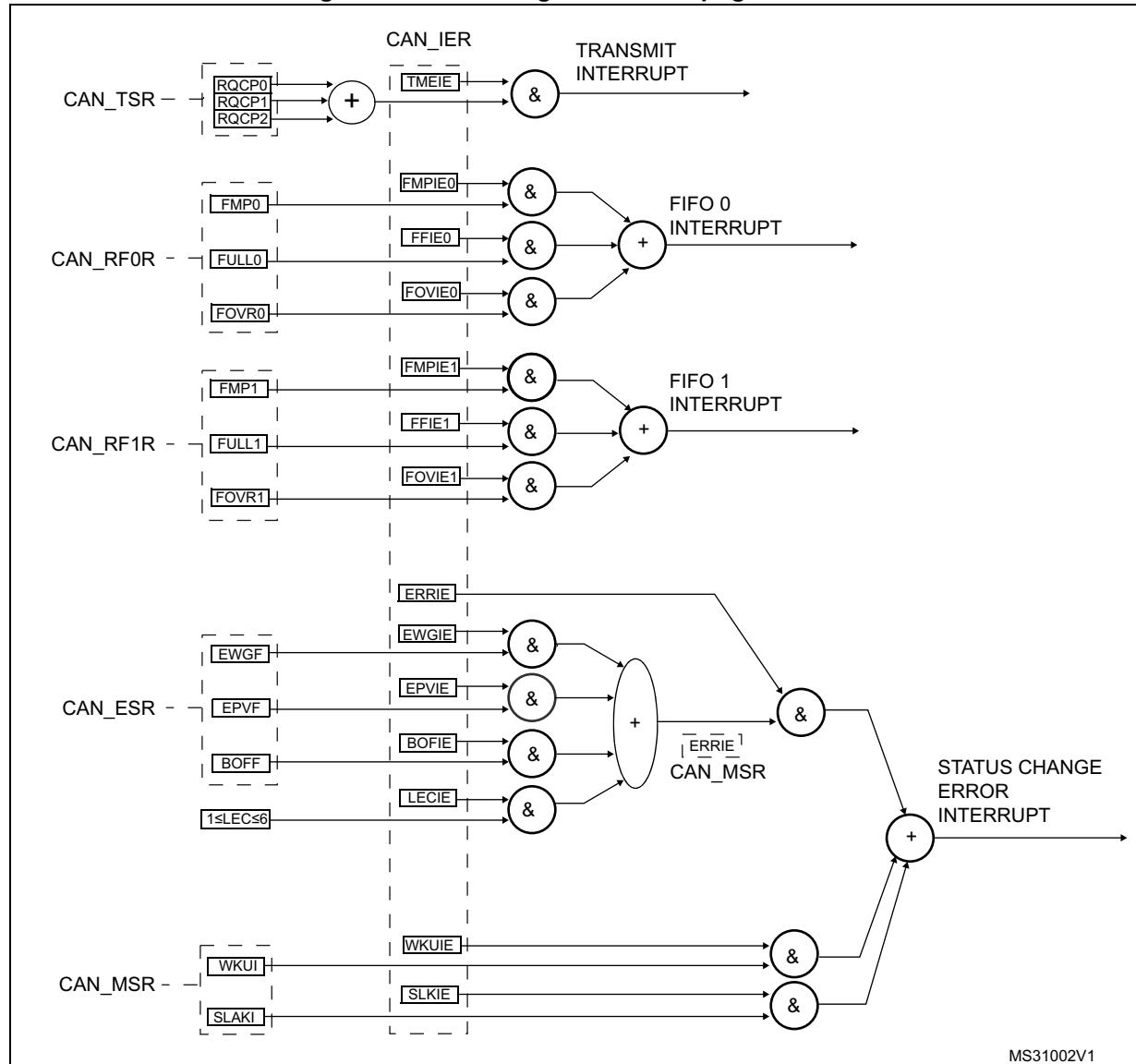


ai15154

## 30.8 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 397. 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, FOVRO 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.

## 30.9 CAN registers

The peripheral registers have to be accessed by words (32 bits).

### 30.9.1 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 389: 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.

### 30.9.2 CAN control and status registers

Refer to [Section 1.1](#) 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 30.7.2: 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 30.7.6: 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 389](#).
- 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 389](#)

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 389](#)

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 30.8: 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 30.7.6 on page 1053](#).

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 30.7.7: Bit timing on page 1053](#).

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.  
 $t_q = (BRP[9:0]+1) \times t_{PCLK}$

### 30.9.3 CAN mailbox registers

This chapter describes the registers of the transmit and receive mailboxes. Refer to [Section 30.7.5: Message storage on page 1051](#) for detailed register mapping.

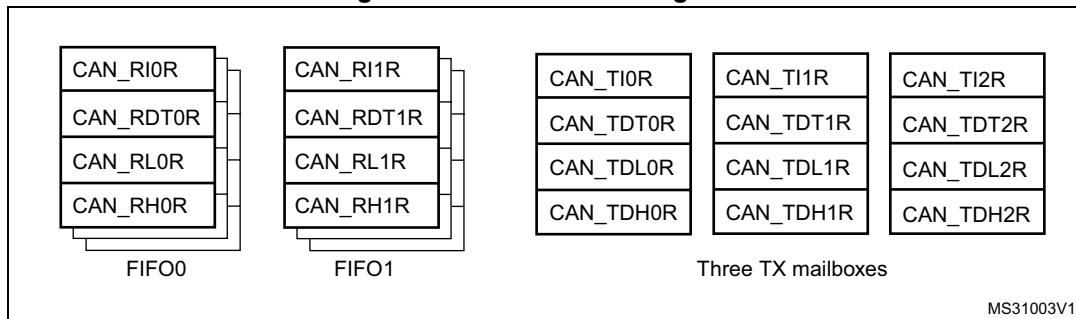
Transmit and receive mailboxes have the same registers except:

- 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 398. 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: 0xFFFF XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA3[7:0]								DATA2[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA1[7:0]								DATA0[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **DATA3[7:0]**: Data byte 3

Data byte 3 of the message.

Bits 23:16 **DATA2[7:0]**: Data byte 2

Data byte 2 of the message.

Bits 15:8 **DATA1[7:0]**: Data byte 1

Data byte 1 of the message.

Bits 7:0 **DATA0[7:0]**: Data byte 0

Data byte 0 of the message.

A message can contain from 0 to 8 data bytes and starts with byte 0.

**CAN mailbox data high register (CAN\_TDhxR) (x = 0..2)**

All bits of this register are write protected when the mailbox is not in empty state.

Address offsets: 0x18C, 0x19C, 0x1AC

Reset value: 0xFFFF XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA7[7:0]								DATA6[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA5[7:0]								DATA4[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 **DATA7[7:0]**: Data byte 7

Data byte 7 of the message.

*Note: If TGT of this message and TTCM are active, DATA7 and DATA6 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
FMI[7:0]															
r	r	r	r	r	r	r	r	Res.	Res.	Res.	Res.	Res.	Res.	DLC[3:0]	
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 30.7.4: Identifier filtering on page 1047 - 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.

### 30.9.4 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.	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.	CANSB[5:0]						Res.	FINIT						
		rw	rw	rw	rw	rw	rw								rw

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

Bits 13:8 **CANSB[5:0]**: CAN start bank

These bits are set and cleared by software. They define the start bank for the CAN interface (Slave) in the range 1 to 27.

Bits 7:1 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.	FBM27	FBM26	FBM25	FBM24	FBM23	FBM22	FBM21	FBM20	FBM19	FBM18	FBM17	FBM16
				rw											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FBM15	FBM14	FBM13	FBM12	FBM11	FBM10	FBM9	FBM8	FBM7	FBM6	FBM5	FBM4	FBM3	FBM2	FBM1	FBM0
rw															

Note: Refer to [Figure 391: Filter bank scale configuration - register organization on page 1049](#)

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

Bits 27:0 **FBMx**: Filter mode

Mode of the registers of Filter x.

0: Two 32-bit registers of filter bank x are in Identifier Mask mode.

1: Two 32-bit registers of filter bank x are in Identifier List mode.

### CAN filter scale register (CAN\_FS1R)

Address offset: 0x20C

Reset value: 0x0000 0000

This register can be written only when the filter initialization mode is set (FINIT=1) in the CAN\_FMR register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	FSC27	FSC26	FSC25	FSC24	FSC23	FSC22	FSC21	FSC20	FSC19	FSC18	FSC17	FSC16
				rw											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FSC15	FSC14	FSC13	FSC12	FSC11	FSC10	FSC9	FSC8	FSC7	FSC6	FSC5	FSC4	FSC3	FSC2	FSC1	FSC0
rw															

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

Bits 27:0 **FScx**: Filter scale configuration

These bits define the scale configuration of Filters 27-0.

0: Dual 16-bit scale configuration

1: Single 32-bit scale configuration

Note: Refer to [Figure 391: Filter bank scale configuration - register organization on page 1049](#).

### 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.	FFA27	FFA26	FFA25	FFA24	FFA23	FFA22	FFA21	FFA20	FFA19	FFA18	FFA17	FFA16
				rw											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FFA15	FFA14	FFA13	FFA12	FFA11	FFA10	FFA9	FFA8	FFA7	FFA6	FFA5	FFA4	FFA3	FFA2	FFA1	FFA0
rw															

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

Bits 27:0 **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.	FACT2 7	FACT2 6	FACT2 5	FACT2 4	FACT2 3	FACT2 2	FACT2 1	FACT2 0	FACT1 9	FACT1 8	FACT1 7	FACT1 6
				rw											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FACT1 5	FACT1 4	FACT1 3	FACT1 2	FACT1 1	FACT1 0	FACT9	FACT8	FACT7	FACT6	FACT5	FACT4	FACT3	FACT2	FACT1	FACT0
rw															

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

Bits 27:0 **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..27, x = 1, 2)

Address offsets: 0x240 to 0x31C

Reset value: 0xFFFF XXXX

*There are 28 filter banks, i=0 to 27. 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 30.7.4: Identifier filtering on page 1047](#).

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 the [Table 213 on page 1078](#).

### 30.9.5 bxCAN register map

Refer to [Section 2.2.2 on page 56](#) for the register boundary addresses. The registers from offset 0x200 to 0x31C are present only in CAN1.

**Table 213. bxCAN register map and reset values**

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6				
0x000	CAN_MCR	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-			
	Reset value	-	Res.	0	LOW[2:0]	0	0	0	1	TME[2:0]	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-				
0x004	CAN_MSR	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-			
	Reset value	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-			
0x008	CAN_TSR	0	Res.	0	LOW[2:0]	0	0	1	1	TME[2:0]	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-				
	Reset value	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-			
0x00C	CAN_RF0R	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-			
	Reset value	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-			
0x010	CAN_RF1R	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-			
	Reset value	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-			
0x014	CAN_IER	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-			
	Reset value	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-	Res.	-			
0x018	CAN_ESR	REC[7:0]						TEC[7:0]						BRP[9:0]						STID[10:0]/EXID[28:18]						EXID[17:0]					
	Reset value	0	0	LBKM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x01C	CAN_BTR	0	SILM	0	TS2[2:0]	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	Res.	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	
0x020-0x17F	-	Res.	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		
0x180	CAN_TI0R	STID[10:0]/EXID[28:18]												EXID[17:0]												IDE					
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		
Res.																															
Res.																															
Res.																															
Res.																															
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 213. bxCAN register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x184	CAN_TDT0R	TIME[15:0]															Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TGT	Res.	Res.	Res.	Res.	DLC[3:0]			
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	x	-	-	-	x	x	x	x	
0x188	CAN_TDL0R	DATA3[7:0]					DATA2[7:0]					DATA1[7:0]					DATA0[7:0]																
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x			
0x18C	CAN_TDHO	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			
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	0			
0x194	CAN_TDT1R	TIME[15:0]															Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TGT	Res.	Res.	Res.	Res.	DLC[3:0]			
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	x	-	-	-	-	x	x	x	
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				
0x19C	CAN_TDHO	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				
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	0				
0x1A4	CAN_TDT2R	TIME[15:0]															Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TGT	Res.	Res.	Res.	Res.	DLC[3:0]			
	Reset value	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	x	-	-	-	-	x	x	x	
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				
0x1AC	CAN_TDHO	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				
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 213. 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	<b>CAN_RDT0R</b>	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			
0x1B8	<b>CAN_RDL0R</b>	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	<b>CAN_RDH0R</b>	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	<b>CAN_RI1R</b>	STID[10:0]/EXID[28:18]															EXID[17:0]													IDE	RTR	x	x	-
	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			
0x1C4	<b>CAN_RDT1R</b>	TIME[15:0]															FMI[7:0]													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	<b>CAN_RDL1R</b>	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	x	x	
0x1CC	<b>CAN_RDH1R</b>	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	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.	Res.	Res.		
0x200	<b>CAN_FMR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
	Reset value	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
0x204	<b>CAN_FM1R</b>	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	FBM[27:0]	
	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	
0x208	-	Res.	-	Res.	-	Res.	-	Res.	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
	-	Res.	-	Res.	-	Res.	-	Res.	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
0x20C	<b>CAN_FS1R</b>	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	FSC[27:0]	
	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	
0x210	-	Res.	-	Res.	-	Res.	-	Res.	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	

Table 213. 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
0x214	CAN_FFA1R	Res.	Res.	Res.	Res.	Res.	Res.	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		
0x218	-	Res.	Res.	Res.	Res.	Res.	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		
0x21C	CAN_FA1R	Res.	Res.	Res.	Res.	Res.	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		
0x220	-	Res.	Res.	Res.	Res.	Res.	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	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		
0x224-0x23F	-	Res.	Res.	Res.	Res.	Res.	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	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		
0x240	CAN_F0R1	FB[31:0]																															
	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		
0x244	CAN_F0R2	FB[31:0]																															
	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		
0x248	CAN_F1R1	FB[31:0]																															
	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		
0x24C	CAN_F1R2	FB[31:0]																															
	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		
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
0x318	CAN_F27R1	FB[31:0]																															
	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		
0x31C	CAN_F27R2	FB[31:0]																															
	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		

## 31 USB on-the-go full-speed/high-speed (OTG\_FS/OTG\_HS)

### 31.1 Introduction

Portions Copyright (c) 2004, 2005 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)
UTMI	USB Transceiver Macrocell Interface
ULPI	UTMI+ Low Pin Interface
LPM	Link power management
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

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 214: OTG\\_HS speeds supported](#) below. OTG\_FS supports the speeds defined in the [Table 215: OTG\\_FS speeds supported](#) below. The USB OTG supports both HNP and SRP. The only external device required is a charge pump for V<sub>BUS</sub> in OTG mode.

**Table 214. 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 215. OTG\_FS speeds supported**

	HS (480 Mb/s)	FS (12 Mb/s)	LS (1.5 Mb/s)
Host mode	-	X	X
Device mode	-	X	-

## 31.2 USB\_OTG main features

The main features can be divided into three categories: general, host-mode and device-mode features.

### 31.2.1 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 3 PHY interfaces
  - An on-chip full-speed PHY
  - An I<sup>2</sup>C interface for external full-speed I<sup>2</sup>C PHY
  - An ULPI interface for external high-speed PHY
- It includes full support (PHY) for the optional On-The-Go (OTG) protocol detailed in the On-The-Go Supplement Rev 1.3 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<sub>BUS</sub> off to conserve battery power in OTG applications
  - It supports OTG monitoring of V<sub>BUS</sub> 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

### 31.2.2 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.

### 31.2.3 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] / 8[HS] dedicated Tx-IN FIFOs (one for each active IN EP) to put less load on the application
- Support for the soft disconnect feature.

### 31.3 USB\_OTG Implementation

### 31.4 USB OTG functional description

#### 31.4.1 USB OTG block diagram

Figure 399. OTG full-speed block diagram

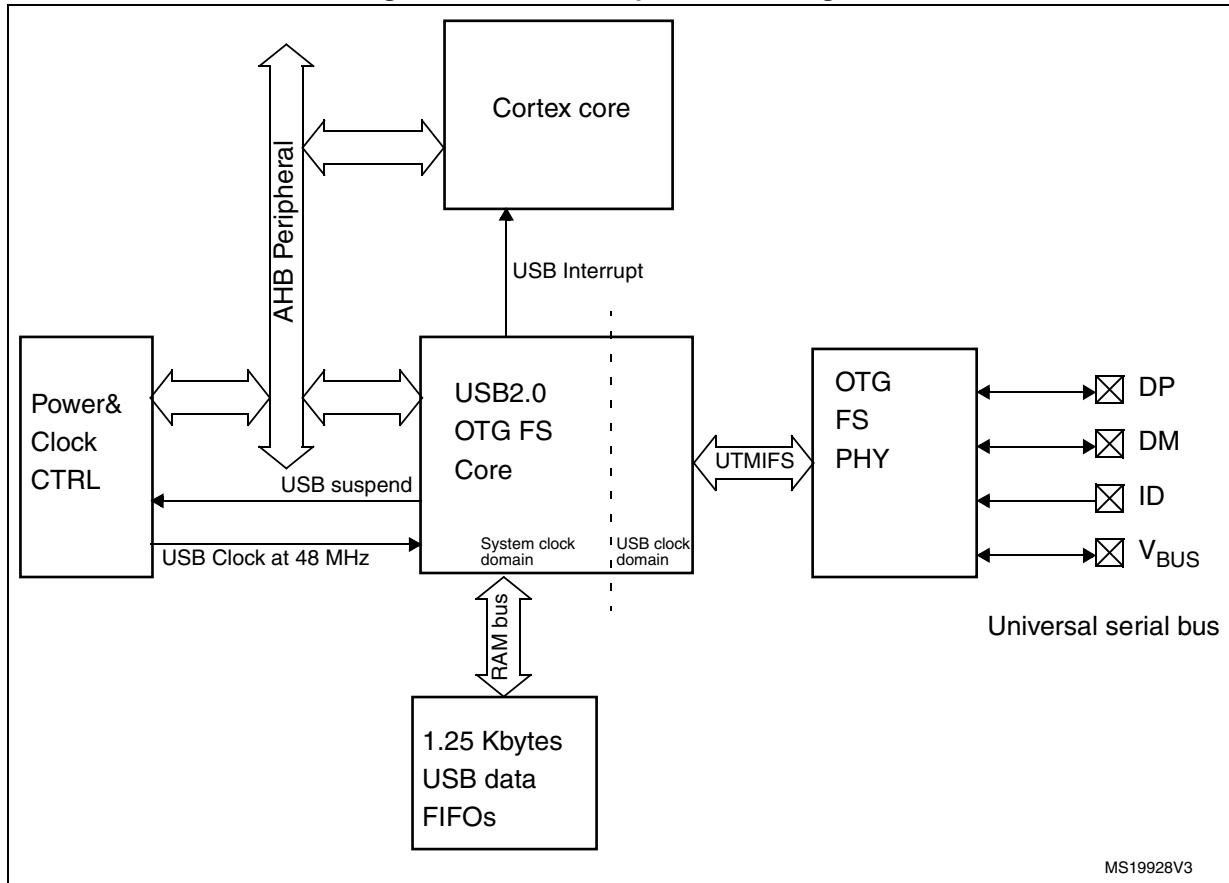
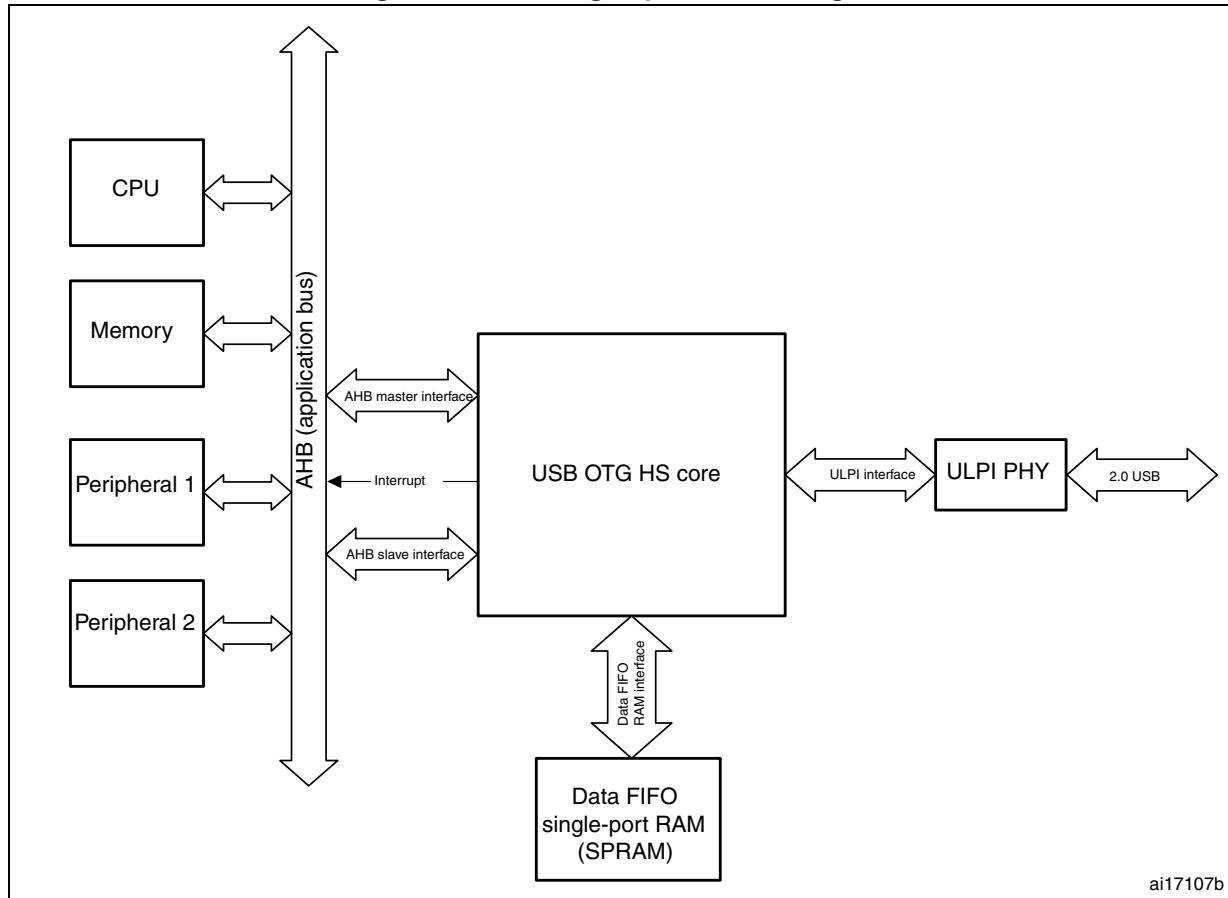


Figure 400. OTG high-speed block diagram



ai17107b

### 31.4.2 OTG core

The USB OTG receives the 48 MHz clock from the reset and clock controller (RCC), via an external quartz. 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 31.13: 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 OTG\_HS PHY or external OTG\_FS PHY using I<sup>2</sup>C interface.

### 31.4.3 Full-speed OTG PHY

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 UTMII+ 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.

**Note:** *The content of this section applies only to USB OTG FS.*

### 31.4.4 Embedded full speed OTG PHY

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 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.

*Note:* The content of this section applies only to USB OTG HS.

### 31.4.5 High-speed OTG PHY

The USB OTG HS core includes an ULPI interface to connect an external HS PHY.

*Note:* The content of this section applies only to USB OTG HS.

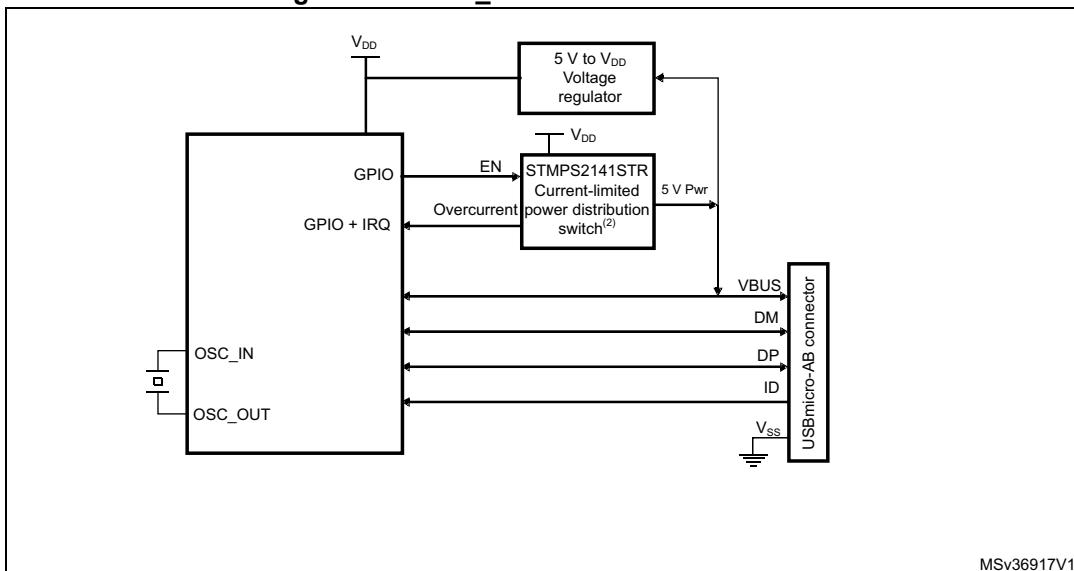
### 31.4.6 External Full-speed OTG PHY using the I<sup>2</sup>C interface

The USB OTG HS core embeds an I<sup>2</sup>C interface allowing to connect an external FS PHY.

*Note:* The content of this section applies only to USB OTG HS.

## 31.5 OTG dual role device (DRD)

Figure 401. OTG\_FS A-B device connection



MSv36917V1

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.

### 31.5.1 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.

### 31.5.2 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 31.16: OTG\\_FS/OTG\\_HS programming model](#).

### 31.5.3 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<sub>BUS</sub> for the A-device to save power. Note that the A-device is always in charge of driving V<sub>BUS</sub> 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 31.16: OTG\\_FS/OTG\\_HS programming model](#).

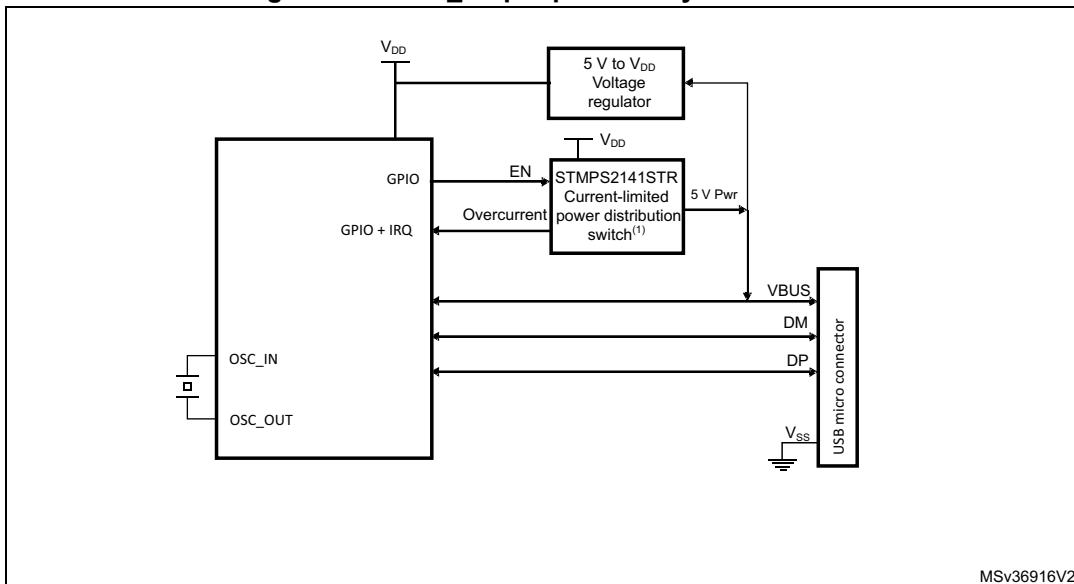
## 31.6 USB peripheral

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 402: USB\\_FS peripheral-only connection](#))
  - The force device mode bit (FDMOD) in the [Section 31.15.4: 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  $V_{DD}$  chip-supply from  $V_{BUS}$ .

**Figure 402. USB\_FS peripheral-only connection**



1. Use a regulator to build a bus-powered device.

### 31.6.1 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.

### 31.6.2 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.

### 31.6.3 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

## 31.7 USB host

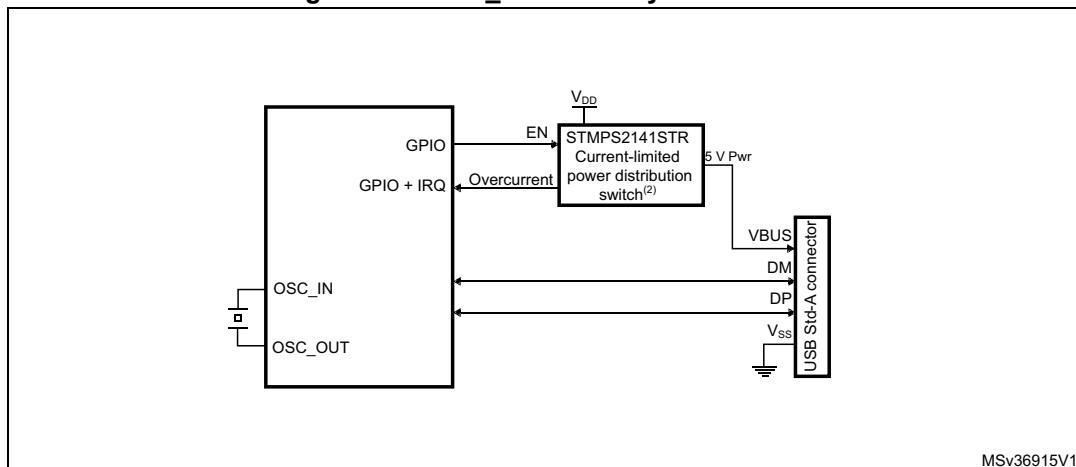
This section gives the functional description of the OTG\_FS/OTG\_HS in the USB host mode. The OTG\_FS/OTG\_HS works as a USB host in the following circumstances:

- 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 in the [31.15.4](#) global USB configuration register (FHMOD bit in 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 403. USB\_FS host-only connection**



1. STMPS2141STR needed only if the application has to support a  $V_{BUS}$  powered device. A basic power switch can be used if 5 V are available on the application board.
2.  $V_{DD}$  range is between 2 V and 3.6 V.

### 31.7.1 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.

### 31.7.2 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. When the application decides to power on  $V_{BUS}$  using the chosen GPIO, it must also set the port power bit in the host port control and status register (PPWR bit in OTG\_HPRT).

#### $V_{BUS}$ valid

When HNP or SRP is enabled the VBUS 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 VBUS sensing pin does not need to be connected to  $V_{BUS}$  and it can be used as GPIO.

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.

## 31.7.3 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

### 31.7.4 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:

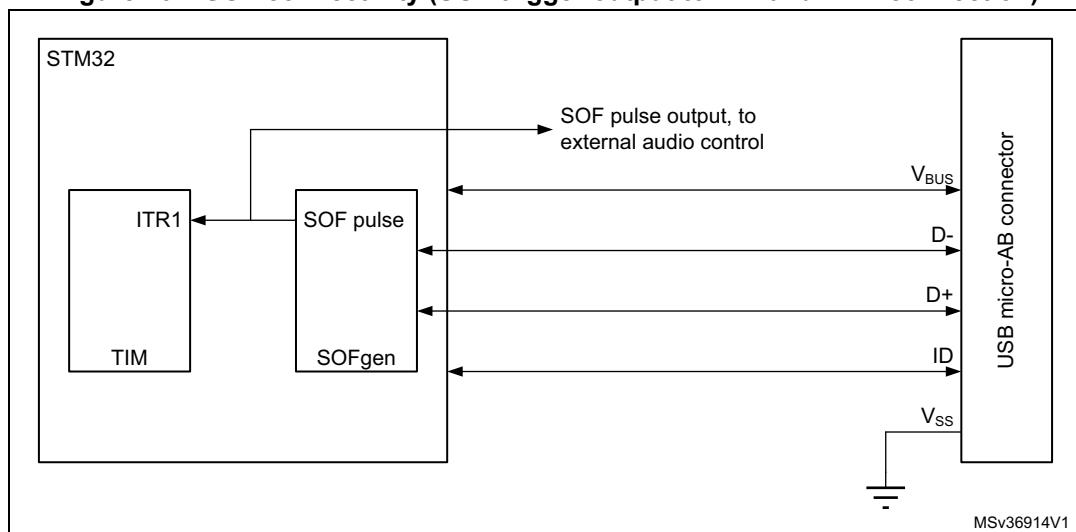
- The number of free entries currently available in the periodic (nonperiodic) request queue (8 max)
- Free space currently available in the periodic (nonperiodic) Tx FIFO (out-transactions)
- IN/OUT token, host channel number and other status information.

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 nonperiodic 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.

## 31.8 SOF trigger

**Figure 404. 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.

### 31.8.1 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 12 system clock 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.

### 31.8.2 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 12 system clock 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.

## 31.9 Power options

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<sub>BUS</sub> detection enable (OTG\_GCCFG/VBDEN)**  
It switches on/off the V<sub>BUS</sub> 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.

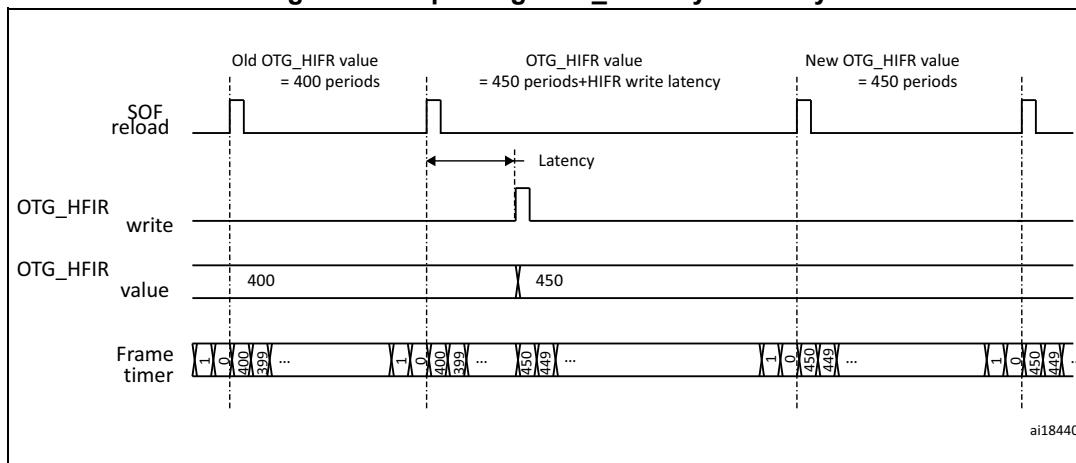
To save dynamic power, the USB data FIFO is clocked only when accessed by the OTG\_FS/OTG\_HS core.

## 31.10 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 405](#).

**Figure 405. Updating OTG\_HFIR dynamically**

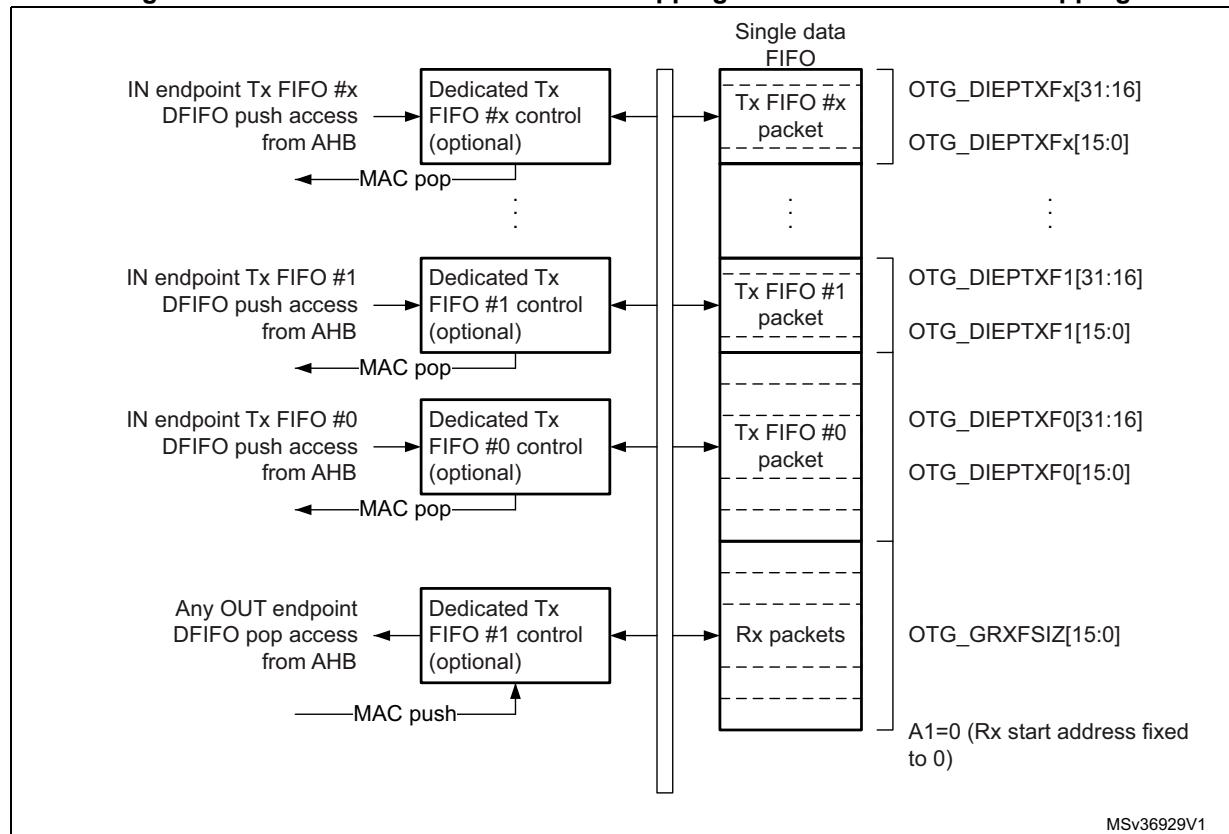


## 31.11 USB data FIFOs

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.

### 31.11.1 Peripheral FIFO architecture

**Figure 406. Device-mode FIFO address mapping and AHB FIFO access mapping**



MSv36929V1

#### 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

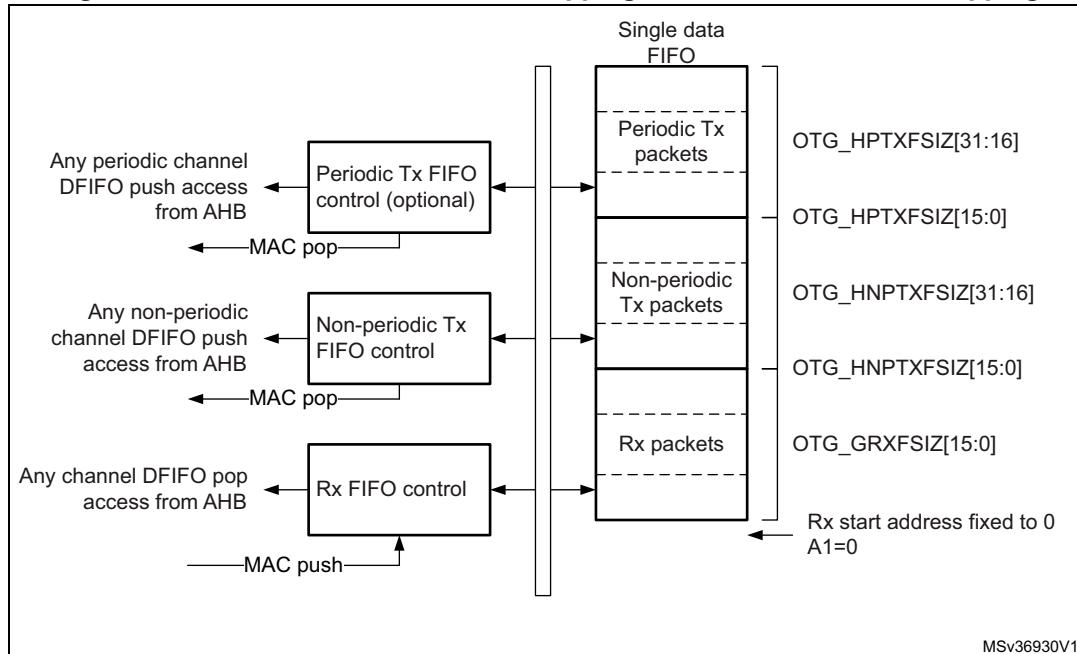
The application keeps receiving the Rx FIFO non-empty interrupt (RXFLVL bit in OTG\_GINTSTS) as long as there is at least one packet available for download. It reads the packet information from the receive status read and pop register (OTG\_GRXSTSP) and finally pops data off the receive FIFO by reading 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- $x$ .

### 31.11.2 Host FIFO architecture

**Figure 407. 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.

### 31.11.3 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 =

$(4 * \text{number of control endpoints} + 6) + ((\text{largest USB packet used} / 4) + 1 \text{ for status information}) + (2 * \text{number of OUT endpoints}) + 1 \text{ for Global NAK}$

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 =  $(4 * 1 + 6) + ((1,024 / 4) + 1) + (2 * 4) + 1 = 276$

**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.

## 31.12 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

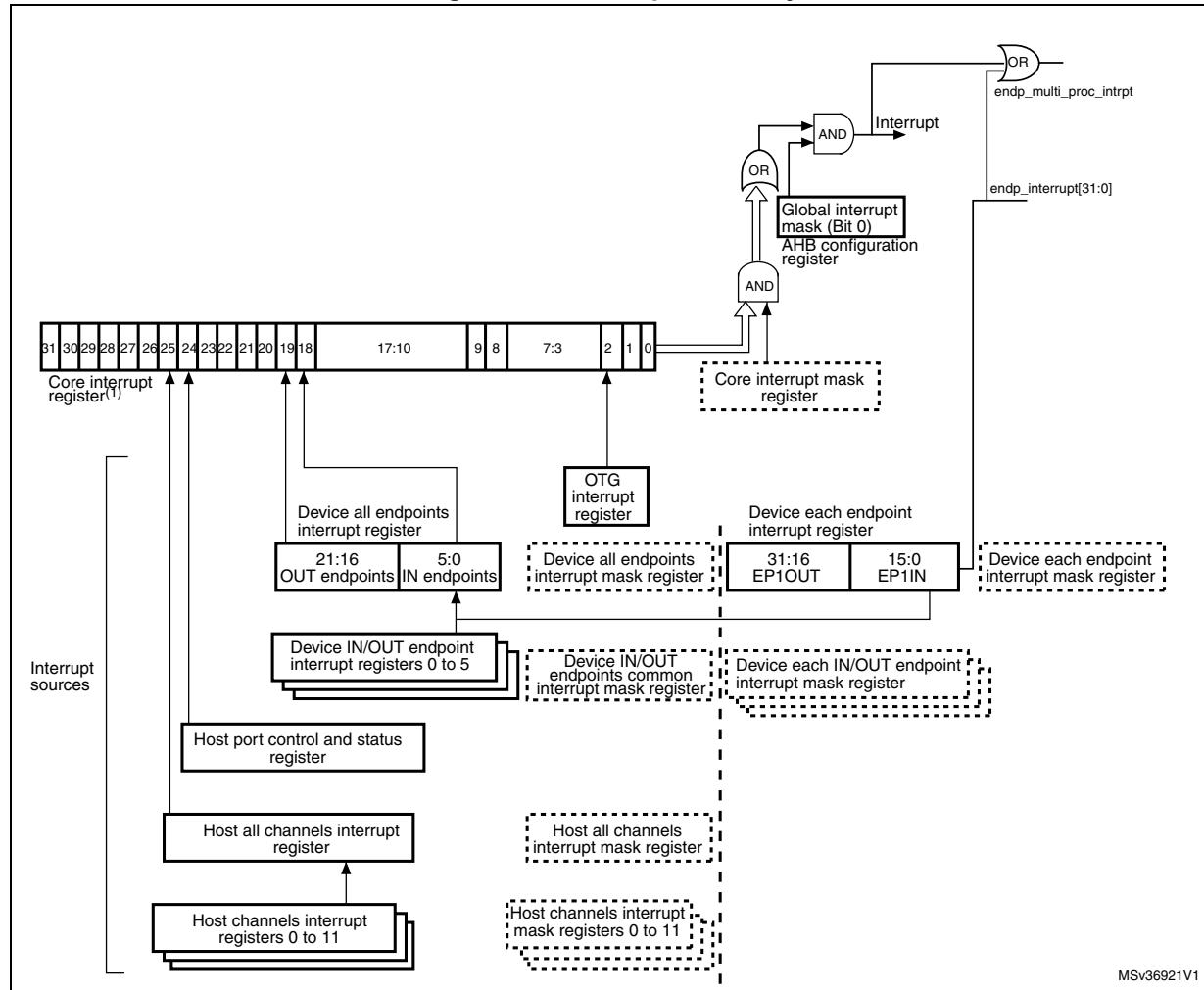
As the OTG\_FS core is able to fill in the 1.25-Kbyte RAM buffer very efficiently, and as 1.25-Kbyte of transmit/receive data is more than enough to cover a full speed frame, the USB system is able to withstand the maximum full-speed data rate for up to one USB frame (1 ms) without any CPU intervention.

## 31.13 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 408* shows the interrupt hierarchy.

Figure 408. Interrupt hierarchy



1. The core interrupt register bits are shown in [OTG core interrupt register \(OTG\\_GINTSTS\) on page 1126](#).

## 31.14 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.

### 31.14.1 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 216. Core global control and status registers (CSRs)**

Acronym	Address offset	Register name
OTG_GOTGCTL	0x000	<i>OTG control and status register (OTG_GOTGCTL) on page 1114</i>
OTG_GOTGINT	0x004	<i>OTG interrupt register (OTG_GOTGINT) on page 1117</i>
OTG_GAHBCFG	0x008	<i>OTG AHB configuration register (OTG_GAHBCFG) on page 1118</i>
OTG_GUSBCFG	0x00C	<i>OTG USB configuration register (OTG_GUSBCFG) on page 1120</i>
OTG_GRSTCTL	0x010	<i>OTG reset register (OTG_GRSTCTL) on page 1124</i>
OTG_GINTSTS	0x014	<i>OTG core interrupt register (OTG_GINTSTS) on page 1126</i>
OTG_GINTMSK	0x018	<i>OTG interrupt mask register (OTG_GINTMSK) on page 1131</i>

**Table 216. Core global control and status registers (CSRs) (continued)**

Acronym	Address offset	Register name
OTG_GRXSTSR	0x01C	<i>OTG_FS Receive status debug read/OTG status read and pop registers (OTG_GRXSTSR/OTG_GRXSTSP) on page 1135</i>
OTG_GRXSTSP	0x020	
OTG_GRXFSIZ	0x024	<i>OTG Receive FIFO size register (OTG_GRXFSIZ) on page 1136</i>
OTG_HNPTXFSIZ/ OTG_DIEPTXF0 <sup>(1)</sup>	0x028	<i>OTG Host non-periodic transmit FIFO size register (OTG_HNPTXFSIZ)/Endpoint 0 Transmit FIFO size (OTG_DIEPTXF0)</i>
OTG_HNPTXSTS	0x02C	<i>OTG non-periodic transmit FIFO/queue status register (OTG_HNPTXSTS) on page 1138</i>
OTG_GI2CCTL	0x030	<i>OTG I2C access register (OTG_GI2CCTL) on page 1139</i>
OTG_GCCFG	0x038	<i>OTG general core configuration register (OTG_GCCFG) on page 1140</i>
OTG_CID	0x03C	<i>OTG core ID register (OTG_CID) on page 1141</i>
OTG_GLPMCFG	0x54	<i>OTG core LPM configuration register (OTG_GLPMCFG) on page 1141</i>
OTG_HPTXFSIZ	0x100	<i>OTG Host periodic transmit FIFO size register (OTG_HPTXFSIZ) on page 1145</i>
OTG_DIEPTXF <sub>x</sub>	0x104 0x124 ... 0x184	<i>OTG device IN endpoint transmit FIFO size register (OTG_DIEPTXF<sub>x</sub>) (<sub>x</sub> = 1..5[FS]/8[HS], where <sub>x</sub> is the FIFO_number) on page 1145 for USB_OTG FS</i>
OTG_DIEPTXF <sub>x</sub>	0x104 0x124 ... 0x1B4	<i>OTG device IN endpoint transmit FIFO size register (OTG_DIEPTXF<sub>x</sub>) (<sub>x</sub> = 1..5[FS]/8[HS], where <sub>x</sub> is the FIFO_number) on page 1145 for USB_OTG HS</i>

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 217. Host-mode control and status registers (CSRs)**

Acronym	Offset address	Register name
OTG_HCFG	0x400	<i>OTG Host configuration register (OTG_HCFG) on page 1146</i>
OTG_HFIR	0x404	<i>OTG Host frame interval register (OTG_HFIR) on page 1147</i>
OTG_HFNUM	0x408	<i>OTG Host frame number/frame time remaining register (OTG_HFNUM) on page 1148</i>
OTG_HPTXSTS	0x410	<i>OTG Host periodic transmit FIFO/queue status register (OTG_HPTXSTS) on page 1148</i>
OTG_HAINT	0x414	<i>OTG Host all channels interrupt register (OTG_HAINT) on page 1149</i>

**Table 217. Host-mode control and status registers (CSRs) (continued)**

Acronym	Offset address	Register name
OTG_HAINTMSK	0x418	<i>OTG Host all channels interrupt mask register (OTG_HAINTMSK) on page 1150</i>
OTG_HPRT	0x440	<i>OTG Host port control and status register (OTG_HPRT) on page 1150</i>
OTG_HCCHARx	0x500 0x520 ... 0x660	<i>OTG Host channel-x characteristics register (OTG_HCCHARx) (x = 0..15[HS] / 11[FS], where x = Channel_number) on page 1153 for USB_OTG FS</i>
OTG_HCCHARx	0x500 0x520 ... 0x6E0	<i>OTG Host channel-x characteristics register (OTG_HCCHARx) (x = 0..15[HS] / 11[FS], where x = Channel_number) on page 1153 for USB_OTG HS</i>
OTG_HCSPLTx	0x504 0x524 ... 0x6E4	<i>OTG Host channel-x split control register (OTG_HCSPLTx) (x = 0..15, where x = Channel_number) on page 1154</i>
OTG_HCDMAX	0x514 0x534 ... 0x6F4	<i>OTG Host channel-x DMA address register (OTG_HCDMAX) (x = 0..15, where x = Channel_number) on page 1160</i>
OTG_HCINTx	0x508 0x528 ... 0x668	<i>OTG Host channel-x interrupt register (OTG_HCINTx) (x = 0..15[HS] / 11[FS], where x = Channel_number) on page 1155 for USB_OTG FS</i>
OTG_HCINTx	0x508 0x528 ... 0x6E8	<i>OTG Host channel-x interrupt register (OTG_HCINTx) (x = 0..15[HS] / 11[FS], where x = Channel_number) on page 1155 for USB_OTG HS</i>
OTG_HCINTMSKx	0x50C 0x52C ... 0x66C	<i>OTG Host channel-x interrupt mask register (OTG_HCINTMSKx) (x = 0..15[HS] / 11[FS], where x = Channel_number) on page 1157 for USB_OTG FS</i>
OTG_HCINTMSKx	0x50C 0x52C ... 0x6EC	<i>OTG Host channel-x interrupt mask register (OTG_HCINTMSKx) (x = 0..15[HS] / 11[FS], where x = Channel_number) on page 1157 for USB_OTG HS</i>

**Table 217. Host-mode control and status registers (CSRs) (continued)**

Acronym	Offset address	Register name
OTG_HCTSIZx	0x510 0x530 .... 0x670	<i>OTG Host channel-x transfer size register (OTG_HCTSIZx) (x = 0..15[HS] / 11[FS], where x = Channel_number) on page 1158 for USB_OTG FS</i>
OTG_HCTSIZx	0x510 0x530 .... 0x6F0	<i>OTG Host channel-x transfer size register (OTG_HCTSIZx) (x = 0..15[HS] / 11[FS], where x = Channel_number) on page 1158 for USB_OTG HS</i>

**Device-mode CSR map**

These registers must be programmed every time the core changes to device mode.

**Table 218. Device-mode control and status registers**

Acronym	Offset address	Register name
OTG_DCFG	0x800	<i>OTG device configuration register (OTG_DCFG) on page 1160</i>
OTG_DCTL	0x804	<i>OTG device control register (OTG_DCTL) on page 1162</i>
OTG_DSTS	0x808	<i>OTG device status register (OTG_DSTS) on page 1165</i>
OTG_DIEPMSK	0x810	<i>OTG device IN endpoint common interrupt mask register (OTG_DIEPMSK) on page 1166</i>
OTG_DOEPMSK	0x814	<i>OTG device OUT endpoint common interrupt mask register (OTG_DOEPMSK) on page 1167</i>
OTG_DAINT	0x818	<i>OTG device all endpoints interrupt register (OTG_DAINT) on page 1168</i>
OTG_DAINTMSK	0x81C	<i>OTG all endpoints interrupt mask register (OTG_DAINTMSK) on page 1169</i>
OTG_DVBUUSDIS	0x828	<i>OTG device VBUS discharge time register (OTG_DVBUUSDIS) on page 1170</i>
OTG_DVBUSPULSE	0x82C	<i>OTG device VBUS pulsing time register (OTG_DVBUSPULSE) on page 1170</i>
OTG_DTHRCTL	0x830	<i>OTG Device threshold control register (OTG_DTHRCTL) on page 1171</i>
OTG_DIEPEMPMSK	0x834	<i>OTG device IN endpoint FIFO empty interrupt mask register (OTG_DIEPEMPMSK) on page 1172</i>
OTG_DEACHINT	0x838	<i>OTG device each endpoint interrupt register (OTG_DEACHINT) on page 1172</i>
OTG_DEACHINTMSK	0x83C	<i>OTG device each endpoint interrupt register mask (OTG_DEACHINTMSK) on page 1173</i>

Table 218. Device-mode control and status registers (continued)

Acronym	Offset address	Register name
OTG_DIEPCTL0	0x900	<i>OTG device control IN endpoint 0 control register (OTG_DIEPCTL0) on page 1173 for USB_OTG FS</i>
OTG_DIEPCTLx	0x920 0x940 ... 0x9A0	<i>OTG device endpoint-x control register (OTG_DIEPCTLx) (x = 1..5[FS] / 0..8[HS], where x = Endpoint_number) on page 1175 for USB_OTG FS</i>
OTG_DIEPCTLx	0x900 0x920 ... 0x9E0	<i>OTG device endpoint-x control register (OTG_DIEPCTLx) (x = 1..5[FS] / 0..8[HS], where x = Endpoint_number) on page 1175 for USB_OTG HS</i>
OTG_DIEPINTx	0x908 0x928 ... 0x9A8	<i>OTG device endpoint-x interrupt register (OTG_DIEPINTx) (x = 0..5[FS] / 8[HS], where x = Endpoint_number) on page 1181 for USB_OTG FS</i>
OTG_DIEPINTx	0x908 0x928 ... 0x9E8	<i>OTG device endpoint-x interrupt register (OTG_DIEPINTx) (x = 0..5[FS] / 8[HS], where x = Endpoint_number) on page 1181 for USB_OTG HS</i>
OTG_DIEPTSIZ0	0x910	<i>OTG device IN endpoint 0 transfer size register (OTG_DIEPTSIZ0) on page 1184</i>
OTG_DIEPDMAx	0x914	<i>OTG Device channel-x DMA address register (OTG_DIEPDMAx) (x = 0..15, where x= Channel_number) on page 1184</i>
OTG_DTXFSTSx	0x918 0x938 ... 0x9B8	<i>OTG device IN endpoint transmit FIFO status register (OTG_DTXFSTSx) (x = 0..5[FS] / 8[HS], where x = Endpoint_number) on page 1187 for USB_OTG FS</i>
OTG_DTXFSTSx	0x918 0x938 ... 0x9F8	<i>OTG device IN endpoint transmit FIFO status register (OTG_DTXFSTSx) (x = 0..5[FS] / 8[HS], where x = Endpoint_number) on page 1187 for USB_OTG HS</i>
OTG_DIEPTSIZx	0x930 0x950 ... 0x9B0	<i>OTG device IN endpoint-x transfer size register (OTG_DIEPTSIZx) (x = 1..5[FS] / 8[HS], where x= Endpoint_number) on page 1186 for USB_OTG FS</i>
OTG_DIEPTSIZx	0x930 0x950 ... 0x9F0	<i>OTG device IN endpoint-x transfer size register (OTG_DIEPTSIZx) (x = 1..5[FS] / 8[HS], where x= Endpoint_number) on page 1186 for USB_OTG HS</i>
OTG_DOEPCTL0	0xB00	<i>OTG device control OUT endpoint 0 control register (OTG_DOEPCTL0) on page 1177</i>

**Table 218. Device-mode control and status registers (continued)**

Acronym	Offset address	Register name
OTG_DOEPDMAx	0xB14	<i>OTG Device channel-x DMA address register (OTG_DOEPDMAx) (x = 0..15, where x = Channel_number) on page 1185</i>
OTG_DOEPCTLx	0xB20 0xB40 ... 0xBA0	<i>OTG device endpoint-x control register (OTG_DOEPCTLx) (x = 1..5[FS]/8[HS], where x = Endpoint_number) on page 1179 for USB_OTG FS</i>
OTG_DOEPCTLx	0xB20 0xB40 ... 0xBE0	<i>OTG device endpoint-x control register (OTG_DOEPCTLx) (x = 1..5[FS]/8[HS], where x = Endpoint_number) on page 1179 for USB_OTG HS</i>
OTG_DOEPINTx	0xB08 0xB28 ... 0xBA8	<i>OTG device endpoint-x interrupt register (OTG_DOEPINTx) (x = 0..5[FS]/8[HS], where x = Endpoint_number) on page 1183 for USB_OTG FS</i>
OTG_DOEPINTx	0xB08 0XB28 ... 0xBE8	<i>OTG device endpoint-x interrupt register (OTG_DOEPINTx) (x = 0..5[FS]/8[HS], where x = Endpoint_number) on page 1183 for USB_OTG HS</i>
OTG_DOEPTSIZ0	0xB10	<i>OTG device OUT endpoint 0 transfer size register (OTG_DOEPTSIZ0) on page 1185</i>
OTG_DOEPTSIZx	0xB30 0xB50 ... 0xBB0	<i>OTG device OUT endpoint-x transfer size register (OTG_DOEPTSIZx) (x = 1..5[FS]/8[HS], where x = Endpoint_number) on page 1188 for USB_OTG FS</i>
OTG_DOEPTSIZx	0xB30 0xB50 ... 0xBF0	<i>OTG device OUT endpoint-x transfer size register (OTG_DOEPTSIZx) (x = 1..5[FS]/8[HS], where x = Endpoint_number) on page 1188 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 219. Data FIFO (DFIFO) access register map**

FIFO access register section	Address range	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
...	...	...
Device IN Endpoint x <sup>(1)</sup> /Host OUT Channel x <sup>(1)</sup> : DFIFO Write Access Device OUT Endpoint x <sup>(1)</sup> /Host IN Channel x <sup>(1)</sup> : 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 220. Power and clock gating control and status registers**

Register name	Acronym	Offset address: 0xE00–0xFFFF
Power and clock gating control register	PCGCCTL	0xE00–0xE04
Reserved	-	0xE05–0xFFFF

## 31.15 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.

### 31.15.1 OTG control and status register (OTG\_GOTGCTL)

Address offset: 0x000

Reset value: 0x0X01 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.	Res.	OTG VER	BSVLD	ASVLD	DBCT	CID STS
											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 31:21 Reserved, must be kept at reset value.

Bit 20 **OTGVER:** OTG version

Selects the OTG revision.

0:OTG Version 1.3.

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, you 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 **DHNPNPEN:** 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 **HSHNPNPEN:** 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<sub>BUS</sub> 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<sub>BUS</sub> 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 you use the USB 1.1 full-speed serial transceiver interface to initiate the session request, the application must wait until V<sub>BUS</sub> discharges to 0.2 V, after the B-Session Valid bit in this register (BSVLD bit in OTG\_GOTGCTL) is cleared. This discharge time varies between different PHYs and can be obtained from the PHY vendor.

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.*

### 31.15.2 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.	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<sub>BUS</sub> is no longer valid for a B-Peripheral session when V<sub>BUS</sub> < 0.8 V.

*Note: Accessible in both device and host modes.*

Bits 1:0 Reserved, must be kept at reset value.

### 31.15.3 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					GINT MSK						
							rw	rw		rw	rw	rw	rw	rw	rw

Note: Configuration register for USB OTG HS

Bits 31:20 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**: 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.

#### 31.15.4 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.	ULPIIP D	PTCI	PCCI	TSDPS	ULPIE VBUSI	ULPIE VBUSD	ULPIC SM.	ULPIA R.	ULPIFS L.	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				HNP CAP	SRP CAP	Res.	PHY SEL	Res.		Res.	TOCAL		
	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 for USB OTG HS and FS

Bits 25:15 Reserved, must be kept at reset value for USB OTG FS

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 for USB OTG HS.

Bit 15 **PHYLPCS:** 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:** USB turnaround time

These bits allows to set the turnaround time in PHY clocks. They must be configured according to [Table 221: TRDT values \(FS\)](#) or [Table 222: 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<sub>BUS</sub> 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.
- 1: USB 1.1 full-speed serial transceiver.

Bits 5:3 Reserved, must be kept at reset value.

Bits 2:0 **TOCAL**: 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 221. 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 222. TRDT values (HS)**

AHB frequency range (MHz)		TRDT minimum value
Min	Max	
30	-	0x9

### 31.15.5 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						TXF FLSH	RXF FLSH	Res.	Res.	PSRST	CSRST
					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**: 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 you dynamically change 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.*

### 31.15.6 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<sub>BUS</sub> 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 ISOIXFR:** 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\_DAINT 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.*

### 31.15.7 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.	IPXFRE M/IISO OXFR M	IISOIX FRM	OEPINT T	IEPINT	Res.	Res.
rw	rw	rw	rw	rw	rw	rw	r	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	USBRS T	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	USBRS T	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 device 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.

### 31.15.8 OTG\_FS Receive status debug read/OTG status read and pop registers (OTG\_GRXSTS/OTG\_GRXSTSP)

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.	PKTSTS				DPID											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
DPID	BCNT												CHNUM			
r	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**: 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**: Byte count

Indicates the byte count of the received IN data packet.

Bits 3:0 **CHNUM**: 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.									DPID						
							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															EPNUM
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 24:21 **FRMNUM:** 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:** 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:** Data PID

Indicates the Data PID of the received OUT data packet

00: DATA0

10: DATA1

01: DATA2

11: MDATA

Bits 14:4 **BCNT:** Byte count

Indicates the byte count of the received data packet.

Bits 3:0 **EPNUM:** Endpoint number

Indicates the endpoint number to which the current received packet belongs.

**31.15.9 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.															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RXFD															
rw															

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

Bits 15:0 **RXFD:** 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.

### 31.15.10 OTG Host non-periodic transmit FIFO size register (OTG\_HNPTXF0SIZ)/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
NPTXF0D/TX0FD															
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
NPTXF0A/TX0FSA															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Host mode

Bits 31:16 **NPTXF0D:** 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 **NPTXF0A:** 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.

### 31.15.11 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							NPTQXS AV							
	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															
	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:** 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 **NPTQXS AV:** 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:** 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

### 31.15.12 OTG I<sup>2</sup>C access register (OTG\_GI2CCTL)

Address offset: 0x030

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BSY DNE	RW.	Res.	I2CD ATSE	I2CDEVADR	Res.	ACK	I2CEN	ADDR							
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
REGADDR								RWDATA							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 **BSYDNE**: I<sup>2</sup>C Busy/Done

The application sets this bit to 1 to start a request on the I<sup>2</sup>C interface. When the transfer is complete, the core deasserts this bit to 0. As long as the bit is set indicating that the I<sup>2</sup>C interface is busy, the application cannot start another request on the interface.

Bit 30 **RW**: Read/Write Indicator

This bit indicates whether a read or write register transfer must be performed on the interface.

0: Write

1: Read

*Note: Read/write bursting is not supported for registers.*

Bit 29 Reserved, must be kept at reset value.

Bit 28 **I2CDATSE0**: I<sup>2</sup>C DatSe0 USB mode

This bit is used to select the full-speed interface USB mode.

0: VP\_VM USB mode

1: DAT\_SE0 USB mode

Bits 27:26 **I2CDEVADR**: I<sup>2</sup>C Device Address

This bit selects the address of the I<sup>2</sup>C slave on the USB 1.1 full-speed serial transceiver corresponding to the one used by the core for OTG signalling.

Bit 25 Reserved, must be kept at reset value.

Bit 24 **ACK**: I<sup>2</sup>C ACK

This bit indicates whether an ACK response was received from the I<sup>2</sup>C slave. It is valid when BSYDNE is cleared by the core, after the application has initiated an I<sup>2</sup>C access.

0: NAK

1: ACK

Bit 23 **I2CEN**: I<sup>2</sup>C Enable

This bit enables the I<sup>2</sup>C master to initiate transactions on the I<sup>2</sup>C interface.

Bits 22:16 **ADDR**: I<sup>2</sup>C Address

This is the 7-bit I<sup>2</sup>C device address used by the application to access any external I<sup>2</sup>C slave, including the I<sup>2</sup>C slave on a USB 1.1 OTG full-speed serial transceiver.

Bits 15:8 **REGADDR**: I<sup>2</sup>C Register Address

These bits allow to program the address of the register to be read from or written to.

Bits 7:0 **RWDATA**: I<sup>2</sup>C Read/Write Data

After a register read operation, these bits hold the read data for the application.  
During a write operation, the application can use this register to program the data to be written to a register.

*Note:* Configuration register applies only to USB OTG HS

**31.15.13 OTG general core configuration register (OTG\_GCCFG)**

Address offset: 0x038

Reset value: 0x0000 XXX0

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

Bits 31:22 Reserved, must be kept at reset value.

Bit 21 **VBDEN**: USB V<sub>BUS</sub> detection enable

Enables V<sub>BUS</sub> sensing comparators to detect V<sub>BUS</sub> valid levels on the V<sub>BUS</sub> PAD for USB host and device operation. If HNP and/or SRP support is enabled, V<sub>BUS</sub> comparators are automatically enabled independently of VBDEN value.

0 = V<sub>BUS</sub> Detection Disabled

1 = V<sub>BUS</sub> Detection Enabled

Bits 20:17 Reserved, must be kept at reset value.

Bit 16 **PWRDWN**: Power down control

Used to activate the transceiver in transmission/reception. When reset, the transceiver is kept in power-down. 0 = USB FS transceiver disabled

1 = USB FS transceiver enabled

Bits 15:4 Reserved, must be kept at reset value.

Bits 3:0 Reserved, must be kept at reset value.

### 31.15.14 OTG core ID register (OTG\_CID)

Address offset: 0x03C

Reset value: 0x0000 3X00 for USB OTG FS

Reset value: 0x0000 3100 for USB OTG HS

This is a read only register containing the Product ID.

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

Bits 31:0 **PRODUCT\_ID**: Product ID field

Application-programmable ID field.

### 31.15.15 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			SND LPM	LPMRCNT			LPMCHIDX			L1RSM OK	
			<b>rw</b>	<b>r</b>	<b>r</b>	<b>r</b>	<b>rs</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>r</b>
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SLP STS	LPMRSP		L1DS EN	BESLTHRS			L1SS EN	REM WAKE	BESL			LPM ACK	LPM EN		
<b>r</b>	<b>r</b>	<b>r</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw/r</b>	<b>rw/r</b>	<b>rw/r</b>	<b>rw/r</b>	<b>rw</b>	<b>rw</b>		

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**: 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:** 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:** 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:** 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:** 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 **L1SSEN:** 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:** 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] $T_{BESL}$  ( $\mu$ 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

### 31.15.16 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															
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															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **PTXFD**: Host periodic Tx FIFO depth

This value is in terms of 32-bit words.

Minimum value is 16

Bits 15:0 **PTXSA**: Host periodic Tx FIFO start address

This field configures the memory start address for periodic transmit FIFO RAM.

### 31.15.17 OTG device IN endpoint transmit FIFO size register (OTG\_DIEPTXF $x$ ) ( $x = 1..5[FS] / 8[HS]$ , where $x$ is the FIFO\_number)

Address offset: 0x104 + (FIFO\_number – 1) × 0x04

Reset values:

FIFO\_number = 8[HS] / 5[FS]: 0x0200 0200 + (8[HS] / 5[FS] \* 0x200)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
INEPTXFD															
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															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **INEPTXFD**: IN endpoint Tx FIFO depth

This value is in terms of 32-bit words.

Minimum value is 16

Bits 15:0 **INEPTXSA**: 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.

### 31.15.18 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:

### 31.15.19 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													
													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**: 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).*

### 31.15.20 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.	RLD CTRL
															rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FRIVL															
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 cannot be reloaded dynamically

1: The HFIR can be dynamically reloaded during runtime.

This bit needs to be programmed during initial configuration and its value must not be changed during runtime.

Bits 15:0 **FRIVL**: 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.

Bits 15:0 **FRIVL**: 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.

### 31.15.21 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															
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															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 **FTREM**: 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**: Frame number

This field increments when a new SOF is transmitted on the USB, and is cleared to 0 when it reaches 0x3FFF.

### 31.15.22 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								PTXQSAV							
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															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:24 **PTXQTOP:** 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:** 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

bxn: n locations available ( $0 \leq n \leq 8$ )

Others: Reserved

Bits 15:0 **PTXFSAVL:** 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

bxn: n words available (where  $0 \leq n \leq \text{PTXFD}$ )

Others: Reserved

### 31.15.23 OTG Host all channels interrupt register (OTG\_HAIT)

Address offset: 0x414

Reset value: 0x0000 000

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 408](#). 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.
HAINT															
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 **HAIINT**: Channel interrupts

One bit per channel: Bit 0 for Channel 0, bit 15 for Channel 15

### 31.15.24 OTG Host all channels interrupt mask register (OTG\_HAIINTMSK)

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.
HAIINTM															
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 **HAIINTM**: Channel interrupt mask

0: Masked interrupt

1: Unmasked interrupt

One bit per channel: Bit 0 for channel 0, bit 15 for channel 15

### 31.15.25 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 408](#). 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	PTCTL	
													r	r	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PTCTL		PPWR	PLSTS		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:** 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:** 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:** 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 (WKUINT 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 (WKUINT 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\_GLPMCFG 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 POCHNG:** 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

### 31.15.26 OTG Host channel-x characteristics register (OTG\_HCCHARx) (x = 0..15[HS] / 11[FS], where x = Channel\_number)

Address offset: 0x500 + (Channel\_number × 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				MCNT				EPTYP		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				MPSIZ										
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:** Device address

This field selects the specific device serving as the data source or sink.

Bits 21:20 **MCNT:** 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:** 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:** Endpoint number

Indicates the endpoint number on the device serving as the data source or sink.

Bits 10:0 **MPSIZ:** Maximum packet size

Indicates the maximum packet size of the associated endpoint.

### 31.15.27 OTG Host channel-x split control register (OTG\_HCSPLTx) (x = 0..15, where x = Channel\_number)

Address offset: 0x504 + (Channel\_number × 0x20)

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	COMP LSPLT
SPLIT EN	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	
XACTPOS		HUBADDR								PRTADDR						
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**: 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**: Hub address

This field holds the device address of the transaction translator's hub.

Bits 6:0 **PRTADDR**: Port address

This field is the port number of the recipient transaction translator.

*Note:* Configuration register applies only to USB OTG HS

### 31.15.28 OTG Host channel-x interrupt register (OTG\_HCINTx) (x = 0..15[HS] / 11[FS], where x = Channel\_number)

Address offset: 0x508 + (Channel\_number × 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 408](#). 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	XFCR
					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.															

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	AHBE RR	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.

### 31.15.29 OTG Host channel-x interrupt mask register (OTG\_HCINTMSK $x$ ) ( $x = 0..15[\text{HS}] / 11[\text{FS}], \text{where } x = \text{Channel\_number}$ )

Address offset: 0x50C + (Channel\_number × 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 **AHBERR:** 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

### 31.15.30 OTG Host channel-x transfer size register (OTG\_HCTSIZx) ( $x = 0..15[\text{HS}] / 11[\text{FS}], \text{where } x = \text{Channel\_number}$ )

Address offset: 0x510 + (Channel\_number × 0x20)

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Res.	DPID		PKTCNT													
	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																
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:** 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:** 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:** 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).

### 31.15.31 OTG Host channel-x DMA address register (OTG\_HCDMax) (x = 0..15, where x = Channel\_number)

Address offset: 0x514 + (Channel\_number × 0x20)

Reset value: 0x0000 0000

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

Bits 31:0 **DMAADDR:** 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.

*Note:* Configuration register applies only to USB OTG HS

### 31.15.32 Device-mode registers

These registers must be programmed every time the core changes to device mode

### 31.15.33 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.
ERRAT IM															
<b>rw</b>			<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>rw</b>	<b>res.</b>	<b>res.</b>	<b>pfivl</b>		DAD								<b>res.</b>	<b>nzlso hsk</b>	<b>dspd</b>
			<b>rw</b>	<b>rw</b>									<b>rw</b>	<b>rw</b>	<b>rw</b>

*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		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
						<b>rw</b>	<b>rw</b>								
ERRAT IM															
<b>rw</b>	<b>rw</b>		<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>	<b>rw</b>
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>rw</b>	<b>xcvr dly</b>	<b>res.</b>	<b>pfivl</b>		DAD								<b>res.</b>	<b>nzlso hsk</b>	<b>dspd</b>
			<b>rw</b>	<b>rw</b>									<b>rw</b>	<b>rw</b>	<b>rw</b>

*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:** 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

Bits 12:11 **PFIVL:** 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:** 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:** 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:** 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 external ULPI PHY

10: Reserved

11: Full speed using internal embedded PHY

**31.15.34 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 PRG DNE	CGO NAK	SGO NAK	CGI NAK	SGI NAK	TCTL				GON STS	GIN STS	SDIS	RWU SIG
				rw	w	w	w	w	rw	rw	rw	r	r	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

A write to this field clears the Global OUT NAK.

Bit 9 **SGONAK:** Set global OUT NAK

A write 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

A write to this field clears the Global IN NAK.

Bit 7 **SGINAK:** Set global IN NAK

A write 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:** 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  $\mu$ 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 223** 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 223. Minimum duration for soft disconnect**

Operating speed	Device state	Minimum duration
Full speed	Suspended	1 ms + 2.5 $\mu$ s
Full speed	Idle	2.5 $\mu$ s
Full speed	Not Idle or Suspended (Performing transactions)	2.5 $\mu$ s
High speed	Not Idle or Suspended (Performing transactions)	125 $\mu$ s

### 31.15.35 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	FNSOF						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FNSOF								Res.	Res.	Res.	Res.	EERR	ENUMSPD	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:** 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:** 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:** 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).

### 31.15.36 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.									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	NAKM	Res.	Res.	Res.	Res.	Res.	Res.	INEPNE MM	INEPNE MM	ITTXFE MSK	TOM	Res.	EPDM	XFR M
		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.	BMA	TXFURM	Res.	INEPNE MM	INEPNE MM	ITTXFE MSK	TOM	Res.	EPDM	XFR M
		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 12:7 Reserved, must be kept at reset value for USB OTG FS.

Bits 12:10 Reserved, must be kept at reset value for USB OTG HS.

Bit 9 **BIM:** BNA interrupt mask for USB OTG HS

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 8 **TXFURM:** FIFO underrun mask for USB OTG HS

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 7 Reserved, must be kept at reset value for USB OTG HS.

Bit 6 **INEPNE:** 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 Reserved, must be kept at reset value.

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

### 31.15.37 OTG device OUT endpoint common interrupt mask register (OTG\_DOEPMSK)

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.											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	OTEPD M	STUPM	Res.	EPDM	XFRCM										
											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	Res.	Res.	Res.	Res.	BOIM	TXFU RM	Res.	B2B STUP	Res.	OTEPD M	STUPM	Res.	EPDM	XFRCM
	rw					rw	rw		rw		rw	rw		rw	rw

*Note:* Configuration register for USB OTG HS

Bits 31:5 Reserved, must be kept at reset value for USB OTG FS.

Bits 31:15 Reserved, must be kept at reset value for USB OTG HS.

Bit 14 **NYET:** NYET interrupt mask for USB OTG HS

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 9 **BOIM:** BNA interrupt mask for USB OTG HS

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 8 **TXFURM:** FIFO underrun mask for USB OTG HS

- 0: Masked interrupt
- 1: Unmasked interrupt

Bit 7 Reserved, must be kept at reset value for USB OTG HS.

Bit 6 **B2BSTUP:** Back-to-back SETUP packets received mask. Applies to control OUT endpoints only. This is for USB OTG HS.

- 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 Reserved, must be kept at reset value.

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

### 31.15.38 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															
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															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 **OEPINT**: 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**: IN endpoint interrupt bits

One bit per IN endpoint:

Bit 0 for IN endpoint 0, bit 3 for endpoint 3.

### 31.15.39 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															
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															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **OEPM**: 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**: 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

### 31.15.40 OTG device V<sub>BUS</sub> discharge time register (OTG\_DVBUSDIS)

Address offset: 0x0828

Reset value: 0x0000 17D7

This register specifies the V<sub>BUS</sub> discharge time after V<sub>BUS</sub> 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.
VBUSDT															
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**: Device V<sub>BUS</sub> discharge time

Specifies the V<sub>BUS</sub> discharge time after V<sub>BUS</sub> pulsing during SRP. This value equals:  
V<sub>BUS</sub> discharge time in PHY clocks / 1 024  
Depending on your V<sub>BUS</sub> load, this value may need adjusting.

### 31.15.41 OTG device V<sub>BUS</sub> pulsing time register (OTG\_DVBUSPULSE)

Address offset: 0x082C

Reset value: 0x0000 05B8

This register specifies the V<sub>BUS</sub> 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.
DVBUSP															
rw	rw	rw	rw	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 **DVBUSP**: Device V<sub>BUS</sub> pulsing time

Specifies the V<sub>BUS</sub> pulsing time during SRP. This value equals:  
V<sub>BUS</sub> pulsing time in PHY clocks / 1 024

### 31.15.42 OTG Device threshold control register (OTG\_DTHRCTL)

Address offset: 0x0830

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16						
Res.	Res.	Res.	Res.	ARPEN	Res.	RXTHRLEN															RXTHREN
				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														ISOTHREN	NONISOTHREN	
RW	RW	RW	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.

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**: Receive threshold length

This field specifies the receive thresholding size in DWORDS. 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 DWORDS. 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**: Transmit threshold length

This field specifies the transmit thresholding size in DWORDS. 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 DWORDS. 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.

**Note:** Configuration register applies only to USB OTG HS

### 31.15.43 OTG device each endpoint interrupt register (OTG\_DEACHINT)

Address offset: 0x0838

Reset value: 0x0000 0000

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.

*Note:* Configuration register applies only to USB OTG HS

### 31.15.44 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.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INEPTXFEM															
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**: 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

### 31.15.45 OTG device each endpoint interrupt register mask (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.	IEP1I NTM	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

### 31.15.46 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				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	
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: 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**: 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

### 31.15.47 OTG device endpoint-x control register (OTG\_DIEPCTLx) (x = 1..5[FS] / 0..8[HS], where x = Endpoint\_number)

Address offset: 0x900 + (Endpoint\_number × 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				STALL	Res.	EPTYP		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										
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:** 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:** Maximum packet size

The application must program this field with the maximum packet size for the current logical endpoint. This value is in bytes.

### 31.15.48 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		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	
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:** 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:** 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

### 31.15.49 OTG device endpoint-x control register (OTG\_DOEPCTLx) ( $x = 1..5[\text{FS}] / 8[\text{HS}], \text{where } x = \text{Endpoint\_number}$ )

Address offset for OUT endpoints:  $0xB00 + (\text{Endpoint\_number} \times 0x20)$

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	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
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 **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:** 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**: Maximum packet size

The application must program this field with the maximum packet size for the current logical endpoint. This value is in bytes.

### 31.15.50 OTG device endpoint-x interrupt register (OTG\_DIEPINTx) (x = 0..5[FS] /8[HS], where x = Endpoint\_number)

Address offset: 0x908 + (Endpoint\_number × 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 408](#). 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.									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TXFE	INEP NE	Res.	ITTXFE	TOC	Res.	EP DISD	XFRC							
								r	r		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	BERR	PKTD RPSTS	Res.	BNA	TXFIF OUD RN	TXFE	INEP NE	Res.	ITTXFE	TOC	Res.	EP DISD	XFRC
		rc_w1	rc_w1	rc_w1		rc_w1	rc_w1	r	r		rc_w1	rc_w1		rc_w1	rc_w1

Note: Configuration register for USB OTG HS

Bits 31:8 Reserved, must be kept at reset value.

Bits 31:14 Reserved, must be kept at reset value for USB OTG HS

Bit 13 **NAK:** NAK input for USB OTG HS

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 for USB OTG HS

Bit 11 **PKTDRPSTS:** Packet dropped status for USB OTG HS

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 for USB OTG HS.

Bit 9 **BNA:** Buffer not available interrupt for USB OTG HS

The core generates this interrupt when the descriptor accessed is not ready for the Core to process, such as host busy or DMA done.

Bit 8 **TXFIFOUDRN:** Transmit Fifo Underrun (TxfifoUndrn) for USB OTG HS

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 Reserved, must be kept at reset value.

Bit 4 **ITTXFE:** IN token received when Tx FIFO is empty

Applies to non-periodic IN endpoints only.

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

Applies only to Control IN endpoints.

Indicates that the core has detected a timeout condition on the USB for the last IN token on this endpoint.

Bit 2 Reserved, must be kept at reset value.

Bit 1 **EPDISD:** Endpoint disabled interrupt

This bit indicates that the endpoint is disabled per the application's request.

Bit 0 **XFRD:** Transfer completed interrupt

This field indicates that the programmed transfer is complete on the AHB as well as on the USB, for this endpoint.

### 31.15.51 OTG device endpoint-x interrupt register (OTG\_DOEPINT $x$ ) ( $x = 0..5[FS] /8[HS]$ , where $x = \text{Endpoint\_number}$ )

Address offset: 0xB08 + (Endpoint\_number × 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 408](#). 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\_DOEPINT $x$  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.									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	B2B STUP	Res.	OTEP DIS	STUP	Res.	EP DISD	XFRC								
									rc_w1		rc_w1	rc_w1		rc_w1	rc_w1

Bits 31:7 Reserved, must be kept at reset value.

Bit 6 **B2BSTUP**: Back-to-back SETUP packets received

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 Reserved, must be kept at reset value.

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 Reserved, must be kept at reset value.

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.

### 31.15.52 OTG device IN endpoint 0 transfer size register (OTG\_DIEPTSIZ0)

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	Res.	Res.	Res.											
											rw	rw			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	XFRSIZ														
											rw	rw	rw	rw	rw

Bits 31:21 Reserved, must be kept at reset value.

Bits 20:19 **PKTCNT:** 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:** 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.

### 31.15.53 OTG Device channel-x DMA address register (OTG\_DIEPDMAx) (x = 0..15, where x= Channel\_number)

Address offset: 0x914 + (Channel\_number × 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															
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															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **DMAADDR:** 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.

### 31.15.54 OTG Device channel-x DMA address register (OTG\_DOEPDMAx) (x = 0..15, where x= Channel\_number)

Address offset: 0xB14 + (Channel\_number × 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															
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															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **DMAADDR:** 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.

### 31.15.55 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		Res.	Res.	Res.	PKTNT	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						
									rw	rw	rw	rw	rw	rw	rw

Bit 31 Reserved, must be kept at reset value.

Bits 30:29 **STUPCNT:** 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:** 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.

### 31.15.56 OTG device IN endpoint-x transfer size register (OTG\_DIEPTSI $_Zx$ ) ( $x = 1..5[\text{FS}] / 8[\text{HS}]$ , where $x = \text{Endpoint\_number}$ )

Address offset: 0x910 + (Endpoint\_number × 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		PKTCNT												XFRSIZ
	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															
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:** 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:** 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:** 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.

### 31.15.57 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:  $0x918 + (\text{Endpoint\_number} \times 0x20)$  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.
INEPTFSAV															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

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

15:0 **INEPTFSAV:** 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

### 31.15.58 OTG device OUT endpoint-x transfer size register (OTG\_DOEPTSI<sub>x</sub>) ( $x = 1..5[\text{FS}] / 8[\text{HS}]$ , where $x = \text{Endpoint\_number}$ )

Address offset: 0xB10 + (Endpoint\_number × 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\_DOEPCTL<sub>x</sub> registers (EPENA bit in OTG\_DOEPCTL<sub>x</sub>), 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		PKTCNT										XFRSIZ		
	r/rw	r/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															
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:** 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:** 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:** 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.

### 31.15.59 OTG power and clock gating control register (OTG\_PCGCCTL)

Address offset: 0xE00

Reset value: 0x0x200B 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	r/w	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\_I1\_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.

### 31.15.60 OTG\_FS/OTG\_HS register map

The table below gives the USB OTG register map and reset values.

**Table 224. OTG\_FS/OTG\_HS register map and reset values**

Offset	Register	Reset value	31
0x000	<b>OTG_GOTGCTL</b>		Res.
0x004	<b>OTG_GOTGINTR</b>		Res.
0x008	<b>OTG_GAHBCFG</b>		Res.
0x008	<b>OTG_GAHBCFG</b>		Res.
0x00C	<b>OTG_GUSBCFG</b>		Res.
0x00C	<b>OTG_GUSBCFG</b>		Res.
0x010	<b>OTG_GRSTCTL</b>		Res.
0x010	<b>OTG_GRSTCTL</b>		Res.

Table 224. OTG\_FS/OTG\_HS register map and reset values (continued)

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	
0x014	OTG_GINTSTS	WKUINT	WKUINT	SRQINT	DISCINT	CIDSCHG	LPMINT	PTXFE	HCINT	Hprtint	RSTDET	Res.	IPXFRI/INCOMPISOOUT	IISOIXFR	OEPINT	IEPINT	Res.	Res.	EOPF	ISOODRP	ENUMDNE	USBRST	USBJSUP	ESUSP	Res.	Res.	NPTXFE	RXFLVL	SOF	OTGIN	MMIS	CMOD	0		
		0	0	0	0	1	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			
0x014	OTG_GINTSTS	Reset value	0	0	0	0	0	PTXFEM	HCINT	Hprtint	RSTDET	Res.	DATAFSUSP	IPXFRI/INCOMPISOOUT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x018	OTG_GINTMSK	WUIM	WUIM	SRQIM	DISCINT	CIDSCHGM	LPMINTM	PTXFEM	HCINT	PRTIM	RSTDET	Res.	IPXFRI/INCOMPISOOUT	IISOIXFR	OEPINT	IEPINT	Res.	Res.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x018	OTG_GINTMSK	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		
0x01C	OTG_GRXSTSR (host 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.	Res.		
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x020	OTG_GRXSTSR (Device mode)	FRMNUM	FRMNUM	PKTSTS	DPID	BCNT	CHNUM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x020	OTG_GRXSTSPR (host 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.	Res.		
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x020	OTG_GRXSTSPR (Device mode)	FRMNUM	FRMNUM	PKTSTS	DPID	BCNT	CHNUM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		0	0	0	0	0	0	0	0	0	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 224. OTG\_FS/OTG\_HS 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	
0x024	OTG_GRXFSIZ	Res	Res	Res	Res	Res	Res	Res	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																																	
0x028	OTG_HNPTXFSIZ/ OTG_DIEPTXF0	NPTXFD/TX0FD												NPTXFSA/TX0FSA																				
	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	0	0	0	0	0	0	0	0		
0x02C	OTG_HNPTXSTS	Res	NPTXQTOP						NPTQXSADV						NPTXFSAV																			
	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		
0x030	OTG_GI2CCTL	BSYDNE	RW	Res	I2CDATSE	I2CDEVADR	Res	ACK	12CEN	ADDR						REGADDR						RWDATA												
	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		
0x038	OTG_GCCFG	Res	Res	Res	Res	Res	Res	Res	Res	VBDEN	Res	Res	Res	Res	PWRDWN	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res		
	Reset value																																	
0x03C	OTG_CID	PRODUCT_ID																																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x054	OTG_GLPMCFG	Res	Res	Res	ENBESL	LPMR CNTSTS	SNDLPM	LPM RCNT	LPMCHIDX	L1RSMOK	SLPSTS	LPM RSP	L1DSEN	BESLTHRS	LISSEN REMMAKE	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		
0x100	OTG_HPTXFSIZ	PTXFSIZ												PTXSA																				
	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	0	0	0	0	0	0	0	0		
0x104	OTG_DIEPTXF1	INEPTXFD												INEPTXSA																				
	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	0	0	0	0	0	0	0	0		
0x108	OTG_DIEPTXF2	INEPTXFD												INEPTXSA																				
	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	0	0	0	0	0	0	0	0		
.	.	.																																
0x204	OTG_DIEPTXF5	INEPTXFD												INEPTXSA																				
	Reset value	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0		
.	.	.																																

Table 224. OTG\_FS/OTG\_HS 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	
0x244	<b>OTG_DIEPTXF7</b>																																	
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x400	<b>OTG_HCFG</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
		Reset value																																
0x404	<b>OTG_HFIR</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
		Reset value																																
0x408	<b>OTG_HFNUM</b>																																	
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x410	<b>OTG_HPTXSTS</b>																																	
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0x414	<b>OTG_HAINT</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
		Reset value																																
0x418	<b>OTG_HAINTMASK</b>																																	
		Reset value																																
0x440	<b>OTG_HPRT</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
		Reset value																																
0x504	<b>OTG_HCSPLT0</b>																																	
		Reset value	0	CHENA	SPLITEN	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
0x500	<b>OTG_HCCHAR0</b>	0	CHDIS	DAD	EPDIR	MCNT	EPTYP	LSDEV	0	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	ODDFRM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x508	<b>OTG_HCINT0</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																																
0x510	<b>OTG_HCTSIZ0</b>	Res.	DPID	PKTCNT	DTERR	FRMOR	BBERR	TXERR	0	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

Table 224. OTG\_FS/OTG\_HS 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
0x50C	<b>OTG_HCINTMSK0</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	
		Reset value																															
0x514	<b>OTG_HCDMA0</b>	DMAADDR																															
		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	
0x520	<b>OTG_HCCHAR1</b>	DAD																															
		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	
0x528	<b>OTG_HCINT1</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	
		Reset value																															
0x52C	<b>OTG_HCINTMSK1</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	
		Reset value																															
0x530	<b>OTG_HCTSIZ1</b>	Res.	DPID	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	0	0	0	0
0x660	<b>OTG_HCCHAR11</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	
		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
0x66C	<b>OTG_HCINTMSK11</b>	Res.	DAD										MCNT	EPTYP	EPDIR	EPNUM										MPSIZ							
		Reset value																															
0x670	<b>OTG_HCTSIZ11</b>	Res.	DPID	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	0	0	0	0

**Table 224. OTG\_FS/OTG\_HS register map and reset values (continued)**

Table 224. OTG\_FS/OTG\_HS register map and reset values (continued)

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
0x7A8	OTG_HCINT15	Reset value	Res.																									
0x800	OTG_DCFG	Reset value	Res.																									
0x800	OTG_DCFG	Reset value	Res.																									
0x800	OTG_DCFG	Reset value	Res.																									
0x804	OTG_DCTL	Reset value	Res.																									
0x808	OTG_DSTS	Reset value	Res.																									
0x810	OTG_DIEPMSK	Reset value	Res.																									
0x814	OTG_DOEPMISK	Reset value	Res.																									
0x814	OTG_DOEPMISK	Reset value	Res.																									
0x818	OTG_DAINT	OEPINT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x818	OTG_DAINT	IEPINT	0	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	

Table 224. OTG\_FS/OTG\_HS 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		
0x81C	<b>OTG_DAINTMSK</b>																																		
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0x828	<b>OTG_DVBUSDIS</b>																																		
		Reset value																																	
0x82C	<b>OTG_DVB_USPULSE</b>																																		
		Reset value																																	
0x830	<b>OTG_DTHRCTL</b>																																		
		Reset value																																	
0x834	<b>OTG_DIE_PEMPPMSK</b>																																		
		Reset value																																	
0x838	<b>OTG_DEACHINT</b>																																		
		Reset value																																	
0x83C	<b>OTG_DEACHI_NTMSK</b>																																		
		Reset value																																	
0x900	<b>OTG_DIEPCTL0</b>																																		
		Reset value	0	EPENA	0	EPDIS	0	SODDERM/SD1/PID	0	SDDPID/SEVNFRM	0	SNAK	0	CNAK	0	TXFNUM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x900	<b>OTG_DIEPCTL0</b>																																		
		Reset value	0	EPENA	0	EPDIS	0	SODDERM/SD1/PID	0	SDDPID/SEVNFRM	0	SNAK	0	CNAK	0	TXFNUM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x908	<b>OTG_DIEPINT0</b>																																		
		Reset value																																	
0x910	<b>OTG_DIEPTSIZ0</b>																																		
		Reset value																																	
0x914	<b>OTG_DIEPDMA</b>																																		
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

Table 224. OTG\_FS/OTG\_HS 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
0x918	OTG_DTXFSTS0	Res.	Res.	EPDIS	SODDFRM	SD0PID/SEVNFRM	SD0PID/SEVNFRM	SNAK	CNAK	TXFNUM	STALL	Res.	EONUMDPID	USBAEP	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		
0x920	OTG_DIEPCTL1	EPENA	Res.	Res.	MCNT	Res.	Res.	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	
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.	Res.	Res.		
	Reset value																																
0x930	OTG_DIEPTSIZ1	Res.	Res.	Res.	Res.	Res.	Res.	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																																
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.	Res.	Res.		
	Reset value																																
0x940	OTG_DIEPCTL2	EPENA	EPDIS	SODDFRM	SD0PID/SEVNFRM	SD0PID/SEVNFRM	SD0PID/SEVNFRM	SNAK	CNAK	TXFNUM	STALL	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	
0x9A0	OTG_DIEPCTL5	Res.	Res.	Res.	Res.	Res.	Res.	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	
0x9A8	OTG_DIEPINT5	Res.	Res.	Res.	Res.	Res.	Res.	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 224. OTG\_FS/OTG\_HS 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
0x9B8	<b>OTG_DTXFSTS5</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																															
0x9E0	<b>OTG_DIEPCTL7</b>	EPENA	EPDIS	SODDFRM	SD0PID/SEVNFRM	SNAK	CNAK	TXFNUM	STALL	EPTYP	EPNUM/DPID	USBAEP	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	0	0	
0x9B0	<b>OTG_DIEPTSIZ5</b>	MCNT	MCNT	PKTCNT	PKTCNT	NAKSTS	NAKSTS	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	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	0	0		
0x9E8	<b>OTG_DIEPINT7</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																															
0x9F0	<b>OTG_DIEPTSIZ7</b>	MCNT	MCNT	PKTCNT	PKTCNT	NAKSTS	NAKSTS	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	XFRSIZ	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	0	0		
0x9F8	<b>OTG_DTXFSTS7</b>	Res.	EPDIS	Res.	Res.	Res.	Res.	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	
0xB00	<b>OTG_DOEPCTL0</b>	EPENA	EPDIS	STALL	SNPM	EPTYP	USBAEP	NAKSTS	NAKSTS	NAKSTS	NAKSTS	NAKSTS	NAKSTS	NAKSTS	NAKSTS	NAKSTS	NAKSTS	NAKSTS	NAKSTS	NAKSTS	NAKSTS	NAKSTS	NAKSTS	NAKSTS	NAKSTS	NAKSTS	NAKSTS	NAKSTS	NAKSTS	NAKSTS	NAKSTS		
		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		
0xB08	<b>OTG_DOEPINT0</b>	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		
		Reset value																															

**Table 224. OTG\_FS/OTG\_HS register map and reset values (continued)**

Table 224. OTG\_FS/OTG\_HS 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
0xBA8	OTG_DOEPIINT5	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	XFRCS			
0xBB0	OTG_DOEPTSIZ5	Res.	RXPID/ STUPCNT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0xBE8	OTG_DOEPIINT7	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.			
0xBF0	OTG_DOEPTSIZ7	Res.	RXPID/ STUPCNT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0xE00	OTG_PCGCCTL	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	XFRCS			
	Reset value																																

Refer to [Section 2.2.2: Memory map and register boundary addresses](#) for the register boundary addresses.

## 31.16 OTG\_FS/OTG\_HS programming model

### 31.16.1 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.

### 31.16.2 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<sub>BUS</sub> 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.

### 31.16.3 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 1237](#).

At this point, the device is ready to accept SOF packets and perform control transfers on control endpoint 0.

### 31.16.4 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.

### 31.16.5 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 HS\_OTG 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 HS\_OTG 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 HS\_OTG host continues sending ping tokens until it receives an ACK, and then switches automatically to the data transaction.

### Operational model

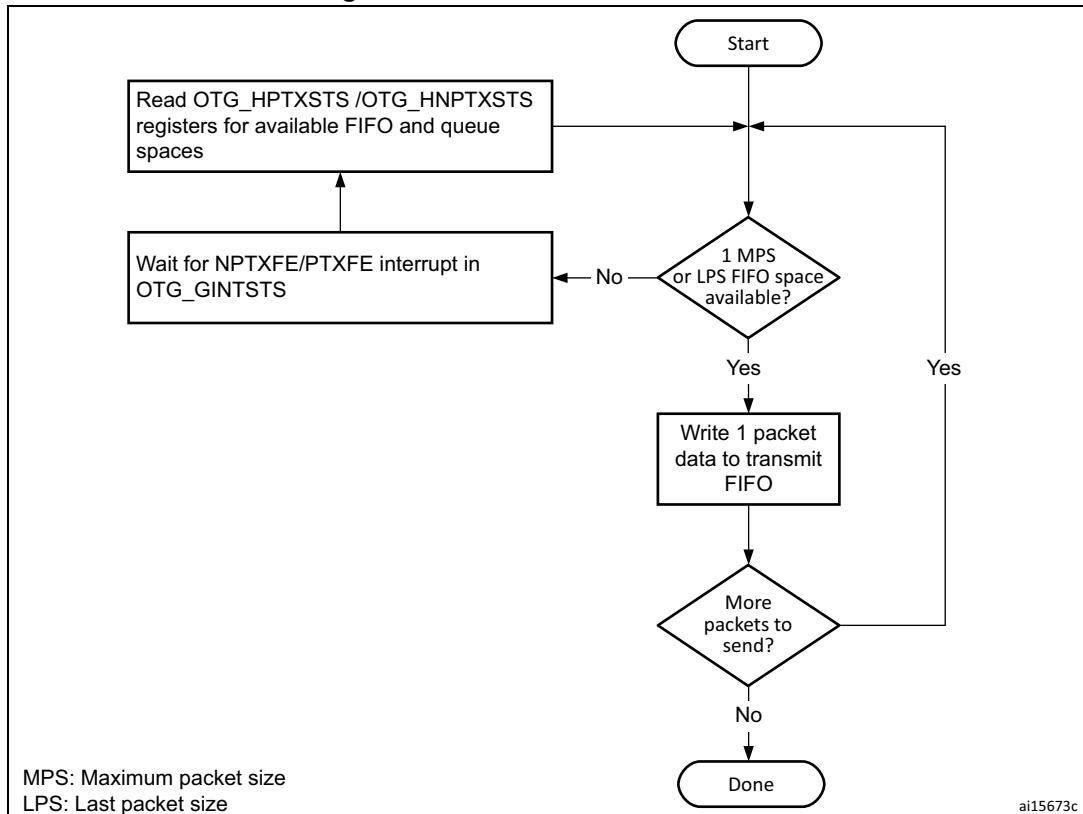
The application must initialize a channel before communicating to the connected device. This section explains the sequence of operation to be performed for different types of USB transactions.

- **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 DWORD 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 DWORDs. If the packet size is non-DWORD 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.

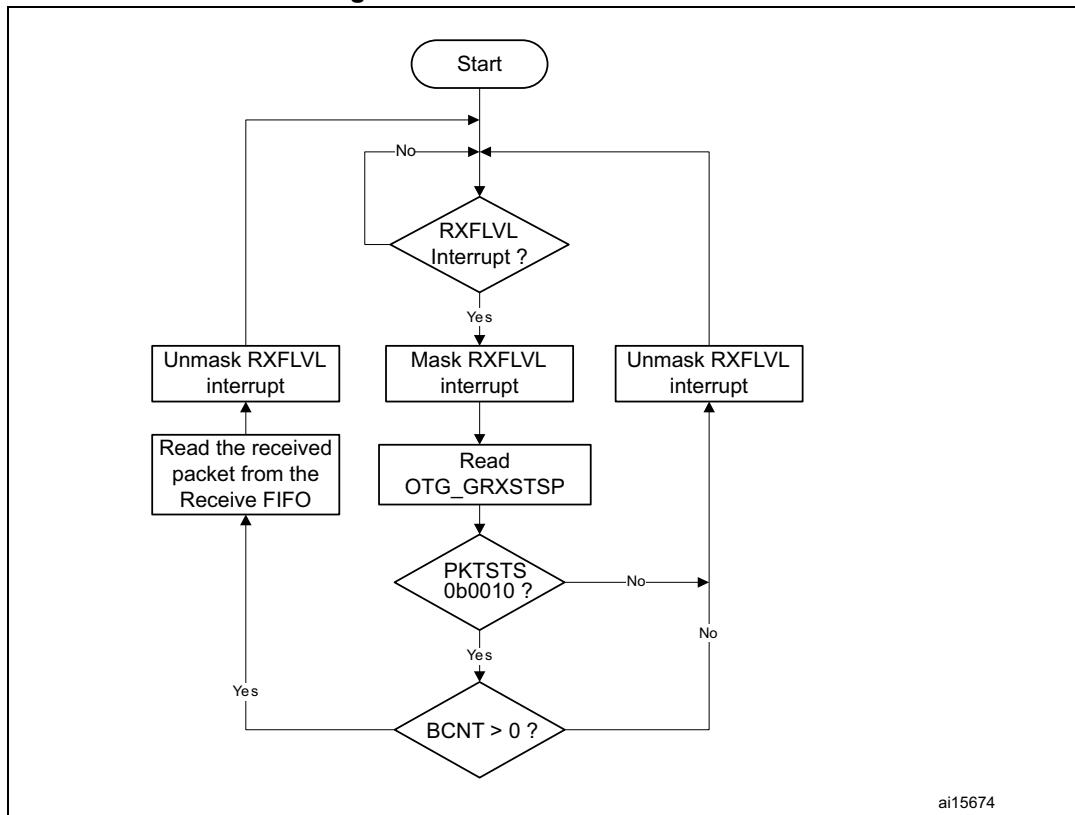
**Figure 409. Transmit FIFO write task**



- **Reading the receive FIFO**

The application must ignore all packet statuses other than IN data packet (bx0010).

Figure 410. 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 411](#). 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:

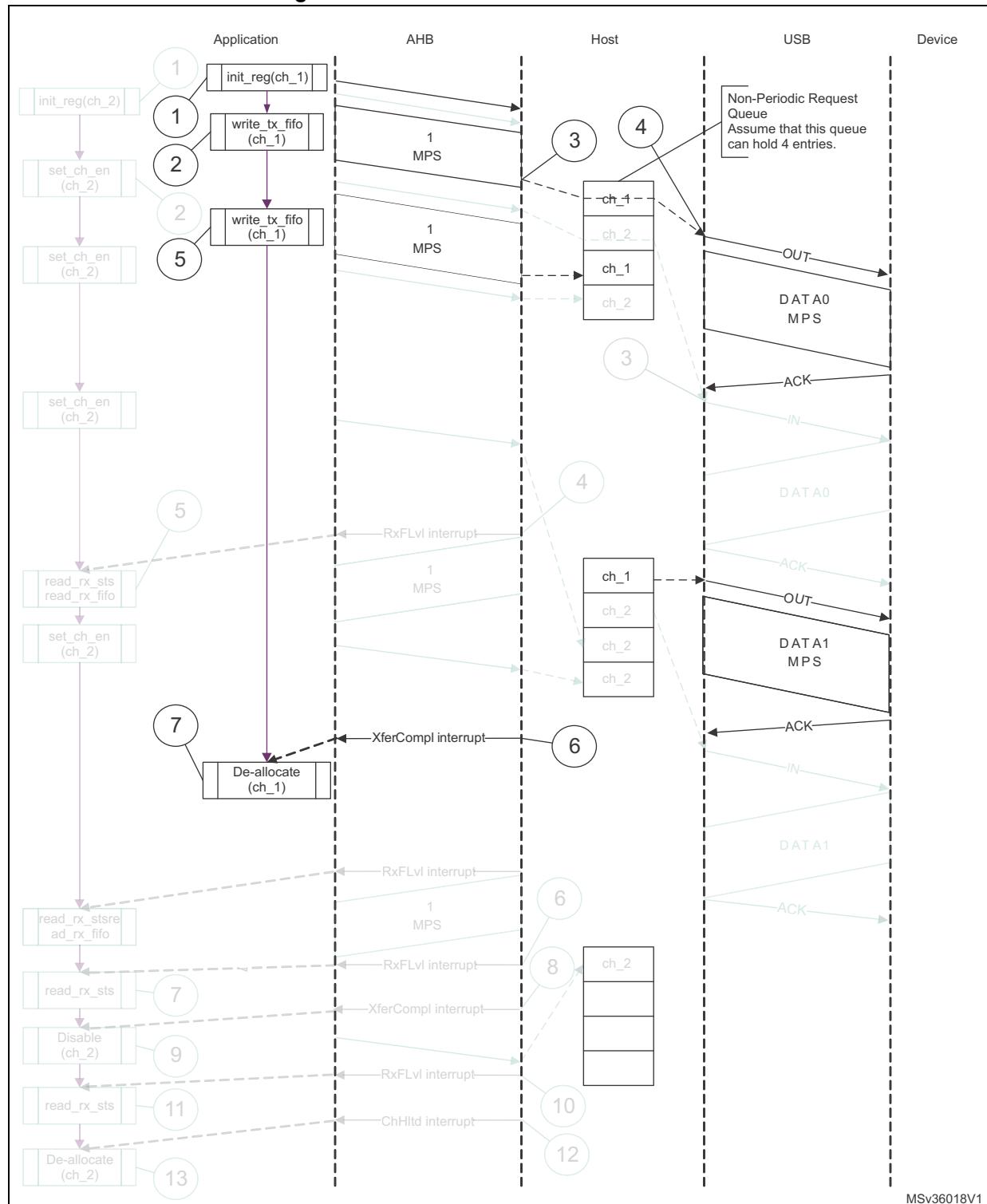
- The application is attempting to send two maximum-packet-size packets (transfer size = 1,024 bytes).
- The non-periodic transmit FIFO can hold two packets (1 KB for HS/128 bytes for FS).
- The non-periodic request queue depth = 4.

- **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 411. 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
}
```

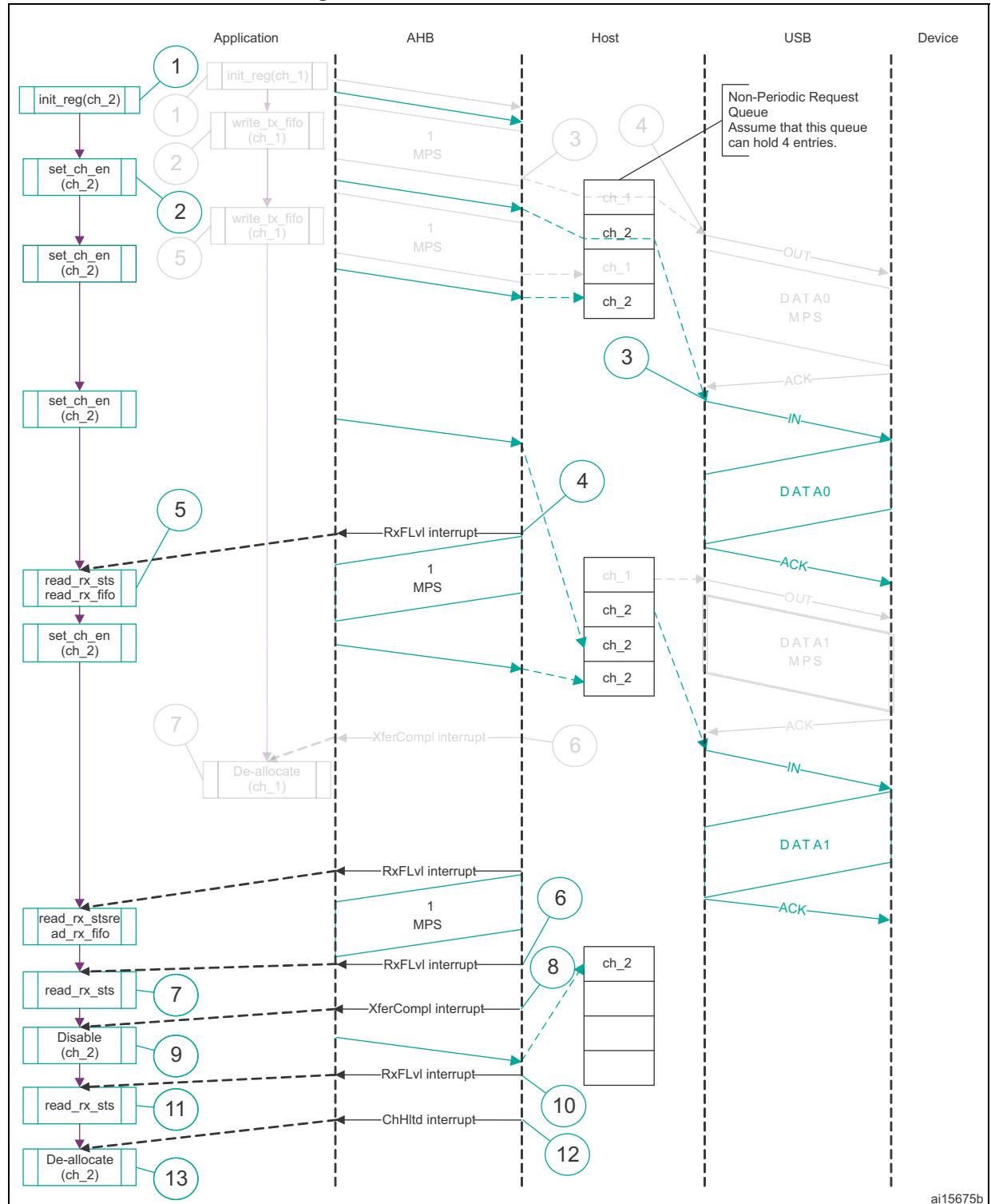
The application is expected to write the requests as and when the Request queue space is available and until the XFRC interrupt is received.

- **Bulk and control IN transactions**

A typical bulk or control IN pipelined transaction-level operation is shown in [Figure 412](#). 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 412. 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**

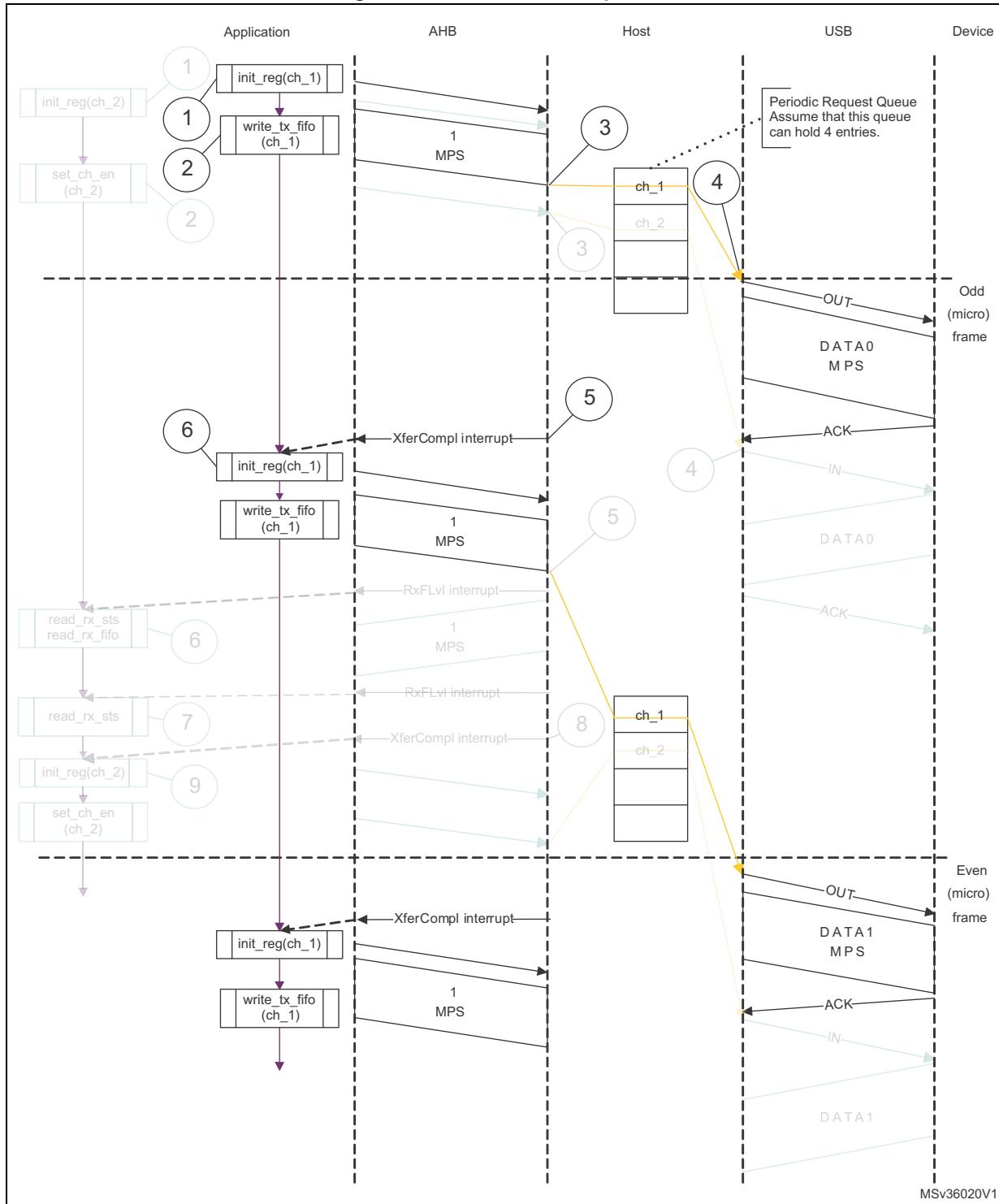
A typical interrupt OUT operation is shown in [Figure 413](#). The assumptions are:

- 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 413. Normal interrupt OUT



1. The grayed elements are not relevant in the context of this figure.

- **Interrupt service routine for interrupt OUT/IN transactions**

- a) 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:

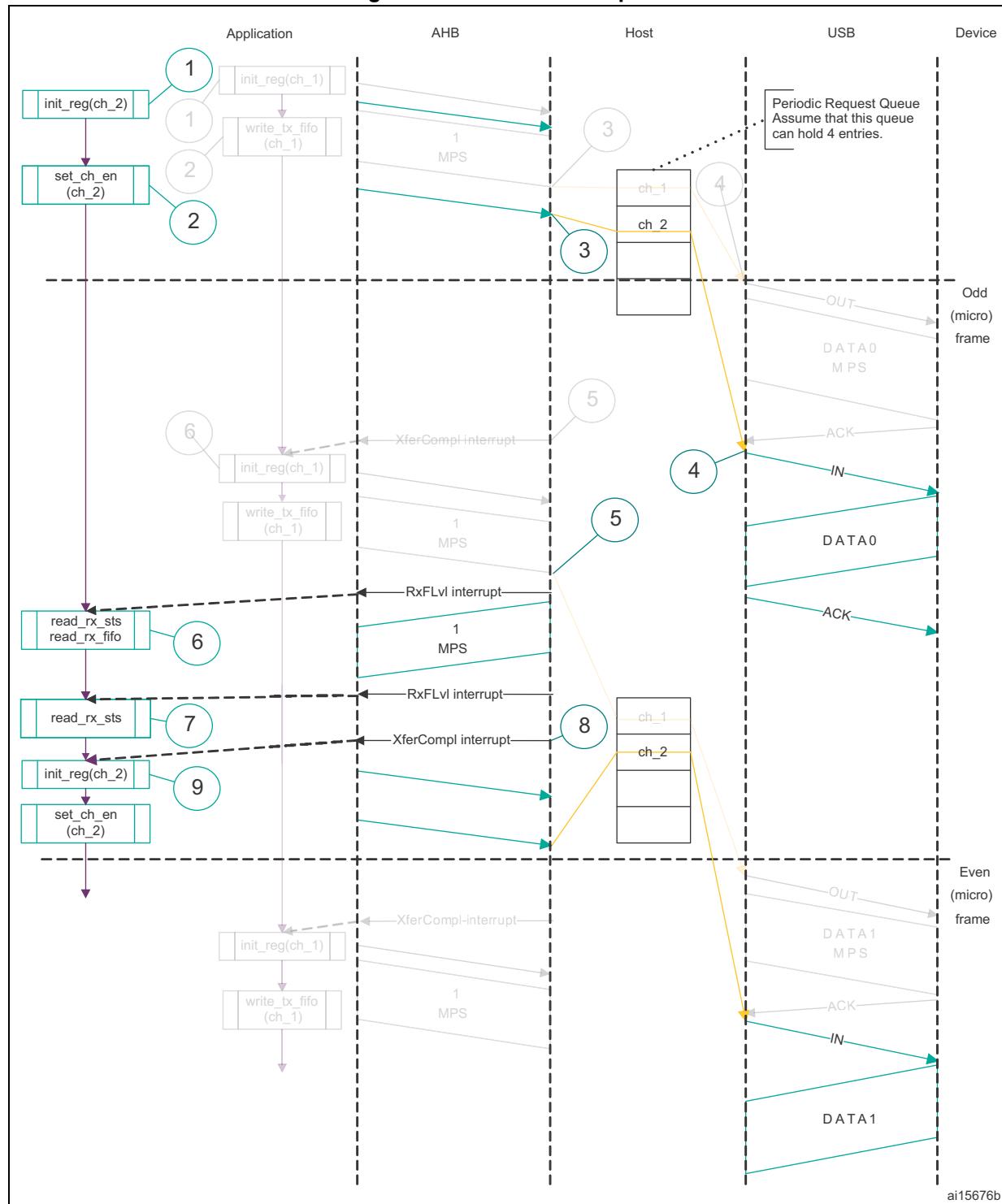
- The application is attempting to receive one packet (up to 1 maximum packet size) in every frame, starting with odd (transfer size = 1 024 bytes).
- The receive FIFO can hold at least one maximum-packet-size packet and two status Words per packet (1 031 bytes).
- Periodic request queue depth = 4.

- **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 GRXSTSR  $\neq$  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 414. Normal interrupt IN



1. The grayed elements are not relevant in the context of this figure.

- **Isochronous OUT transactions**

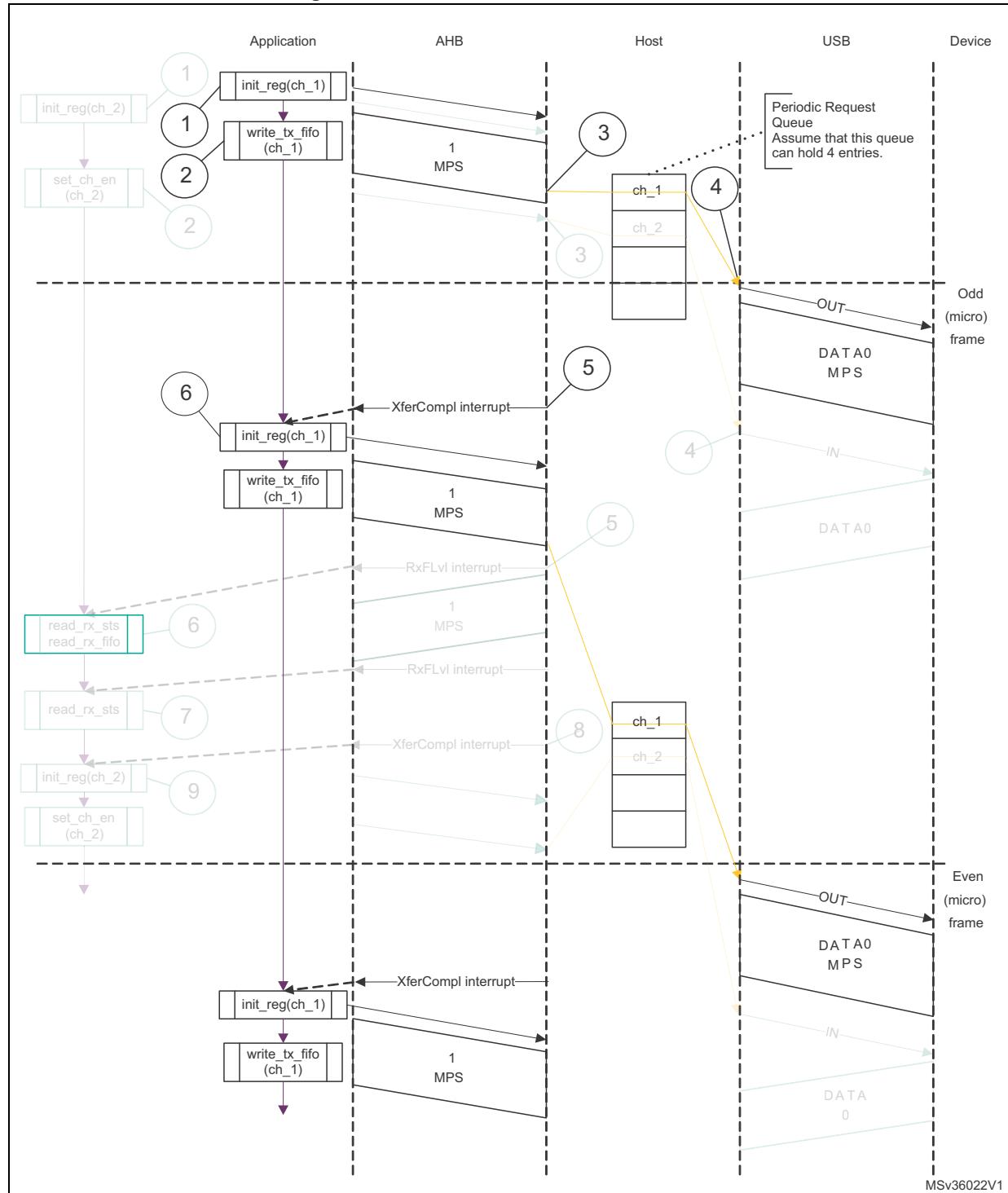
A typical isochronous OUT operation is shown in [Figure 414](#). The assumptions are:

- 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 415. 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 (FRMOR/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**

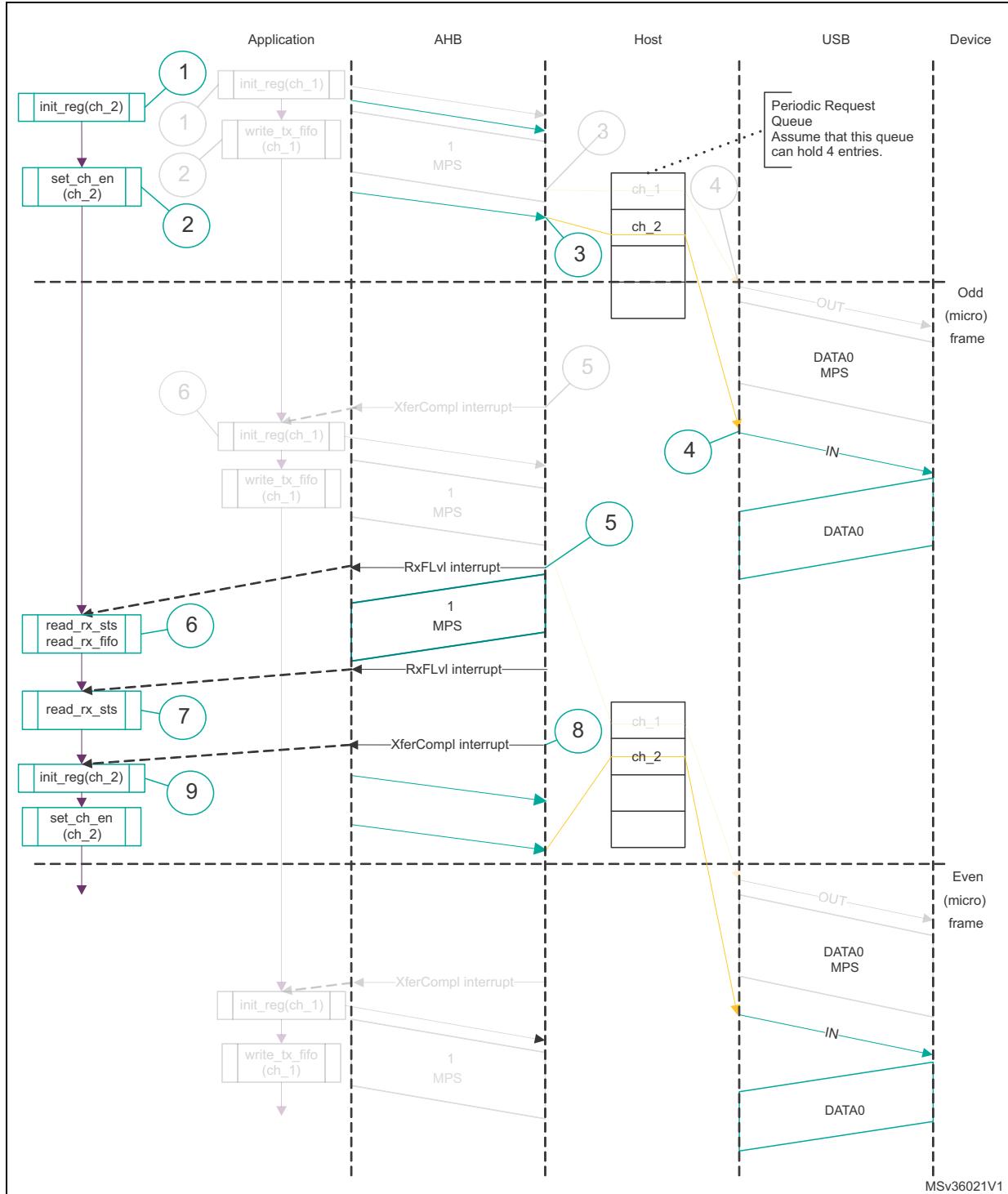
The assumptions are:

- 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 416. 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.

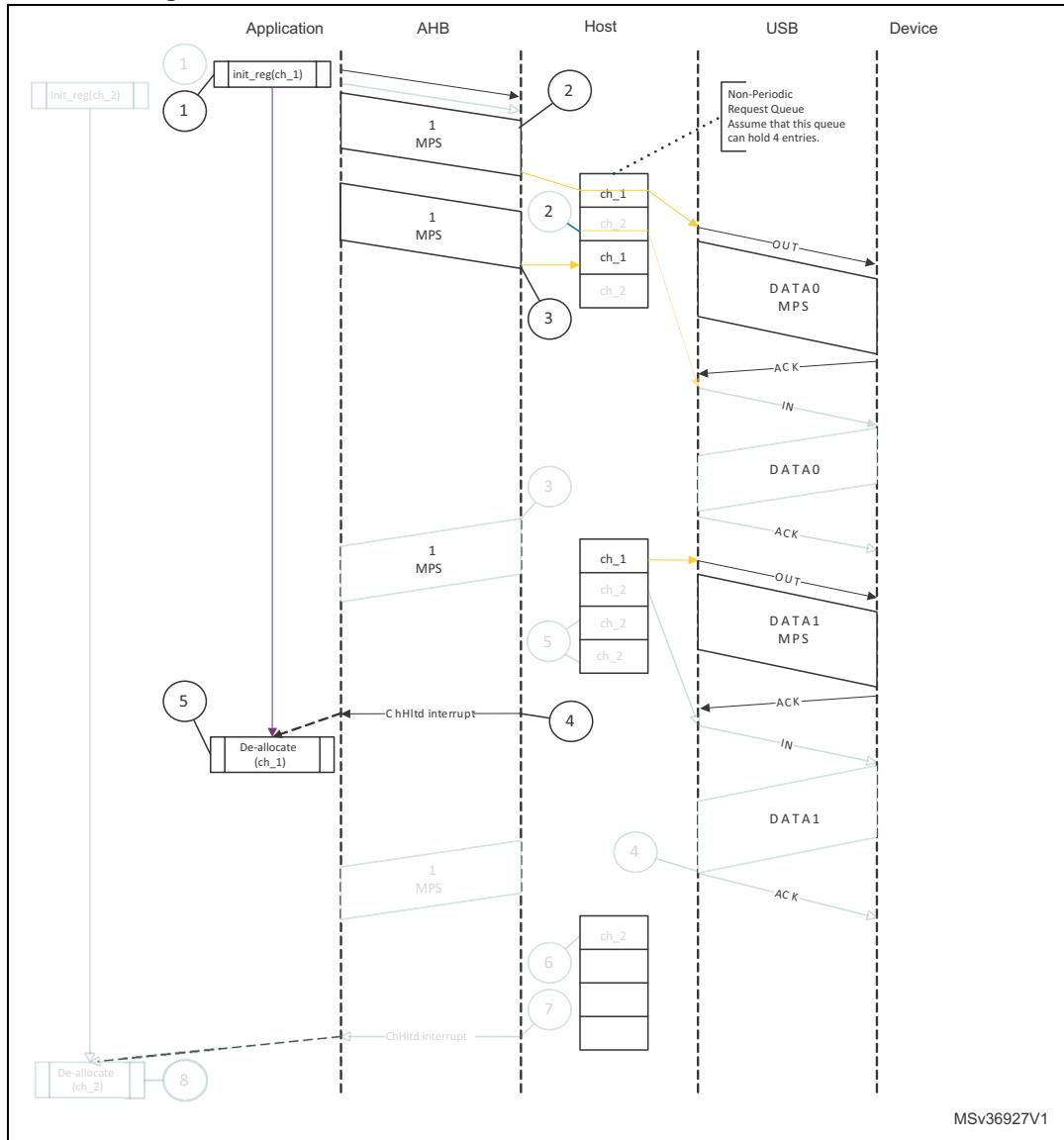
**Note:** The following sections covering DMA configurations apply only to USB OTG HS.

- **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 HS\_OTG 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 DWORD of the second (last) packet, the OTG\_HS host masks channel 1 internally for further arbitration.
4. The HS\_OTG 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 417. Normal bulk/control OUT/SETUP transactions - DMA



- **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.

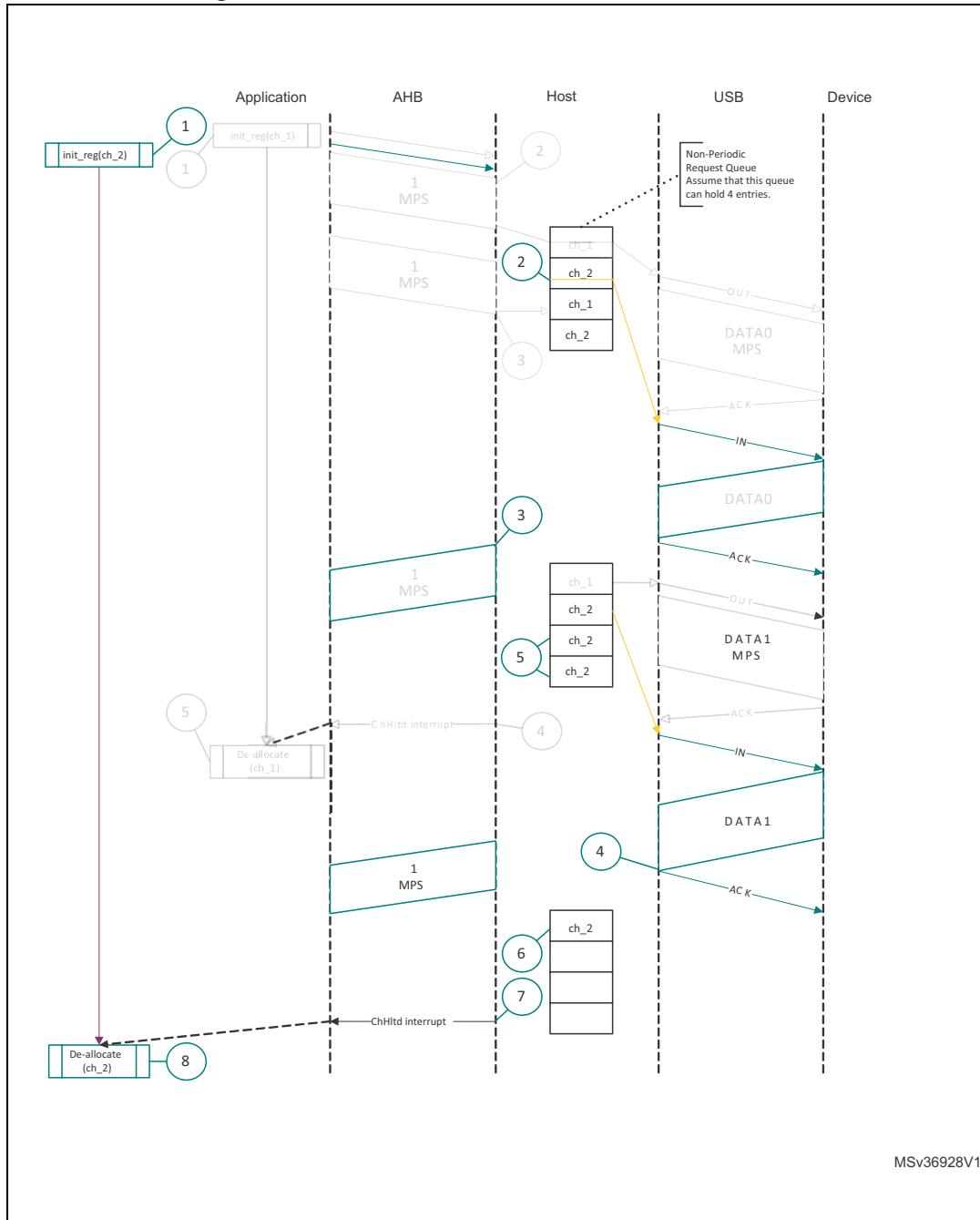
The core does not generate a separate interrupt when NAK or NYET is received by the host functionality.

- **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 418. Normal bulk/control IN transaction - DMA



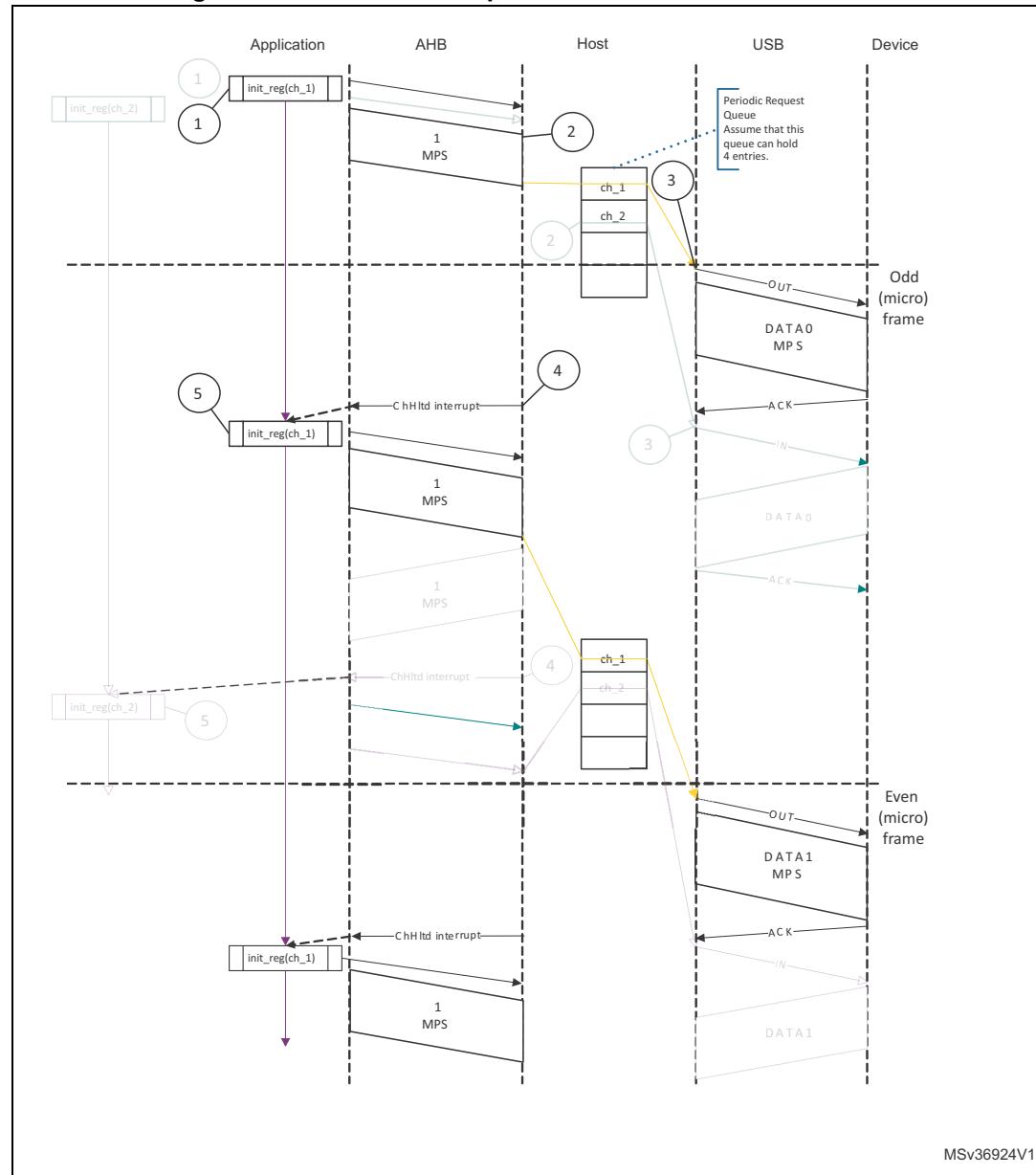
- **Interrupt OUT transactions in DMA mode**

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 DWORD fetch. In high-bandwidth transfers,

the HS\_OTG 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.

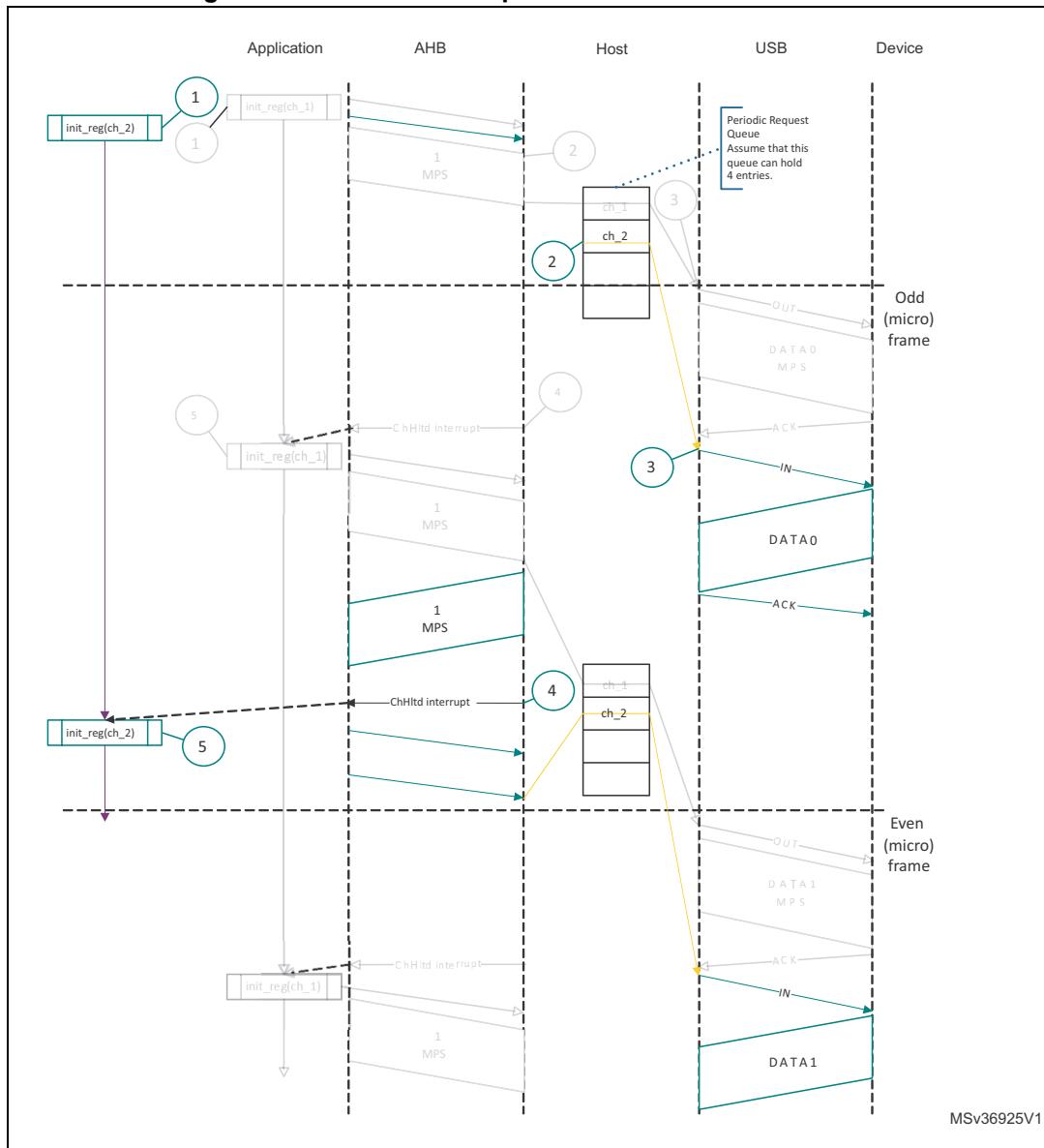
**Figure 419. Normal interrupt OUT transactions - DMA mode**



- **Interrupt IN transactions in DMA mode**

The sequence of operations (channelx) is as follows:

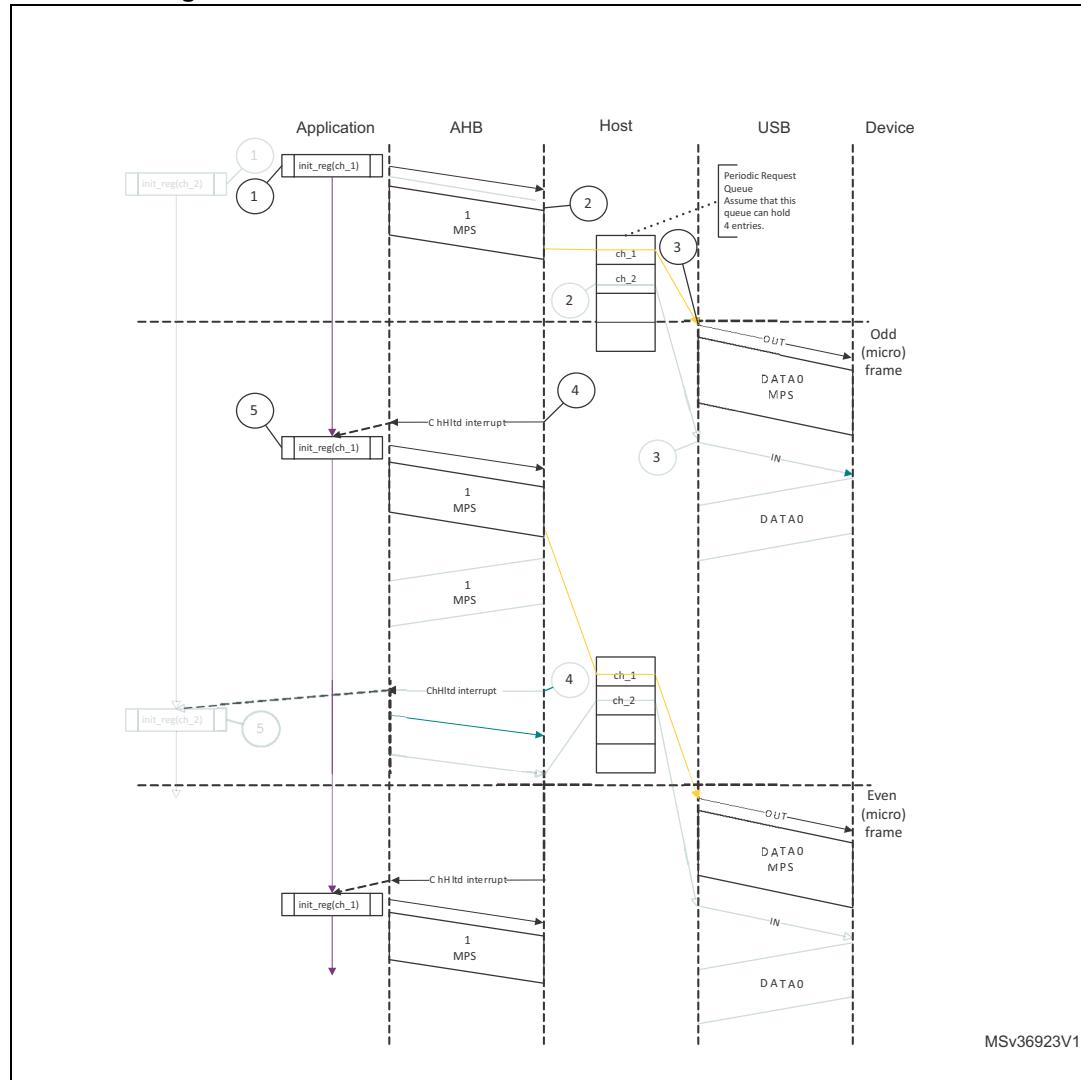
1. Initialize and enable channel x as explained in [Section : Channel initialization](#).
2. The OTG\_HS host writes an IN request to the request queue as soon as the channel x gets the grant from the arbiter (round-robin with fairness). In high-bandwidth transfers, the OTG\_HS host writes consecutive 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 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 420. 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 DWORD 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 HS\_OTG host generates a CHH interrupt.
5. In response to the CHH interrupt, reinitialize the channel for the next transfer.

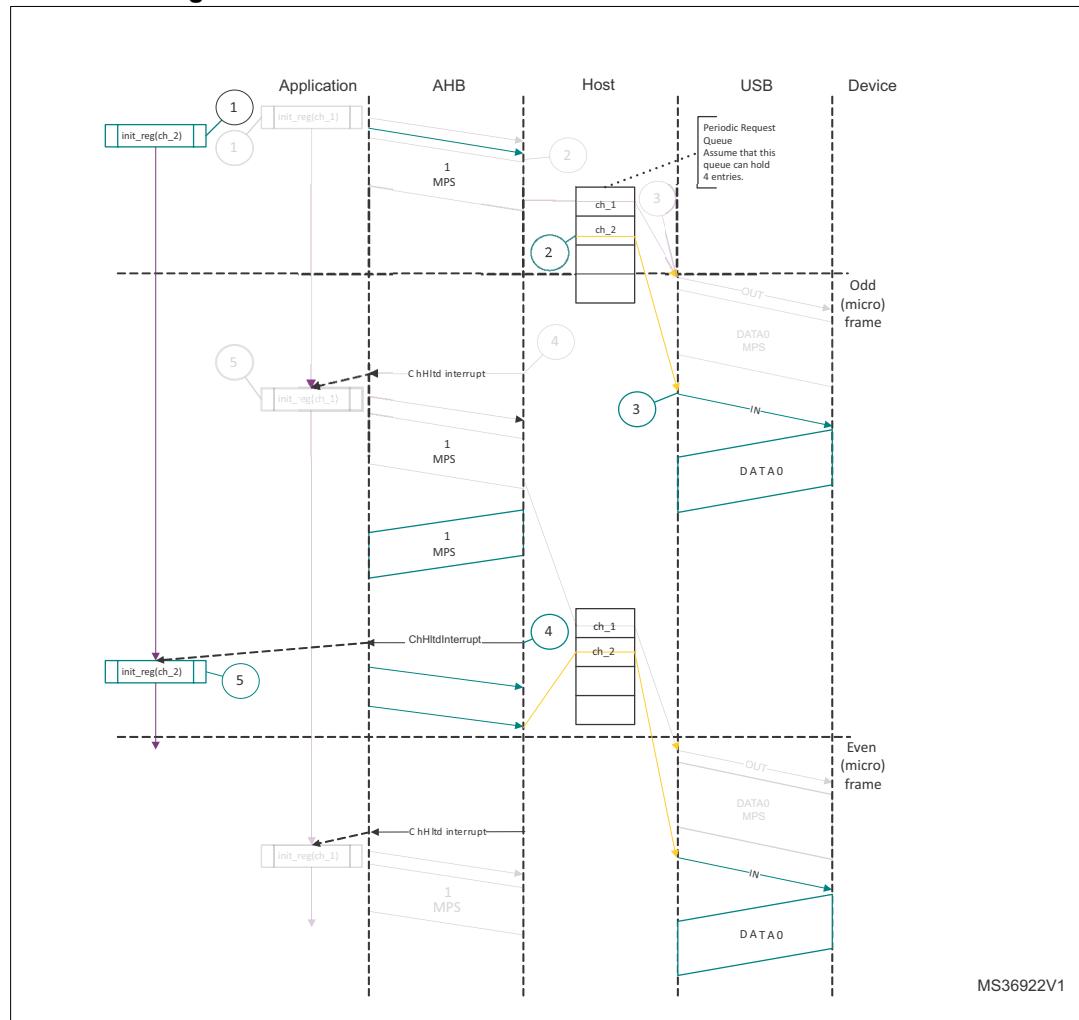
**Figure 421. Normal isochronous OUT transaction - DMA mode**



- **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 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 422. 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 DWORD 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 HS\_OTG host starts reading the packet.
3. The HS\_OTG 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 HS\_OTG host starts reading the packet.
    3. After successfully transmitting the start split (begin), the HS\_OTG 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.

*Note:* The content of this section applies only to USB OTG HS.

### 31.16.6 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.

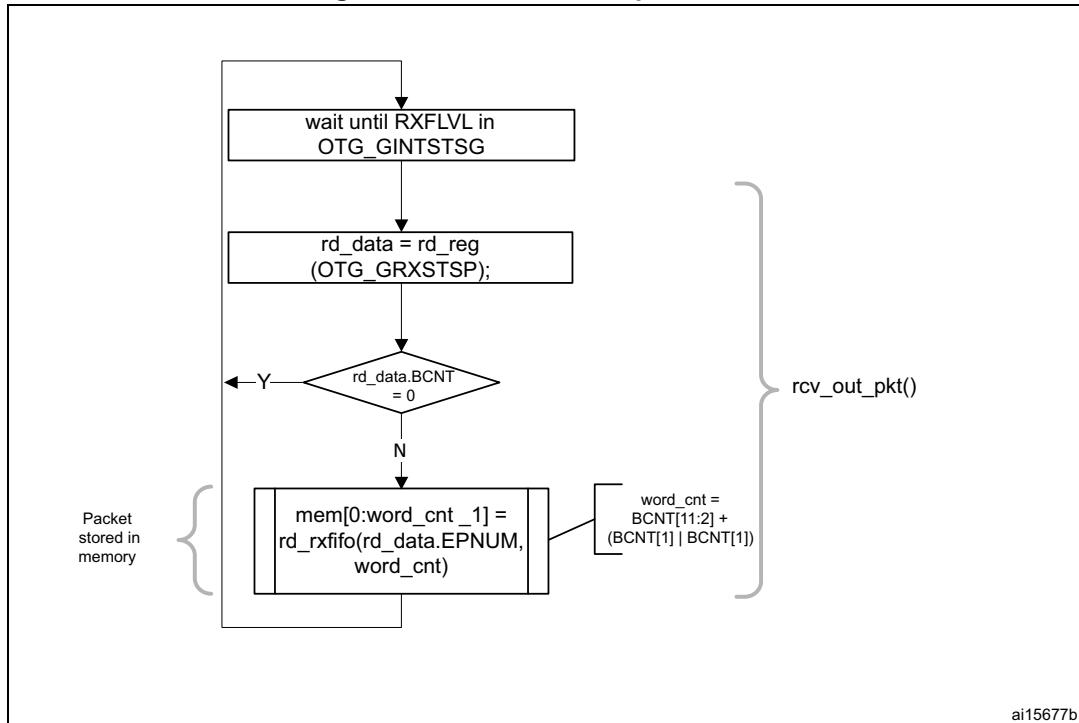
- **Packet read**

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 423* provides a flowchart of the above procedure.

**Figure 423. Receive FIFO packet read**



## **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.

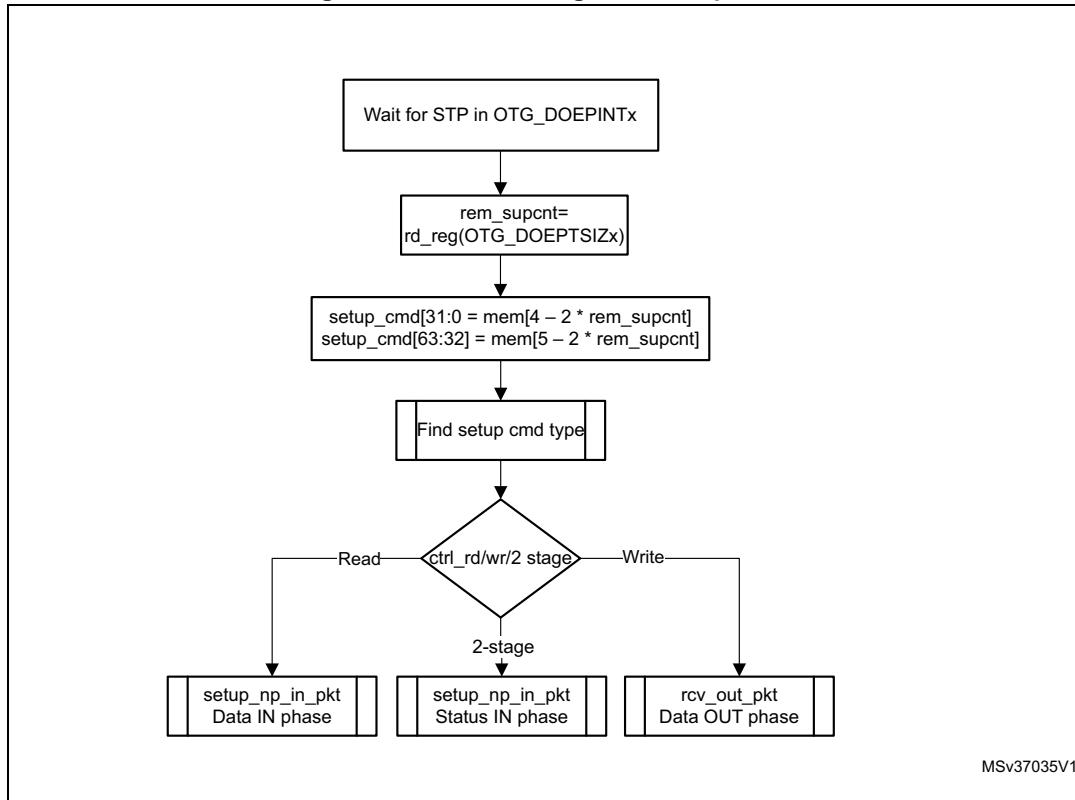
- **Internal data flow**

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 424. 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 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.

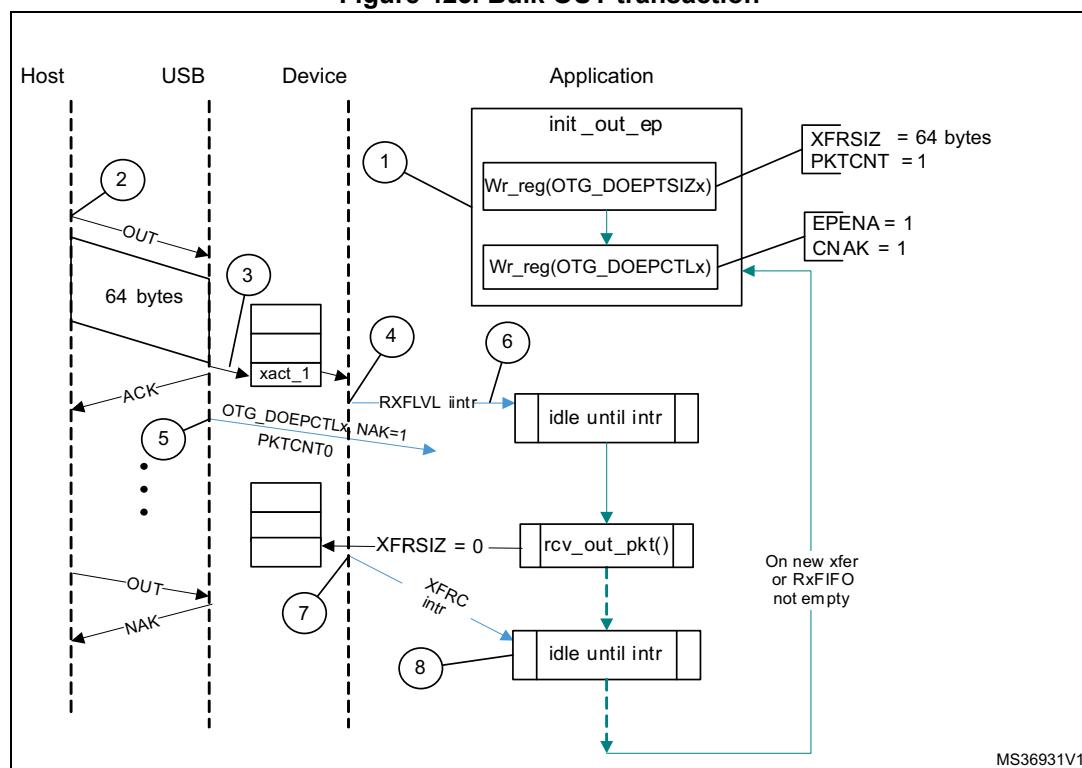
## Examples

This section describes and depicts some fundamental transfer types and scenarios.

- Bulk OUT transaction

*Figure 425* 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 425. 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.

1. host attempts to send data (OUT token) to an endpoint.
2. When the core receives the OUT token on the USB, it stores the packet in the Rx FIFO because space is available there.
3. After writing the complete packet in the Rx FIFO, the core then asserts the RXFLVL interrupt (in OTG\_GINTSTS).
4. 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.
5. The application processes the interrupt and reads the data from the Rx FIFO.
6. When the application has read all the data (equivalent to XFRSIZ), the core generates an XFRC interrupt (in OTG\_DOEPINTx).
7. 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:
 
$$\text{Transfer size[EPNUM]} = x \times \text{MPSIZ[EPNUM]} + \text{sp}$$
 If ( $\text{sp} > 0$ ), then  $\text{packet count[EPNUM]} = x + 1$ .  
 Otherwise,  $\text{packet count[EPNUM]} = x$
  - To transmit a single zero-length data packet:
 
$$\text{Transfer size[EPNUM]} = 0$$

$$\text{Packet count[EPNUM]} = 1$$
  - 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:  $\text{transfer size[EPNUM]} = x \times \text{MPSIZ[epnum]}$ ;  $\text{packet count} = n$ ;
   
Second transfer:  $\text{transfer size[EPNUM]} = 0$ ;  $\text{packet count} = 1$ ;
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)  $\times \text{MPSIZ[EPNUM]}$
  - 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\_DIEPTSIZx register with the transfer size and corresponding packet count.
2. Program the OTG\_DIEPCTLx 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\_DTXFSTSx 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\_DIEPINTx) 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 1252](#) 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:

transfer size[EPNUM] =  $x \times \text{MPSIZ}[EPNUM] + sp$   
 (where  $x$  is an integer  $\geq 0$ , and  $0 \leq sp < \text{MPSIZ}[EPNUM]$ )

If ( $sp > 0$ ), packet count[EPNUM] =  $x + 1$

Otherwise, packet count[EPNUM] =  $x$ ;

MCNT[EPNUM] = 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:
  - transfer size[EPNUM] = 0
  - packet count[EPNUM] = 1
  - MCNT[EPNUM] = packet count[EPNUM]
- 2. The application can only schedule data transfers one frame at a time.
  - $(MCNT - 1) \times \text{MPSIZ} \leq \text{XFERSIZ} \leq MCNT \times \text{MPSIZ}$
  - PKTCNT = MCNT (in OTG\_DIEPTSIZx)
  - If  $\text{XFERSIZ} < MCNT \times \text{MPSIZ}$ , the last data packet of the transfer is a short packet.
  - Note that: MCNT is in OTG\_DIEPTSIZx, MPSIZ is in OTG\_DIEPCTLx, PKTCNT is in OTG\_DIEPTSIZx and XFERSIZ is in OTG\_DIEPTSIZx
- 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\_DIEPTSIz 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\_DIEPTSIz 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.

### 31.16.7 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).

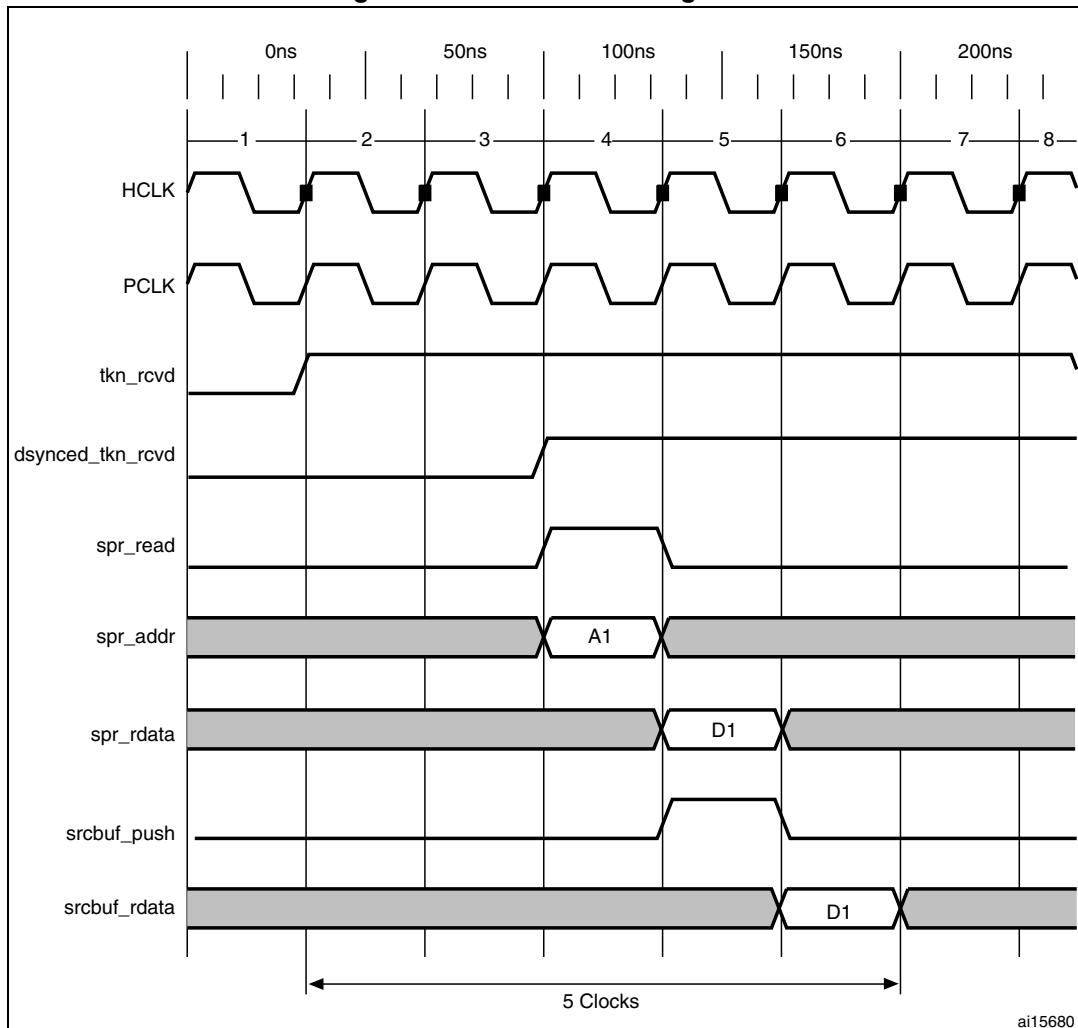
If the AHB is running at a higher frequency than the PHY, the application can use a smaller value for TRDT (in OTG\_GUSBCFG).

*Figure 426* 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 221: TRDT values \(FS\)](#) or [Table 222: TRDT values \(HS\)](#).

Figure 426. TRDT max timing case

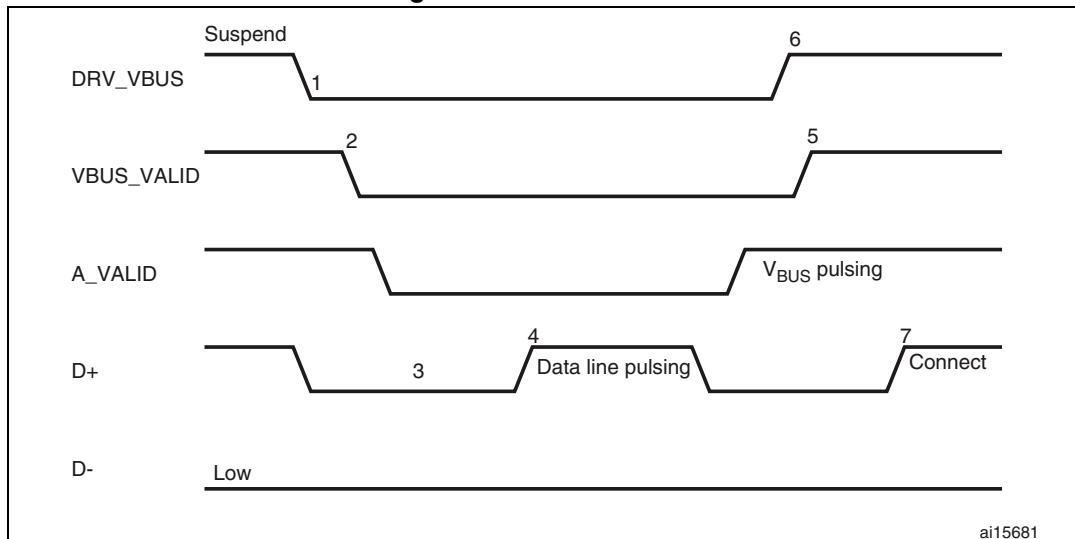


### 31.16.8 OTG programming model

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 427. A-device SRP**

1.  $\text{DRV\_VBUS} = V_{\text{BUS}}$  drive signal to the PHY  
 $\text{VBU}_S\text{\_VALID} = V_{\text{BUS}}$  valid signal from PHY  
 $\text{A\_VALID} = \text{A-peripheral } V_{\text{BUS}}$  level signal to PHY  
 $D^+ = \text{Data plus line}$   
 $D^- = \text{Data minus line}$

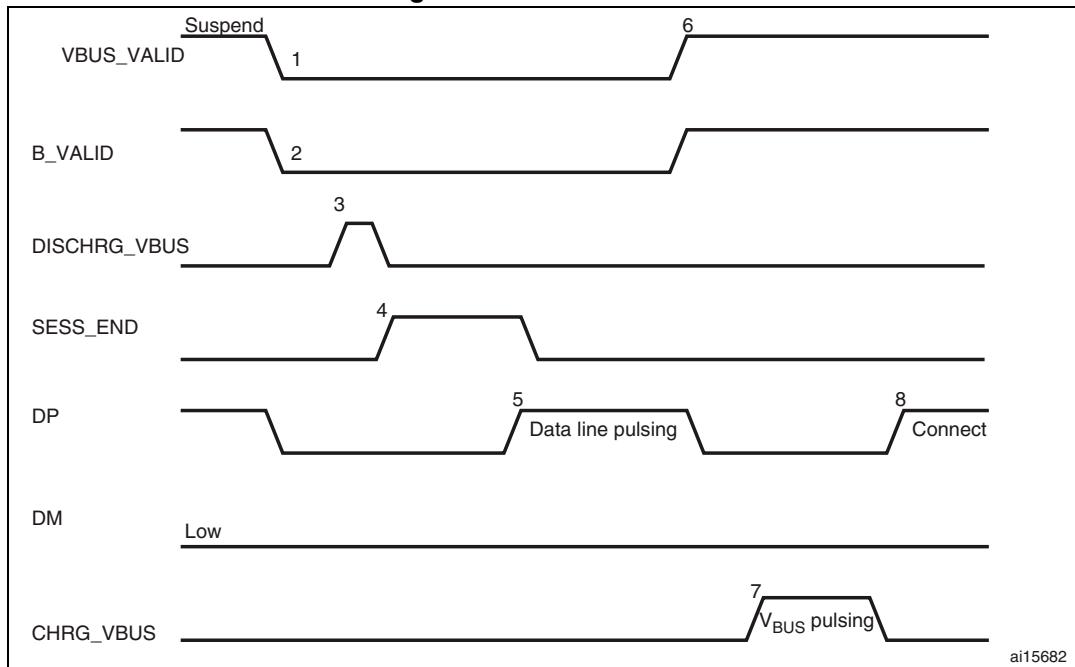
The following points refer and describe the signal numeration shown in the [Figure 427](#):

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  $\text{VBUS\_VALID}$  signal.
3. The device must detect SE0 for at least 2 ms to start SRP when  $V_{\text{BUS}}$  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  $V_{\text{BUS}}$  above the A-device session valid (2.0 V minimum) for  $V_{\text{BUS}}$  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  $\text{VBUS\_VALID}$  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 428. B-device SRP



ai15682

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 428](#):

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

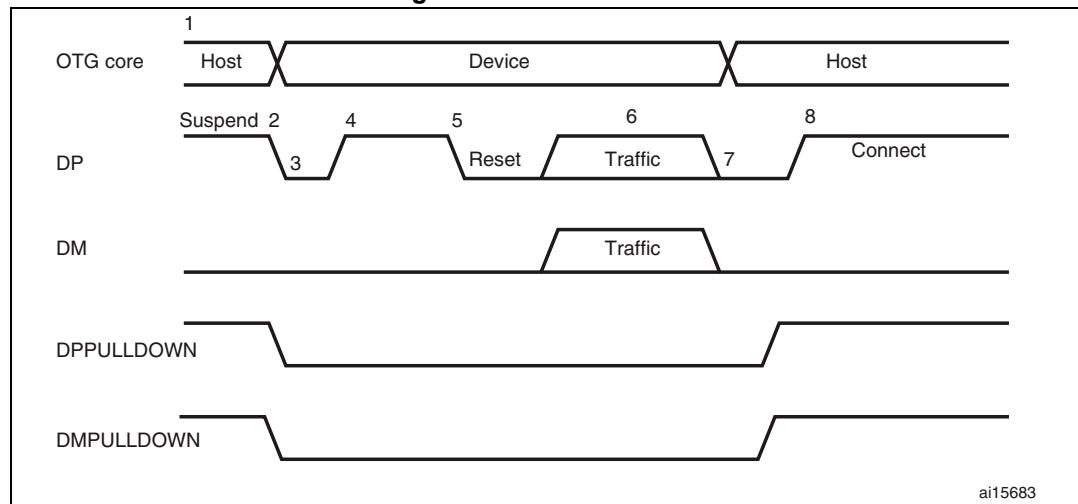
success status change bit in the OTG interrupt status register. The application reads the session request success bit in the OTG control and status register.

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 429. 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 429](#):

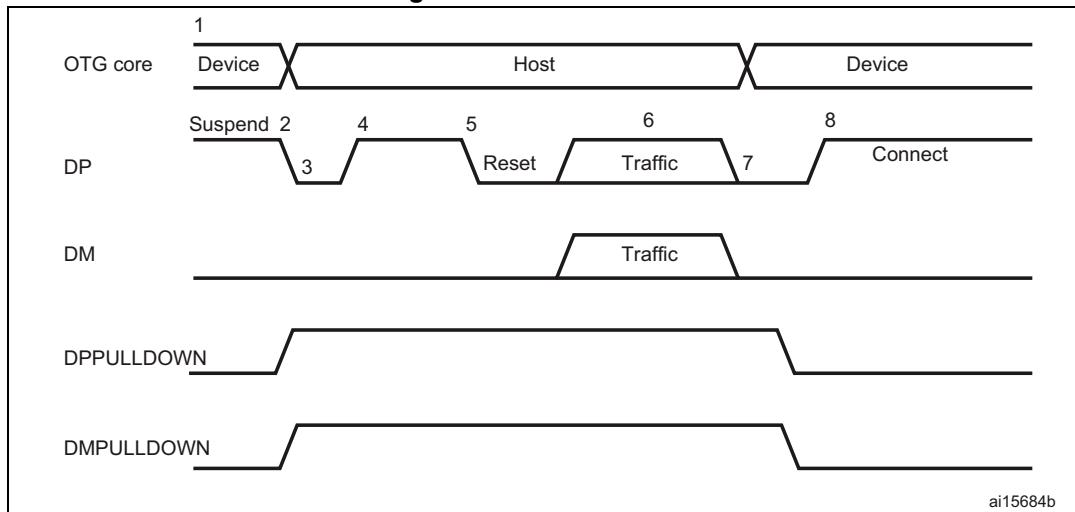
1. The OTG\_FS/OTG\_HS controller sends the B-device a SetFeature b\_hnp\_enable descriptor to enable HNP support. The B-device's ACK response indicates that the B-device supports HNP. The application must set host Set HNP Enable bit in the OTG

- 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 430. B-device HNP



ai15684b

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 430](#):

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\_HPRT) 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.

## 32 HDMI-CEC controller (HDMI-CEC)

### 32.1 Introduction

Consumer Electronics Control (CEC) is part of HDMI (High-Definition Multimedia Interface) standard as *appendix supplement 1*. It contains a protocol that provides high-level control functions between various audiovisual products. CEC operates at low speeds, with minimum processing and memory overhead.

The HDMI-CEC controller provides hardware support for this protocol.

### 32.2 HDMI-CEC controller main features

- Complies with HDMI-CEC v1.4 Specification
- 32 kHz CEC kernel with 2 clock source options
  - HSI RC oscillator with fixed prescaler (HSI/488)
  - LSE oscillator
- Works in Stop mode for ultra low-power applications
- Configurable Signal Free Time before start of transmission
  - Automatic by hardware, according to CEC state and transmission history
  - Fixed by software (7 timing options)
- Configurable Peripheral Address (OAR)
- Supports Listen mode
  - Enables reception of CEC messages sent to destination address different from OAR without interfering with the CEC line
- Configurable Rx-tolerance margin
  - Standard tolerance
  - Extended tolerance
- Receive-Error detection
  - Bit rising error (BRE), with optional stop of reception (BRESTP)
  - Short bit period error (SBPE)
  - Long bit period error (LBPE)
- Configurable error-bit generation
  - on BRE detection (BREGEN)
  - on LBPE detection (LBPEGEN)
  - always generated on SBPE detection
- Transmission error detection (TXERR)
- Arbitration Lost detection (ARBLST)
  - With automatic transmission retry
- Transmission underrun detection (TXUDR)
- Reception overrun detection (RXOVR)

## 32.3 HDMI-CEC functional description

### 32.3.1 HDMI-CEC pin

The CEC bus consists of a single bidirectional line that is used to transfer data in and out of the device. It is connected to a +3.3 V supply voltage via a 27 kΩ pull-up resistor. The output stage of the device must have an open-drain or open-collector to allow a wired-and connection.

The HDMI-CEC controller manages the CEC bidirectional line as an alternate function of a standard GPIO, assuming that it is configured as Alternate Function Open Drain. The 27 kΩ pull-up must be added externally to the STM32.

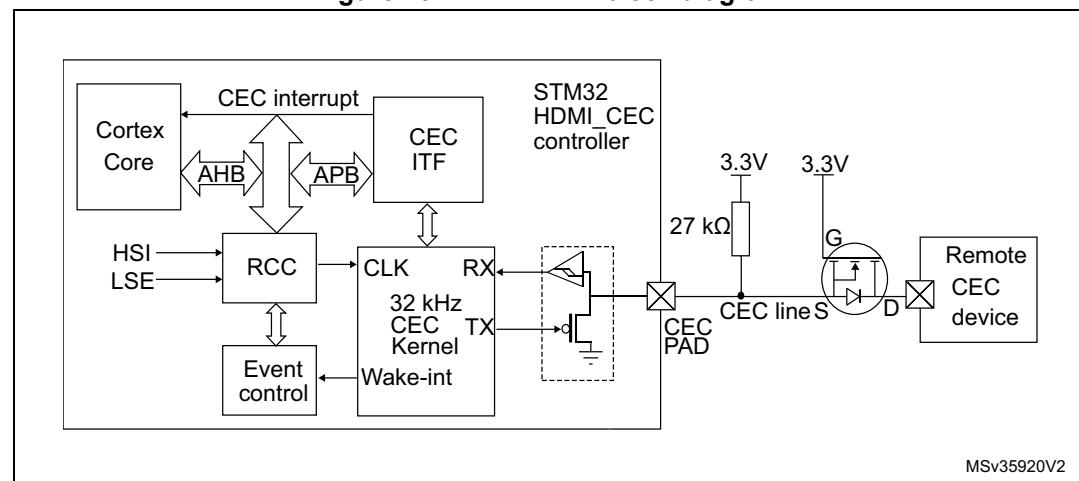
To not interfere with the CEC bus when the application power is removed, it is mandatory to isolate the CEC pin from the bus in such conditions. This could be done by using a MOS transistor, as shown on [Figure 431](#).

**Table 225. HDMI pin**

Name	Signal type	Remarks
CEC	bidirectional	two states: 1 = high impedance 0 = low impedance A 27 kΩ must be added externally.

### 32.3.2 HDMI-CEC block diagram

**Figure 431. HDMI-CEC block diagram**



### 32.3.3 Message description

All transactions on the CEC line consist of an initiator and one or more followers. The initiator is responsible for sending the message structure and the data. The follower is the recipient of any data and is responsible for setting any acknowledgment bits.

A message is conveyed in a single frame which consists of a start bit followed by a header block and optionally an opcode and a variable number of operand blocks.

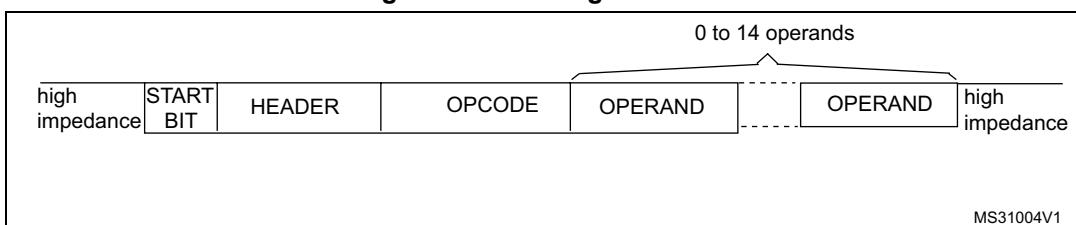
All these blocks are made of a 8-bit payload - most significant bit is transmitted first - followed by an end of message (EOM) bit and an acknowledge (ACK) bit.

The EOM bit is set in the last block of a message and kept reset in all others. In the event that a message contains additional blocks after an EOM is indicated, those additional blocks should be ignored. The EOM bit may be set in the header block to 'ping' other devices, to make sure they are active.

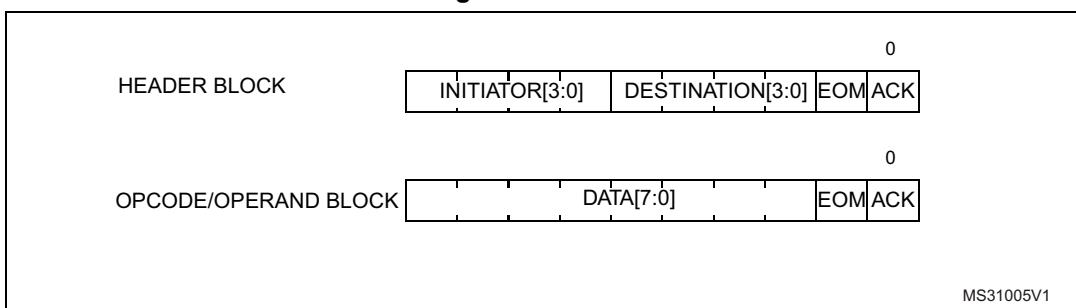
The acknowledge bit is always set to high impedance by the initiator so that it can be driven low either by the follower which has read its own address in the header or by the follower which needs to reject a broadcast message.

The header consists of the source logical address field, and the destination logical address field. Note that the special address 0xF is used for broadcast messages.

**Figure 432. Message structure**



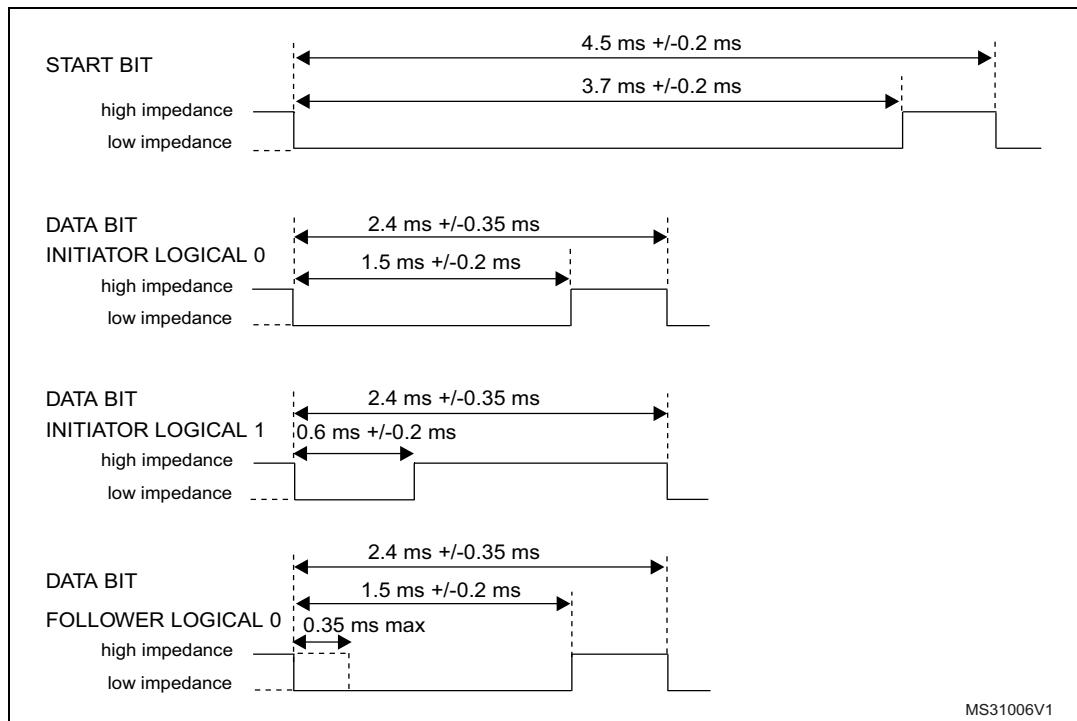
**Figure 433. Blocks**



### 32.3.4 Bit timing

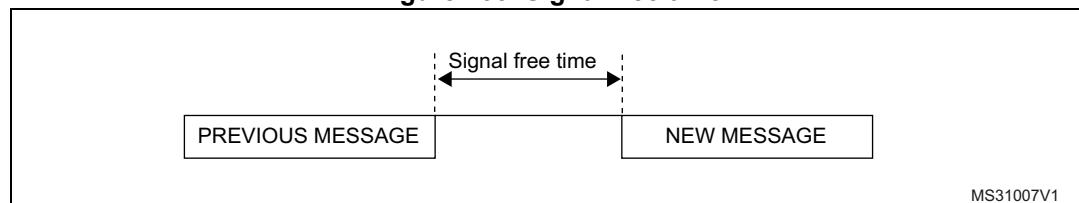
The format of the start bit is unique and identifies the start of a message. It should be validated by its low duration and its total duration.

All remaining data bits in the message, after the start bit, have consistent timing. The high to low transition at the end of the data bit is the start of the next data bit except for the final bit where the CEC line remains high.

**Figure 434. Bit timings**

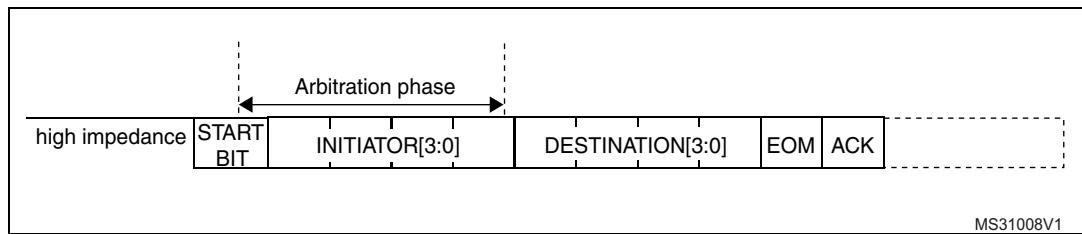
## 32.4 Arbitration

All devices that have to transmit - or retransmit - a message onto the CEC line have to ensure that it has been inactive for a number of bit periods. This signal free time is defined as the time starting from the final bit of the previous frame and depends on the initiating device and the current status as shown in the table below.

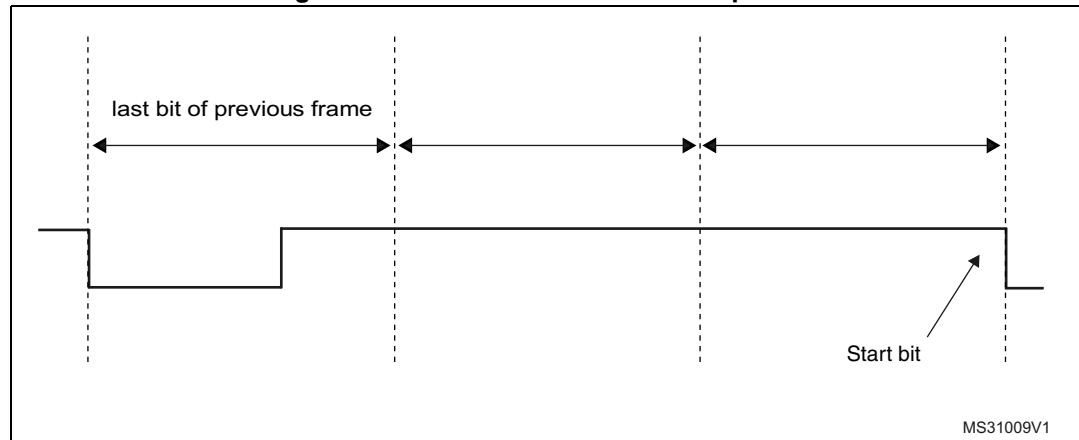
**Figure 435. Signal free time**

Since only one initiator is allowed at any one time, an arbitration mechanism is provided to avoid conflict when more than one initiator begins transmitting at the same time.

CEC line arbitration commences with the leading edge of the start bit and continues until the end of the initiator address bits within the header block. During this period, the initiator shall monitor the CEC line, if whilst driving the line to high impedance it reads it back to 0, it then assumes it has lost arbitration, stops transmitting and becomes a follower.

**Figure 436. Arbitration phase**

The [Figure 437](#) shows an example for a SFT of three nominal bit periods

**Figure 437. SFT of three nominal bit periods**

A configurable time window is counted before starting the transmission.

In the SFT=0x0 configuration the HDMI-CEC device performs automatic SFT calculation ensuring compliance with the HDMI-CEC Standard:

- 2.5 data bit periods if the CEC is the last bus initiator with unsuccessful transmission
- 4 data bit periods if the CEC is the new bus initiator
- 6 data bit periods if the CEC is the last bus initiator with successful transmission

This is done to guarantee the maximum priority to a failed transmission and the lowest one to the last initiator that completed successfully its transmission.

Otherwise there is the possibility to configure the SFT bits to count a fixed timing value. Possible values are 0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5 data bit periods.

### 32.4.1 SFT option bit

In case of SFTOPT=0 configuration SFT starts being counted when the start-of-transmission command is set by software (TXSOM=1).

In case of SFTOPT=1, SFT starts automatically being counted by the HDMI-CEC device when a bus-idle or line error condition is detected. If the SFT timer is completed at the time TXSOM command is set then transmission starts immediately without latency. If the SFT

timer is still running instead, the system waits until the timer elapses before transmission can start.

In case of SFTOPT=1 a bus-event condition starting the SFT timer is detected in the following cases:

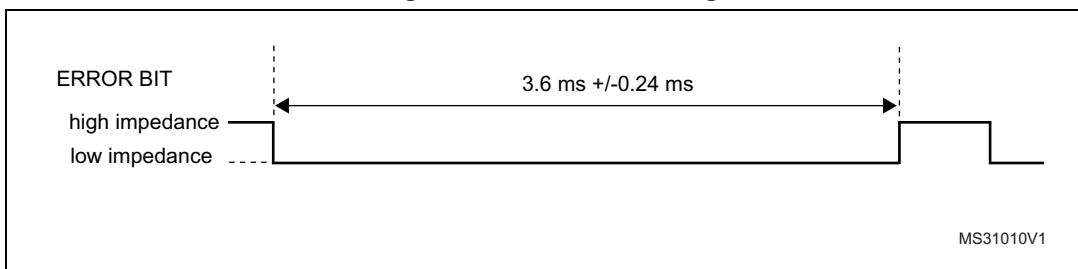
- In case of a regular end of transmission/reception, when TXEND/RXEND bits are set at the minimum nominal data bit duration of the last bit in the message (ACK bit).
- In case of a transmission error detection, SFT timer starts when the TXERR transmission error is detected (TXERR=1).
- In case of a missing acknowledge from the CEC follower, the SFT timer starts when the TXACKE bit is set, that is at the nominal sampling time of the ACK bit.
- In case of a transmission underrun error, the SFT timer starts when the TXUDR bit is set at the end of the ACK bit.
- In case of a receive error detection implying reception abort, the SFT timer starts at the same time the error is detected. If an error bit is generated, then SFT starts being counted at the end of the error bit.
- In case of a wrong start bit or of any uncodified low impedance bus state from idle, the SFT timer is restarted as soon as the bus comes back to hi-impedance idleness.

## 32.5 Error handling

### 32.5.1 Bit error

If a data bit - excluding the start bit - is considered invalid, the follower is expected to notify such error by generating a low bit period on the CEC line of 1.4 to 1.6 times the nominal data bit period, i.e. 3.6 ms nominally.

**Figure 438. Error bit timing**



### 32.5.2 Message error

A message is considered lost and therefore may be retransmitted under the following conditions:

- a message is not acknowledged in a directly addressed message
- a message is negatively acknowledged in a broadcast message
- a low impedance is detected on the CEC line while it is not expected (line error)

Three kinds of error flag can be detected when the CEC interface is receiving a data bit:

### 32.5.3 Bit Rising Error (BRE)

BRE (bit rising error): is set when a bit rising edge is detected outside the windows where it is expected (see [Figure 439](#)). BRE flag also generates a CEC interrupt if the BREIE=1.

In the case of a BRE detection, the message reception can be stopped according to the BRESTOP bit value and an error bit can be generated if BREGEN bit is set.

When BRE is detected in a broadcast message with BRESTOP=1 an error bit is generated even if BREGEN=0 to enforce initiator's retry of the failed transmission. Error bit generation can be disabled by configuring BREGEN=0, BRDNOGEN=1.

### 32.5.4 Short Bit Period Error (SBPE)

SBPE is set when a bit falling edge is detected earlier than expected (see [Figure 439](#)). SBPE flag also generates a CEC interrupt if the SBPEIE=1.

An error bit is always generated on the line in case of a SBPE error detection. An Error Bit is not generated upon SBPE detection only when Listen mode is set (LSTN=1) and the following conditions are met:

- A directly addressed message is received containing SBPE
- A broadcast message is received containing SBPE AND BRDNOGEN=1

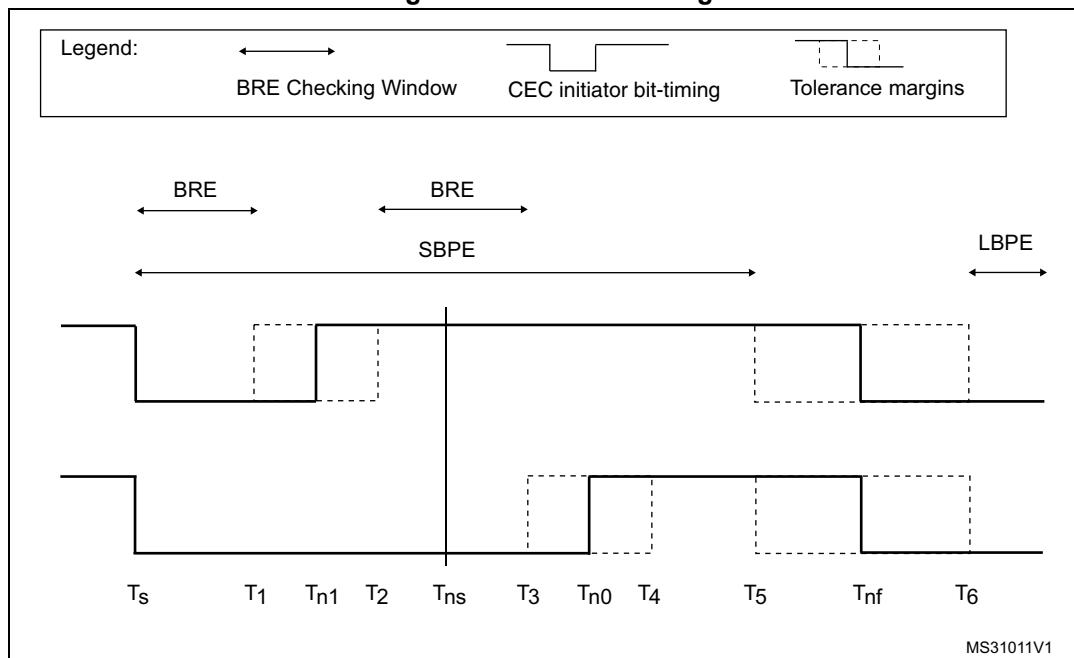
### 32.5.5 Long Bit Period Error (LBPE)

LBPE is set when a bit falling edge is not detected in a valid window (see [Figure 439](#)). LBPE flag also generates a CEC interrupt if the LBPEIE=1.

LBPE always stops the reception, an error bit is generated on the line when LBPEGEN bit is set.

When LBPE is detected in a broadcast message an error bit is generated even if LBPEGEN=0 to enforce initiator's retry of the failed transmission. Error bit generation can be disabled by configuring LBPEGEN=0, BRDNOGEN=1.

*Note:* *The BREGEN=1, BRESTOP=0 configuration must be avoided*

**Figure 439. Error handling**

MS31011V1

**Table 226. Error handling timing parameters**

Time	RXTOL	ms	Description
$T_s$	x	0	Bit start event.
$T_1$	1	0.3	The earliest time for a low - high transition when indicating a logical 1.
	0	0.4	
$T_{n1}$	x	0.6	The nominal time for a low - high transition when indicating a logical 1.
$T_2$	0	0.8	The latest time for a low - high transition when indicating a logical 1.
	1	0.9	
$T_{ns}$	x	1.05	Nominal sampling time.
$T_3$	1	1.2	The earliest time a device is permitted return to a high impedance state (logical 0).
	0	1.3	
$T_{n0}$	x	1.5	The nominal time a device is permitted return to a high impedance state (logical 0).
$T_4$	0	1.7	The latest time a device is permitted return to a high impedance state (logical 0).
	1	1.8	
$T_5$	1	1.85	The earliest time for the start of a following bit.
	0	2.05	
$T_{nf}$	x	2.4	The nominal data bit period.
$T_6$	0	2.75	The latest time for the start of a following bit.
	1	2.95	

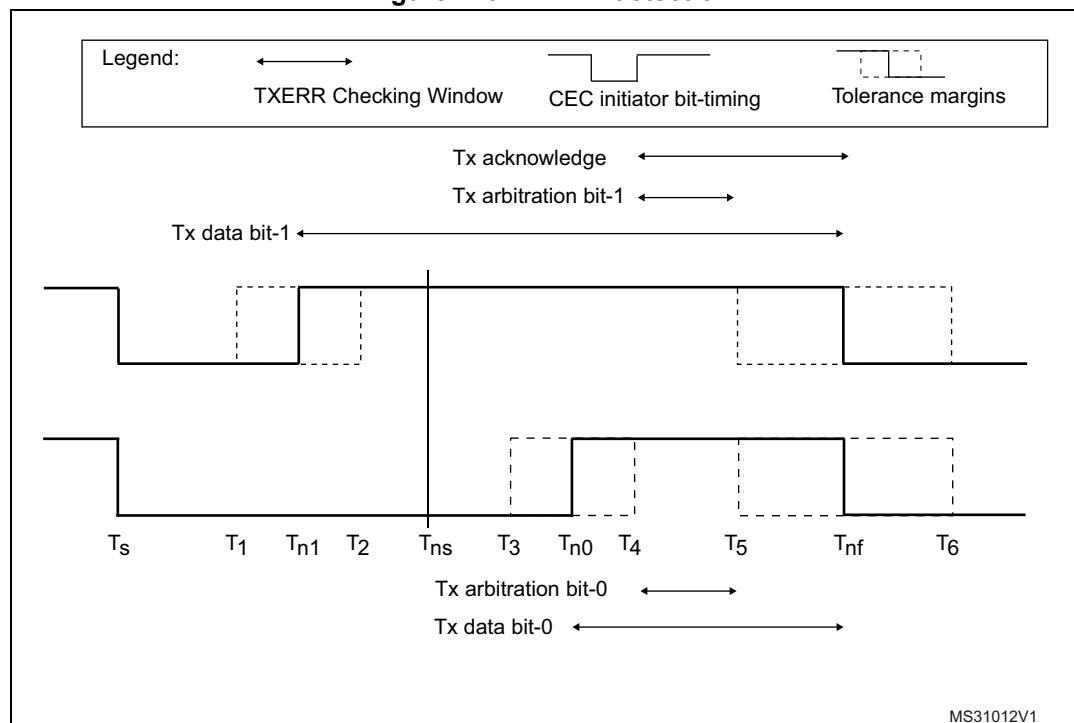
### 32.5.6 Transmission Error Detection (TXERR)

The CEC initiator sets the TXERR flag if detecting low impedance on the CEC line when it is transmitting high impedance and is not expecting a follower asserted bit. TXERR flag also generates a CEC interrupt if the TXERRIE=1.

TXERR assertion stops the message transmission. Application is in charge to retry the failed transmission up to 5 times.

TXERR checks are performed differently depending on the different states of the CEC line and on the RX tolerance configuration.

**Figure 440. TXERR detection**



**Table 227. TXERR timing parameters**

Time	RXTOL	ms	Description
$T_s$	x	0	Bit start event.
$T_1$	1	0.3	The earliest time for a low - high transition when indicating a logical 1.
	0	0.4	
$T_{n1}$	x	0.6	The nominal time for a low - high transition when indicating a logical 1.
$T_2$	0	0.8	The latest time for a low - high transition when indicating a logical 1.
	1	0.9	
$T_{ns}$	x	1.05	Nominal sampling time.
$T_3$	1	1.2	The earliest time a device is permitted return to a high impedance state (logical 0).
	0	1.3	

**Table 227. TXERR timing parameters (continued)**

Time	RXTOL	ms	Description
T <sub>n0</sub>	x	1.5	The nominal time a device is permitted return to a high impedance state (logical 0).
T <sub>4</sub>	0	1.7	The latest time a device is permitted return to a high impedance state (logical 0).
	1	1.8	
T <sub>5</sub>	1	1.85	The earliest time for the start of a following bit.
	0	2.05	
T <sub>nf</sub>	x	2.4	The nominal data bit period.
T <sub>6</sub>	0	2.75	The latest time for the start of a following bit.
	1	2.95	

## 32.6 HDMI-CEC interrupts

An interrupt can be produced:

- during reception if a Receive Block Transfer is finished or if a Receive Error occurs.
- during transmission if a Transmit Block Transfer is finished or if a Transmit Error occurs.

**Table 228. HDMI-CEC interrupts**

Interrupt event	Event flag	Enable Control bit
Rx-Byte Received	RXBR	RXBRIE
End of reception	RXEND	RXENDIE
Rx-Overrun	RXOVR	RXOVRIE
RxBit Rising Error	BRE	BREIE
Rx-Short Bit Period Error	SBPE	SBPEIE
Rx-Long Bit Period Error	LBPE	LBPEIE
Rx-Missing Acknowledge Error	RXACKE	RXACKEIE
Arbitration lost	ARBLST	ARBLSTIE
Tx-Byte Request	TXBR	TXBRIE
End of transmission	TXEND	TXENDIE
Tx-Buffer Underrun	TXUDR	TXUDRIE
Tx-Error	TXERR	TXERRIE
Tx-Missing Acknowledge Error	TXACKE	TXACKEIE

## 32.7 HDMI-CEC registers

Refer to [Section 1.1 on page 51](#) for a list of abbreviations used in register descriptions.

### 32.7.1 CEC control register (CEC\_CR)

Address offset: 0x00

Reset value: 0x0000 0000

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

Bits 31:3 Reserved, must be kept at reset value.

#### Bit 2 TXEOM: Tx End Of Message

The TXEOM bit is set by software to command transmission of the last byte of a CEC message. TXEOM is cleared by hardware at the same time and under the same conditions as for TXSOM.

0: TXDR data byte is transmitted with EOM=0  
1: TXDR data byte is transmitted with EOM=1

Note: TXEOM must be set when CECEN=1

*TXEOM must be set before writing transmission data to TXDR*

*If TXEOM is set when TXSOM=0, transmitted message will consist of 1 byte (HEADER) only (PING message)*

#### Bit 1 TXSOM: Tx Start Of Message

TXSOM is set by software to command transmission of the first byte of a CEC message. If the CEC message consists of only one byte, TXEOM must be set before of TXSOM.

Start-Bit is effectively started on the CEC line after SFT is counted. If TXSOM is set while a message reception is ongoing, transmission will start after the end of reception.

TXSOM is cleared by hardware after the last byte of the message is sent with a positive acknowledge (TXEND=1), in case of transmission underrun (TXUDR=1), negative acknowledge (TXACKE=1), and transmission error (TXERR=1). It is also cleared by CECEN=0. It is not cleared and transmission is automatically retried in case of arbitration lost (ARBLST=1).

TXSOM can be also used as a status bit informing application whether any transmission request is pending or under execution. The application can abort a transmission request at any time by clearing the CECEN bit.

0: No CEC transmission is on-going

1: CEC transmission command

**Note:** TXSOM must be set when CECEN=1

*TXSOM must be set when transmission data is available into TXDR*

*HEADER's first four bits containing own peripheral address are taken from TXDR[7:4], not from CEC\_CFGR.OAR which is used only for reception*

#### Bit 0 **CECEN:** CEC Enable

The CECEN bit is set and cleared by software. CECEN=1 starts message reception and enables the TXSOM control. CECEN=0 disables the CEC peripheral, clears all bits of CEC\_CR register and aborts any on-going reception or transmission.

0: CEC peripheral is off

1: CEC peripheral is on

### 32.7.2 CEC configuration register (CEC\_CFGR)

This register is used to configure the HDMI-CEC controller.

Address offset: 0x04

Reset value: 0x0000 0000

**Caution:** It is mandatory to write CEC\_CFGR only when CECEN=0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LSTN	OAR[14:0]														
rw	rw														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	SFT OPT	BRDN OGEN	LBPE GEN	BRE GEN	BRE STP	RX TOL	SFT[2:0]		
							rw	rw	rw	rw	rw	rw	rw		

#### Bit 31 **LSTN:** Listen mode

LSTN bit is set and cleared by software.

0: CEC peripheral receives only message addressed to its own address (OAR). Messages addressed to different destination are ignored. Broadcast messages are always received.

1: CEC peripheral receives messages addressed to its own address (OAR) with positive acknowledge. Messages addressed to different destination are received, but without interfering with the CEC bus: no acknowledge sent.

#### Bits 30:16 **OAR:** Own addresses configuration

The OAR bits are set by software to select which destination logical addresses has to be considered in receive mode. Each bit, when set, enables the CEC logical address identified by the given bit position.

At the end of HEADER reception, the received destination address is compared with the enabled addresses. In case of matching address, the incoming message is acknowledged and received. In case of non-matching address, the incoming message is received only in listen mode (LSTN=1), but without acknowledge sent. Broadcast messages are always received.

Example:

OAR = 0b000 0000 0010 0001 means that CEC acknowledges addresses 0x0 and 0x5. Consequently, each message directed to one of these addresses is received.

#### Bits 15:9 Reserved, must be kept at reset value.

**Bit 8 SFTOP: SFT Option Bit**

The SFTOPT bit is set and cleared by software.

0: SFT timer starts when TXSOM is set by software

1: SFT timer starts automatically at the end of message transmission/reception.

**Bit 7 BRDNOGEN: Avoid Error-Bit Generation in Broadcast**

The BRDNOGEN bit is set and cleared by software.

0: BRE detection with BRESTP=1 and BREGEN=0 on a broadcast message generates an Error-Bit on the CEC line. LBPE detection with LBPEGEN=0 on a broadcast message generates an Error-Bit on the CEC line

1: Error-Bit is not generated in the same condition as above. An Error-Bit is not generated even in case of an SBPE detection in a broadcast message if listen mode is set.

**Bit 6 LBPEGEN: Generate Error-Bit on Long Bit Period Error**

The LBPEGEN bit is set and cleared by software.

0: LBPE detection does not generate an Error-Bit on the CEC line

1: LBPE detection generates an Error-Bit on the CEC line

*Note: If BRDNOGEN=0, an Error-bit is generated upon LBPE detection in broadcast even if LBPEGEN=0*

**Bit 5 BREGEN: Generate Error-Bit on Bit Rising Error**

The BREGEN bit is set and cleared by software.

0: BRE detection does not generate an Error-Bit on the CEC line

1: BRE detection generates an Error-Bit on the CEC line (if BRESTP is set)

*Note: If BRDNOGEN=0, an Error-bit is generated upon BRE detection with BRESTP=1 in broadcast even if BREGEN=0*

**Bit 4 BRESTOP: Rx-Stop on Bit Rising Error**

The BRESTOP bit is set and cleared by software.

0: BRE detection does not stop reception of the CEC message. Data bit is sampled at 1.05 ms.

1: BRE detection stops message reception

**Bit 3 RXTOL: Rx-Tolerance**

The RXTOL bit is set and cleared by software.

0: Standard tolerance margin:

- Start-Bit, +/- 200 µs rise, +/- 200 µs fall.
- Data-Bit: +/- 200 µs rise, +/- 350 µs fall.

1: Extended Tolerance

- Start-Bit: +/- 400 µs rise, +/- 400 µs fall
- Data-Bit: +/-300 µs rise, +/- 500 µs fall

**Bits 2:0 SFT: Signal Free Time**

SFT bits are set by software. In the SFT=0x0 configuration the number of nominal data bit periods waited before transmission is ruled by hardware according to the transmission history. In all the other configurations the SFT number is determined by software.

” 0x0

- 2.5 Data-Bit periods if CEC is the last bus initiator with unsuccessful transmission (ARBLST=1, TXERR=1, TXUDR=1 or TXACKE= 1)
- 4 Data-Bit periods if CEC is the new bus initiator
- 6 Data-Bit periods if CEC is the last bus initiator with successful transmission (TXEOM=1)

” 0x1: 0.5 nominal data bit periods

” 0x2: 1.5 nominal data bit periods

” 0x3: 2.5 nominal data bit periods

” 0x4: 3.5 nominal data bit periods

” 0x5: 4.5 nominal data bit periods

” 0x6: 5.5 nominal data bit periods

” 0x7: 6.5 nominal data bit periods

### 32.7.3 CEC Tx data register (CEC\_TXDR)

Address offset: 0x8

Reset value: 0x0000 0000

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

Bits 31:8 Reserved, must be kept at reset value.

Bits 7:0 **TXD[7:0]**: Tx Data register.

TXD is a write-only register containing the data byte to be transmitted.

### 32.7.4 CEC Rx Data Register (CEC\_RXDR)

Address offset: 0xC

Reset value: 0x0000 0000

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

Bits 31:8 Reserved, must be kept at reset value.

Bits 7:0 **RXD[7:0]**: Rx Data register.

RXD is read-only and contains the last data byte which has been received from the CEC line.

### 32.7.5 CEC Interrupt and Status Register (CEC\_ISR)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	TX ACKE	TX ERR	TX UDR	TX END	TXBR	ARB LST	RX ACKE	LBPE	SBPE	BRE	RX OVR	RX END	RXBR
			rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:13 Reserved, must be kept at reset value.

**Bit 12 TXACKE:** Tx-Missing Acknowledge Error

In transmission mode, TXACKE is set by hardware to inform application that no acknowledge was received. In case of broadcast transmission, TXACKE informs application that a negative acknowledge was received. TXACKE aborts message transmission and clears TXSOM and TXEOM controls.

TXACKE is cleared by software write at 1.

**Bit 11 TXERR:** Tx-Error

In transmission mode, TXERR is set by hardware if the CEC initiator detects low impedance on the CEC line while it is released. TXERR aborts message transmission and clears TXSOM and TXEOM controls.

TXERR is cleared by software write at 1.

**Bit 10 TXUDR:** Tx-Buffer Underrun

In transmission mode, TXUDR is set by hardware if application was not in time to load TXDR before of next byte transmission. TXUDR aborts message transmission and clears TXSOM and TXEOM control bits.

TXUDR is cleared by software write at 1

**Bit 9 TXEND:** End of Transmission

TXEND is set by hardware to inform application that the last byte of the CEC message has been successfully transmitted. TXEND clears the TXSOM and TXEOM control bits.

TXEND is cleared by software write at 1.

**Bit 8 TXBR:** Tx-Byte Request

TXBR is set by hardware to inform application that the next transmission data has to be written to TXDR. TXBR is set when the 4th bit of currently transmitted byte is sent. Application must write the next byte to TXDR within 6 nominal data-bit periods before transmission underrun error occurs (TXUDR).

TXBR is cleared by software write at 1.

**Bit 7 ARBLST:** Arbitration Lost

ARBLST is set by hardware to inform application that CEC device is switching to reception due to arbitration lost event following the TXSOM command. ARBLST can be due either to a contending CEC device starting earlier or starting at the same time but with higher HEADER priority. After ARBLST assertion TXSOM bit keeps pending for next transmission attempt.

ARBLST is cleared by software write at 1.

**Bit 6 RXACKE:** Rx-Missing Acknowledge

In receive mode, RXACKE is set by hardware to inform application that no acknowledge was seen on the CEC line. RXACKE applies only for broadcast messages and in listen mode also for not directly addressed messages (destination address not enabled in OAR). RXACKE aborts message reception.

RXACKE is cleared by software write at 1.

**Bit 5 LBPE:** Rx-Long Bit Period Error

LBPE is set by hardware in case a Data-Bit waveform is detected with Long Bit Period Error. LBPE is set at the end of the maximum bit-extension tolerance allowed by RXTOL, in case falling edge is still longing. LBPE always stops reception of the CEC message. LBPE generates an Error-Bit on the CEC line if LBPEGEN=1. In case of broadcast, Error-Bit is generated even in case of LBPEGEN=0.

LBPE is cleared by software write at 1.

**Bit 4 SBPE:** Rx-Short Bit Period Error

SBPE is set by hardware in case a Data-Bit waveform is detected with Short Bit Period Error. SBPE is set at the time the anticipated falling edge occurs. SBPE generates an Error-Bit on the CEC line.

SBPE is cleared by software write at 1.

Bit 3 **BRE**: Rx-Bit Rising Error

BRE is set by hardware in case a Data-Bit waveform is detected with Bit Rising Error. BRE is set either at the time the misplaced rising edge occurs, or at the end of the maximum BRE tolerance allowed by RXTOL, in case rising edge is still longing. BRE stops message reception if BRESTP=1. BRE generates an Error-Bit on the CEC line if BREGEN=1.

BRE is cleared by software write at 1.

Bit 2 **RXOVR**: Rx-Overrun

RXOVR is set by hardware if RXBR is not yet cleared at the time a new byte is received on the CEC line and stored into RXD. RXOVR assertion stops message reception so that no acknowledge is sent. In case of broadcast, a negative acknowledge is sent.

RXOVR is cleared by software write at 1.

Bit 1 **RXEND**: End Of Reception

RXEND is set by hardware to inform application that the last byte of a CEC message is received from the CEC line and stored into the RXD buffer. RXEND is set at the same time of RXBR.

RXEND is cleared by software write at 1.

Bit 0 **RXBR**: Rx-Byte Received

The RXBR bit is set by hardware to inform application that a new byte has been received from the CEC line and stored into the RXD buffer.

RXBR is cleared by software write at 1.

**32.7.6 CEC interrupt enable register (CEC\_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.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	TXACK IE	TXERR IE	TX UDRIE	TXEND IE	TXBR IE	ARBLST IE	RXACK IE	LBPE IE	SBPE IE	BREIE	RXOVR IE	RXEND IE	RXBR IE
			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:13 Reserved, must be kept at reset value.

Bit 12 **TXACKIE**: Tx-Missing Acknowledge Error Interrupt Enable

The TXACKIE bit is set and cleared by software.

0: TXACKE interrupt disabled

1: TXACKE interrupt enabled

Bit 11 **TXERRIE**: Tx-Error Interrupt Enable

The TXERRIE bit is set and cleared by software.

0: TXERR interrupt disabled

1: TXERR interrupt enabled

Bit 10 **TXUDRIE**: Tx-Underrun Interrupt Enable

The TXUDRIE bit is set and cleared by software.

0: TXUDR interrupt disabled

1: TXUDR interrupt enabled

**Bit 9 TXENDIE:** Tx-End Of Message Interrupt Enable

The TXENDIE bit is set and cleared by software.

- 0: TXEND interrupt disabled
- 1: TXEND interrupt enabled

**Bit 8 TXBRIE:** Tx-Byte Request Interrupt Enable

The TXBRIE bit is set and cleared by software.

- 0: TXBR interrupt disabled
- 1: TXBR interrupt enabled

**Bit 7 ARBLSTIE:** Arbitration Lost Interrupt Enable

The ARBLSTIE bit is set and cleared by software.

- 0: ARBLST interrupt disabled
- 1: ARBLST interrupt enabled

**Bit 6 RXACKIE:** Rx-Missing Acknowledge Error Interrupt Enable

The RXACKIE bit is set and cleared by software.

- 0: RXACKE interrupt disabled
- 1: RXACKE interrupt enabled

**Bit 5 LBPEIE:** Long Bit Period Error Interrupt Enable

The LBPEIE bit is set and cleared by software.

- 0: LBPE interrupt disabled
- 1: LBPE interrupt enabled

**Bit 4 SBPEIE:** Short Bit Period Error Interrupt Enable

The SBPEIE bit is set and cleared by software.

- 0: SBPE interrupt disabled
- 1: SBPE interrupt enabled

**Bit 3 BREIE:** Bit Rising Error Interrupt Enable

The BREIE bit is set and cleared by software.

- 0: BRE interrupt disabled
- 1: BRE interrupt enabled

**Bit 2 RXOVRIE:** Rx-Buffer Overrun Interrupt Enable

The RXOVRIE bit is set and cleared by software.

- 0: RXOVR interrupt disabled
- 1: RXOVR interrupt enabled

**Bit 1 RXENDIE:** End Of Reception Interrupt Enable

The RXENDIE bit is set and cleared by software.

- 0: RXEND interrupt disabled
- 1: RXEND interrupt enabled

**Bit 0 RXBRIE:** Rx-Byte Received Interrupt Enable

The RXBRIE bit is set and cleared by software.

- 0: RXBR interrupt disabled
- 1: RXBR interrupt enabled

**Caution:** (\*) It is mandatory to write CEC\_IER only when CECEN=0

### 32.7.7 HDMI-CEC register map

The following table summarizes the HDMI-CEC registers.

**Table 229. HDMI-CEC register map and reset values**

Offset	Register	Reset value	31
0x00	<b>CEC_CR</b>	Res.	30
0x04	<b>CEC_CFGR</b>	0	29
0x08	<b>CEC_TXDR</b>	Res.	28
0x0C	<b>CEC_RXDR</b>	Res.	27
0x10	<b>CEC_ISR</b>	Res.	26
0x14	<b>CEC_IER</b>	Res.	25
	Reset value	0	24
	Reset value	0	23
	Reset value	0	22
	Reset value	0	21
	Reset value	0	20
	Reset value	0	19
	Reset value	0	18
	Reset value	0	17
	Reset value	0	16
	Reset value	0	15
	Reset value	0	14
	Reset value	0	13
	o TXACKIE	o TXACKE	12
	o TXERRIE	o TXERR	11
	o TXUDRIE	o TXUDR	10
	o TXENDIE	o TXEND	9
	o TXBRIE	o TXBR	8
	o ARBLSTIE	o ARBLST	7
	o RXACKIE	o RXACKE	6
	o LBPEIE	o LBPE	5
	o SBPEIE	o SBPE	4
	o BREIE	o BRE	3
	o RXOVRIE	o RXOVR	2
	o RXENDIE	o RXEND	1
	o RXBRIE	o RXBR	0
			TXD[7:0]
		0 0	RXD[7:0]
		0 0	TXEOM
		0 0	TXSOM
		0 0	CECEN

Refer to [Section 2.2.2 on page 56](#) for the register boundary addresses.

## 33 Debug support (DBG)

### 33.1 Overview

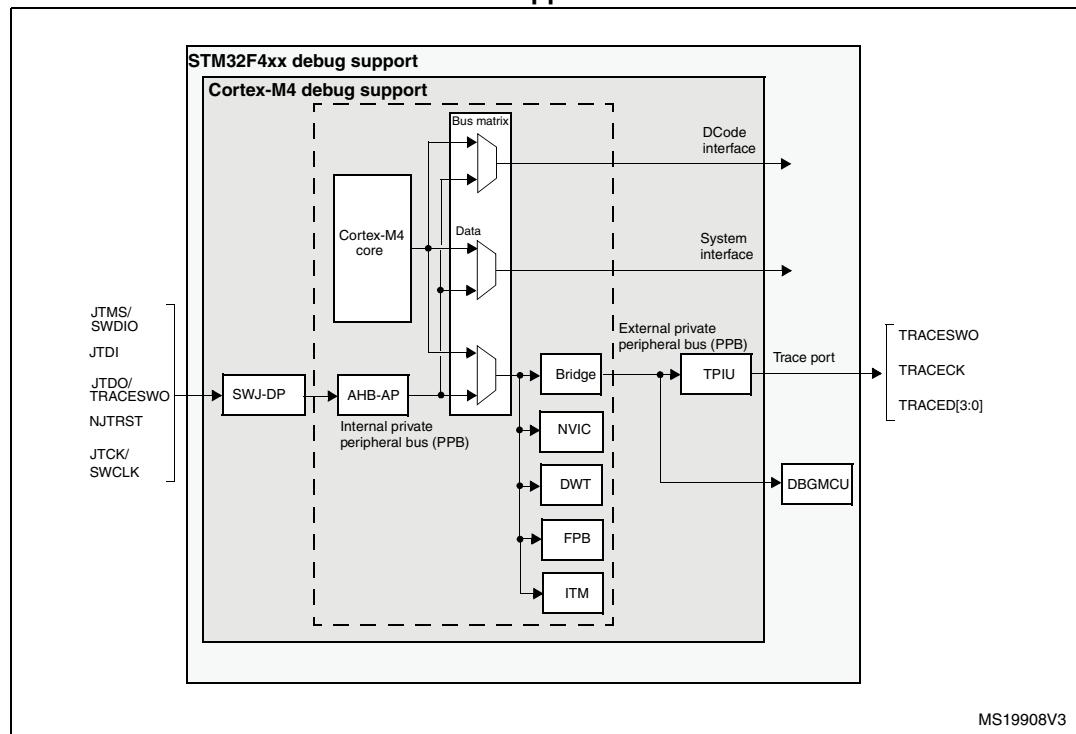
The STM32F446xx are built around a Cortex<sup>®</sup>-M4 with FPU core which contains hardware extensions for advanced debugging features. The debug extensions allow the core to be stopped either on a given instruction fetch (breakpoint) or data access (watchpoint). When stopped, the core's internal 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 STM32F446xx MCUs.

Two interfaces for debug are available:

- Serial wire
- JTAG debug port

**Figure 441. Block diagram of STM32 MCU and Cortex<sup>®</sup>-M4 with FPU-level debug support**



Note:

*The debug features embedded in the Cortex<sup>®</sup>-M4 with FPU core are a subset of the ARM<sup>®</sup> CoreSight Design Kit.*

The ARM® Cortex®-M4 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
- TPUI: 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 STM32F446xx:

- 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®-M4 with FPU core, refer to the Cortex®-M4 with FPU-r0p1 Technical Reference Manual and to the CoreSight Design Kit-r0p1 TRM (see [Section 33.2: Reference ARM® documentation](#)).*

## 33.2 Reference ARM® documentation

- Cortex®-M4 with FPU r0p1 Technical Reference Manual (TRM)  
(see Related documents on page 1)
- ARM® Debug Interface V5
- ARM® CoreSight Design Kit revision r0p1 Technical Reference Manual

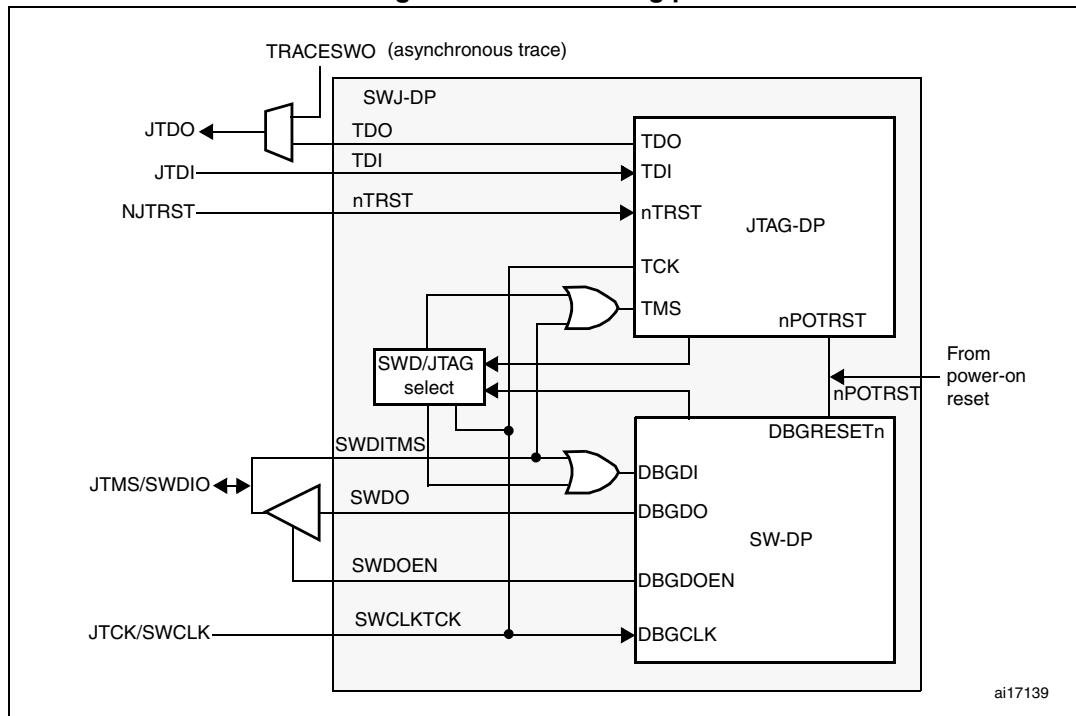
## 33.3 SWJ debug port (serial wire and JTAG)

The core of the STM32F446xx 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 442. SWJ debug port



*Figure 442* 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.

### 33.3.1 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

## 33.4 Pinout and debug port pins

The STM32F446xx MCUs are available in various packages with different numbers of available pins. As a result, some functionality (ETM) related to pin availability may differ between packages.

### 33.4.1 SWJ debug port pins

Five pins are used as outputs from the STM32F446xx for the SWJ-DP as *alternate functions* of general-purpose I/Os. These pins are available on all packages.

**Table 230. 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

### 33.4.2 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 STM32F446xx MCUs offers the possibility of disabling some or all of the SWJ-DP ports and so, of releasing the associated pins for general-purpose IO (GPIO) usage. For more details on how to disable SWJ-DP port pins, refer to [Section 7.3.2: I/O pin multiplexer and mapping](#).

**Table 231. 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 nTRST, 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.).

### 33.4.3 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.*

### 33.4.4 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.*

### 33.5 STM32F446xx JTAG TAP connection

The STM32F446xx MCUs integrate two serially connected JTAG TAPs, the boundary scan TAP (IR is 5-bit wide) and the Cortex®-M4 with FPU TAP (IR is 4-bit wide).

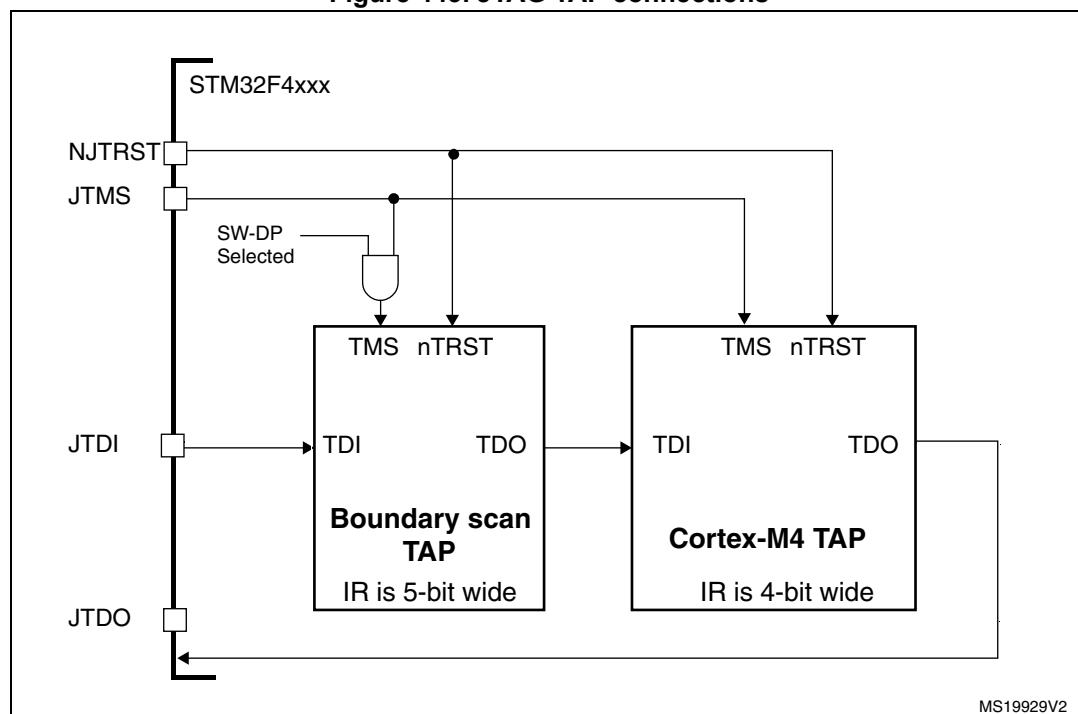
To access the TAP of the Cortex®-M4 with FPU for debug purposes:

1. First, it is necessary to shift the BYPASS instruction of the boundary scan TAP.
2. Then, for each IR shift, the scan chain contains 9 bits (=5+4) and the unused TAP instruction must be shifted in using the BYPASS instruction.
3. For each data shift, the unused TAP, which is in BYPASS mode, adds 1 extra data bit in the data scan chain.

**Note:**

**Important:** Once Serial-Wire is selected using the dedicated ARM® JTAG sequence, the boundary scan TAP is automatically disabled (JTMS forced high).

Figure 443. JTAG TAP connections



## 33.6 ID codes and locking mechanism

There are several ID codes inside the STM32F446xx 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.

### 33.6.1 MCU device ID code

The STM32F446xx 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 33.16 on page 1304](#)). This code is accessible using the JTAG debug pCat.2ort (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

Bits 15:12 Reserved, must be kept at reset value.

Bits 11:0 **DEV\_ID[11:0]**: Device identifier  
The device ID is 0x421

### 33.6.2 Boundary scan TAP

#### JTAG ID code

The TAP of the STM32F446xx BSC (boundary scan) integrates a JTAG ID code equal to 0x06413041.

### 33.6.3 Cortex®-M4 with FPU TAP

The TAP of the ARM® Cortex®-M4 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 0x4BA00477 (corresponds to Cortex®-M4 with FPU r0p1, see [Section 33.2: Reference ARM® documentation](#)).

### 33.6.4 Cortex®-M4 with FPU JEDEC-106 ID code

The ARM® Cortex®-M4 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.

## 33.7 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®-M4 with FPU r0p1 Technical Reference Manual (*TRM*), for references, see [Section 33.2: Reference ARM® documentation](#)).

**Table 232. JTAG debug port data registers**

IR(3:0)	Data register	Details
1111	BYPASS [1 bit]	-
1110	IDCODE [32 bits]	ID CODE 0x4BA00477 (ARM® Cortex®-M4 with FPU r0p1 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 <a href="#">Table 233</a> for a description of the A(3:2) bits

**Table 232. 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"> <li>– When transferring data IN:           <ul style="list-style-type: none"> <li>Bits 34:3 = DATA[31:0] = 32-bit data to shift in for a write request</li> <li>Bits 2:1 = A[3:2] = 2-bit address (sub-address AP registers).</li> <li>Bit 0 = RnW= Read request (1) or write request (0).</li> </ul> </li> <li>– When transferring data OUT:           <ul style="list-style-type: none"> <li>Bits 34:3 = DATA[31:0] = 32-bit data which is read following a read request</li> <li>Bits 2:0 = ACK[2:0] = 3-bit Acknowledge:               <ul style="list-style-type: none"> <li>010 = OK/FAULT</li> <li>001 = WAIT</li> <li>OTHER = reserved</li> </ul> </li> </ul> </li> </ul> <p>There are many AP Registers (see AHB-AP) addressed as the combination of:</p> <ul style="list-style-type: none"> <li>– The shifted value A[3:2]</li> <li>– The current value of the DP SELECT register</li> </ul>
1000	ABORT [35 bits]	<p>Abort register</p> <ul style="list-style-type: none"> <li>– Bits 31:1 = Reserved</li> <li>– Bit 0 = DAPABORT: write 1 to generate a DAP abort.</li> </ul>

**Table 233. 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)

## 33.8 SW debug port

### 33.8.1 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.

### 33.8.2 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 234. Packet request (8-bits)**

Bit	Name	Description
0	Start	Must be “1”
1	APnDP	0: DP Access 1: AP Access
2	RnW	0: Write Request 1: Read Request
4:3	A(3:2)	Address field of the DP or AP registers (refer to <a href="#">Table 233</a> )
5	Parity	Single bit parity of preceding bits
6	Stop	0
7	Park	Not driven by the host. Must be read as “1” by the target because of the pull-up

Refer to the Cortex®-M4 with FPUr0p1 *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 235. 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 236. 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.

### 33.8.3 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 **0x2BA01477** (corresponding to Cortex®-M4 with FPU r0p1).

- 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®-M4 with FPU r0p1 TRM and the CoreSight Design Kit r0p1 TRM.

### 33.8.4 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.

### 33.8.5 SW-DP registers

Access to these registers are initiated when APnDP=0

Table 237. 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. <b>0x2BA01477</b> (identifies the SW-DP)
00	Write	-	ABORT	-

**Table 237. 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

### 33.8.6 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

### 33.9 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®-M4 with FPU includes 9 x 32-bits registers:

**Table 238. Cortex®-M4 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®-M4 with FPU r0p1 TRM for further details.

### 33.10 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 239. 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 <b>TRCENA</b> 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®-M4 with FPU r0p1 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.

### 33.11 Capability of the debugger host to connect under system reset

The reset system of the STM32F446xx 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®-M4 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.*

### 33.12 FPB (Flash patch breakpoint)

The FPB unit:

- implements hardware breakpoints
- patches code and data from code space to system space. This feature gives the possibility to correct software bugs located in the Code Memory Space.

The use of a Software Patch or a Hardware Breakpoint is exclusive.

The FPB consists of:

- 2 literal comparators for matching against literal loads from Code Space and remapping to a corresponding area in the System Space.
- 6 instruction comparators for matching against instruction fetches from Code Space. They can be used either to remap to a corresponding area in the System Space or to generate a Breakpoint Instruction to the core.

### 33.13 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

### 33.14 ITM (instrumentation trace macrocell)

#### 33.14.1 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®-M4 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 you program or use the ITM.

#### 33.14.2 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 240. 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 33.17.2: TRACE pin assignment](#) and [Section 33.16.3: 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)

## 33.15 ETM (Embedded trace macrocell)

### 33.15.1 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 33.13: 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 33.17: TPIU \(trace port interface unit\)](#)) and then outputs the complete packet sequence to the debugger host.

### 33.15.2 Signal protocol, packet types

This part is described in the chapter 7 ETMv3 Signal Protocol of the ARM® IHI 0014N document.

### 33.15.3 Main ETM registers

For more information on registers refer to the chapter 3 of the ARM® IHI 0014N specification.

**Table 241. Main ETM registers**

Address	Register	Details
0xE0041FB0	ETM Lock Access	Write 0xC5ACCE55 to unlock the write access to the other ETM registers.
0xE0041000	ETM Control	This register controls the general operation of the ETM, for instance how tracing is enabled.
0xE0041010	ETM Status	This register provides information about the current status of the trace and trigger logic.
0xE0041008	ETM Trigger Event	This register defines the event that will control trigger.
0xE004101C	ETM Trace Enable Control	This register defines which comparator is selected.
0xE0041020	ETM Trace Enable Event	This register defines the trace enabling event.
0xE0041024	ETM Trace Start/Stop	This register defines the traces used by the trigger source to start and stop the trace, respectively.

### 33.15.4 Configuration example

To output a simple value to the TPIU:

- Configure the TPIU and enable the I/O\_TRACEN to assign TRACE I/Os in the STM32F446xx debug configuration register.
- Write 0xC5ACCE55 to the ETM Lock Access Register to unlock the write access to the ETM registers
- Write 0x00001D1E to the control register (configure the trace)
- Write 0000406F to the Trigger Event register (define the trigger event)
- Write 0000006F to the Trace Enable Event register (define an event to start/stop)
- Write 00000001 to the Trace Start/stop register (enable the trace)
- Write 0000191E to the ETM Control Register (end of configuration)

## 33.16 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

### 33.16.1 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.

### 33.16.2 Debug support for timers, watchdog, bxCAN and I<sup>2</sup>C

During a breakpoint, it is necessary to choose how the counter of timers and watchdog should behave:

- They can continue to count inside a breakpoint. This is usually required when a PWM is controlling a motor, for example.
- They can stop to count inside a breakpoint. This is required for watchdog purposes.

For the bxCAN, the user can choose to block the update of the receive register during a breakpoint.

For the I<sup>2</sup>C, the user can choose to block the SMBUS timeout during a breakpoint.

### 33.16.3 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.

## 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_IOEN	Res.	Res.	DBG_STANDBY	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\_IOEN**: Trace pin assignment control

- With TRACE\_IOEN=0:

TRACE\_MODE=xx: TRACE pins not assigned (default state)

- With TRACE\_IOEN=1:

- TRACE\_MODE=00: TRACE pin assignment for Asynchronous Mode
- TRACE\_MODE=01: TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 1
- TRACE\_MODE=10: TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 2
- TRACE\_MODE=11: TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 4

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).

### 33.16.4 Debug MCU APB1 freeze register (DBGMCU\_APB1\_FZ)

The DBGMCU\_APB1\_FZ register is used to configure the MCU under Debug. It concerns APB1 peripherals. It is mapped on the external PPB bus at address 0xE004 2008.

The register 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-bits access are supported.

Power-on reset (POR): 0x0000 0000 (not reset by system reset)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	DBG_CAN2_STOP	DBG_CAN1_STOP	DBG_I2CFMP_SMBUS_TIMEOUT	DBG_I2C3_SMBUS_TIMEOUT	DBG_I2C2_SMBUS_TIMEOUT	DBG_I2C1_SMBUS_TIMEOUT	Res.	Res.	Res.	Res.	Res.
					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.	DBG_IWDG_STOP	DBG_WWDG_STOP	DBG_RTC_STOP	Res.	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

Bits 31:27 Reserved, must be kept at reset value.

Bit 26 **DBG\_CAN2\_STOP:** Debug CAN2 stopped when Core is halted

- 0: Same behavior as in normal mode
- 1: The CAN2 receive registers are frozen

Bit 25 **DBG\_CAN1\_STOP:** Debug CAN2 stopped when Core is halted

- 0: Same behavior as in normal mode
- 1: The CAN2 receive registers are frozen

Bit 24 **DBG\_I2CFMP\_SMBUS\_TIMEOUT:** SMBUS timeout mode stopped when Core is halted

- 0: Same behavior as in normal mode
- 1: The SMBUS timeout is frozen

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 Reserved, must be kept at reset value.

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

### 33.16.5 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	DBG_TIM9_STOP												
													rw	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

## 33.17 TPIU (trace port interface unit)

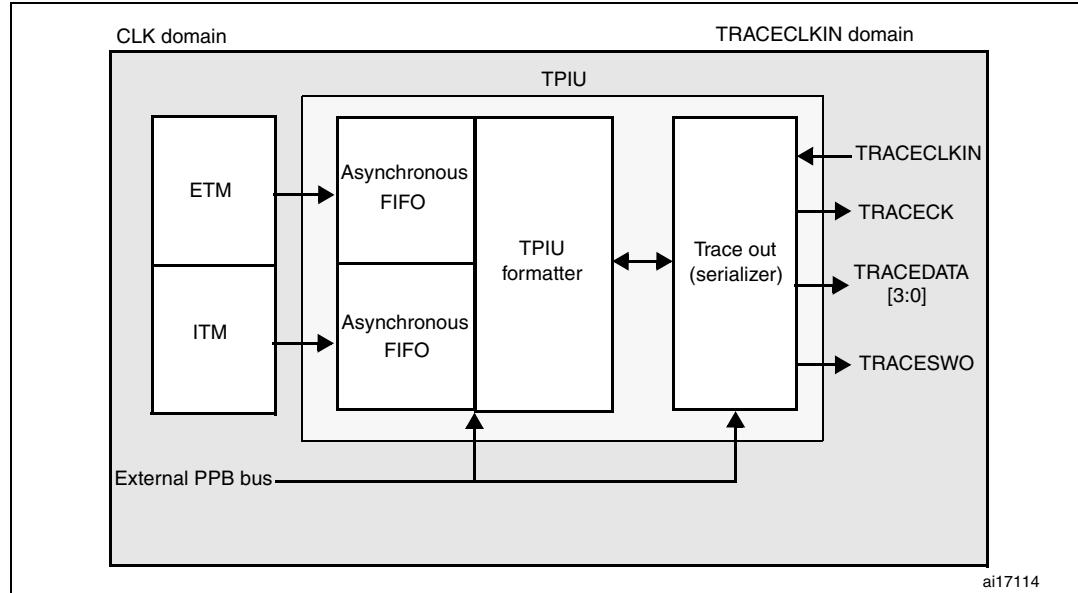
### 33.17.1 Introduction

The TPIU acts as a bridge between the on-chip trace data from the ITM and the ETM.

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 444. TPIU block diagram



### 33.17.2 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 242. Asynchronous TRACE pin assignment**

TPUI pin name	Trace synchronous mode	
	Type	Description
TRACESWO	O	TRACE Async Data Output <sup>(1)</sup>

1. Refer to the Alternate function mapping table in the datasheet

- 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 243. Synchronous TRACE pin assignment**

TPUI pin name	Trace synchronous mode	
	Type	Description
TRACECK	O	TRACE Clock <sup>(1)</sup>
TRACED[3:0]	O	TRACE Sync Data Outputs <sup>(1)</sup> Can be 1, 2 or 4.

1. Refer to the Alternate function mapping table in the datasheet

### TPUI TRACE pin assignment

By default, these pins are NOT assigned. They can be assigned by setting the TRACE\_IOEN and TRACE\_MODE bits in the [Debug MCU configuration register](#). This configuration has to be done by the debugger host.

In addition, the number of pins to assign depends on the trace configuration (asynchronous or synchronous).

- **Asynchronous mode:** 1 extra pin is needed
- **Synchronous mode:** from 2 to 5 extra pins are needed depending on the size of the data trace port register (1, 2 or 4):
  - TRACECK
  - TRACED(0) if port size is configured to 1, 2 or 4
  - TRACED(1) if port size is configured to 2 or 4
  - TRACED(2) if port size is configured to 4
  - TRACED(3) if port size is configured to 4

To assign the TRACE pin, the debugger host must program the bits TRACE\_IOEN and TRACE\_MODE[1:0] of the Debug MCU configuration Register (DBGMCU\_CR). By default the TRACE pins are not assigned.

This register is mapped on the external PPB and is reset by the PORESET (and not by the SYSTEM reset). It can be written by the debugger under SYSTEM reset.

**Table 244. Flexible TRACE pin assignment**

DBGMCU_CR register		Pins assigned for:	TRACE IO pin assigned						
TRACE_IOEN	TRACE_MODE [1:0]		PB3 /JTDO/ TRACESWO	PE2/ TRACECK	TRACED[0] (1)	TRACED[1] (1)	TRACED[2] (1)	TRACED[3] (1)	
0	XX	No Trace (default state)	Released (2)	-					
1	00	Asynchronous Trace	TRACESWO	-	-	Released (usable as GPIO)			
1	01	Synchronous Trace 1 bit	Released (2)	TRACECK	TRACED[0]	-	-	-	
1	10	Synchronous Trace 2 bit		TRACECK	TRACED[0]	TRACED[1]	-	-	
1	11	Synchronous Trace 4 bit		TRACECK	TRACED[0]	TRACED[1]	TRACED[2]	TRACED[3]	

1. Refer to the Alternate function mapping table in the datasheets.
2. When Serial Wire mode is used, it is released. But when JTAG is used, it is assigned to JTDO.

**Note:** By default, the TRACECLKIN input clock of the TPIU is tied to GND. It is assigned to HCLK two clock cycles after the bit TRACE\_IOEN has been set.

The debugger must then program the Trace Mode by writing the PROTOCOL[1:0] bits in the SPP\_R (Selected Pin Protocol) register of the TPIU.

- PROTOCOL=00: Trace Port Mode (synchronous)
- PROTOCOL=01 or 10: Serial Wire (Manchester or NRZ) Mode (asynchronous mode). Default state is 01

It then also configures the TRACE port size by writing the bits [3:0] in the CPSPS\_R (Current Sync Port Size Register) of the TPIU:

- 0x1 for 1 pin (default state)
- 0x2 for 2 pins
- 0x8 for 4 pins

### 33.17.3 TPUI formatter

The formatter protocol outputs data in 16-byte frames:

- seven bytes of data
- eight bytes of mixed-use bytes consisting of:
  - 1 bit (LSB) to indicate it is a DATA byte ('0) or an ID byte ('1).
  - 7 bits (MSB) which can be data or change of source ID trace.
- one byte of auxiliary bits where each bit corresponds to one of the eight mixed-use bytes:
  - if the corresponding byte was a data, this bit gives bit0 of the data.
  - if the corresponding byte was an ID change, this bit indicates when that ID change takes effect.

**Note:** Refer to the ARM® CoreSight Architecture Specification v1.0 (ARM® IHI 0029B) for further information

### 33.17.4 TPUI frame synchronization packets

The TPUI can generate two types of synchronization packets:

- The Frame Synchronization packet (or Full Word Synchronization packet)  
It consists of the word: 0x7F\_FF\_FF\_FF (LSB emitted first). This sequence can not occur at any other time provided that the ID source code 0x7F has not been used.  
It is output periodically **between** frames.  
In continuous mode, the TPA must discard all these frames once a synchronization frame has been found.
- The Half-Word Synchronization packet  
It consists of the half word: 0x7F\_FF (LSB emitted first).  
It is output periodically **between or within** frames.  
These packets are only generated in continuous mode and enable the TPA to detect that the TRACE port is in IDLE mode (no TRACE to be captured). When detected by the TPA, it must be discarded.

### 33.17.5 Transmission of the synchronization frame packet

There is no Synchronization Counter register implemented in the TPUI of the core. Consequently, the synchronization trigger can only be generated by the **DWT**. Refer to the registers DWT Control Register (bits SYNCTAP[11:10]) and the DWT Current PC Sampler Cycle Count Register.

The TPUI Frame synchronization packet (0x7F\_FF\_FF\_FF) is emitted:

- after each TPUI reset release. This reset is synchronously released with the rising edge of the TRACECLKIN clock. This means that this packet is transmitted when the

TRACE\_IOEN 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 TPUI (trace source ID added).

### 33.17.6 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.

### 33.17.7 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.

TRACESWO is multiplexed with JTDO when using the SW-DP pin. This way, this functionality is available in all STM32F446xx 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%.

### 33.17.8 TRACECLKIN connection in STM32F446xx

In the STM32F446xx, 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 STM32F446xx 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\_IOEN 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.*

### 33.17.9 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 245. Important TPIU registers**

Address	Register	Description
0xE0040004	Current port size	<p>Allows the trace port size to be selected:</p> <ul style="list-style-type: none"> <li>Bit 0: Port size = 1</li> <li>Bit 1: Port size = 2</li> <li>Bit 2: Port size = 3, not supported</li> <li>Bit 3: Port Size = 4</li> </ul> <p>Only 1 bit must be set. By default, the port size is one bit. (0x00000001)</p>
0xE00400F0	Selected pin protocol	<p>Allows the Trace Port Protocol to be selected:</p> <ul style="list-style-type: none"> <li>Bit1:0=</li> <li>00: Sync Trace Port Mode</li> <li>01: Serial Wire Output - manchester (default value)</li> <li>10: Serial Wire Output - NRZ</li> <li>11: reserved</li> </ul>
0xE0040304	Formatter and flush control	<p>Bits 31-9 = always '0'</p> <p>Bit 8 = TrigIn = always '1' to indicate that triggers are indicated</p> <p>Bits 7-4 = always 0</p> <p>Bits 3-2 = always 0</p> <p>Bit 1 = EnFCont. In Sync Trace mode (Select_Pin_Protocol register bit1:0=00), this bit is forced to '1': the formatter is automatically enabled in continuous mode. In asynchronous mode (Select_Pin_Protocol register bit1:0 &gt;&gt; 00), this bit can be written to activate or not the formatter.</p> <p>Bit 0 = always 0</p> <p>The resulting default value is 0x102</p> <p><b>Note:</b> In synchronous mode, because the TRACECTL pin is not mapped outside the chip, the formatter is always enabled in continuous mode -this way the formatter inserts some control packets to identify the source of the trace packets).</p>
0xE0040300	Formatter and flush status	Not used in Cortex®-M4 with FPU, always read as 0x00000008

### 33.17.10 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

## 33.18 DBG register map

**Table 246. DBG register map and reset values**

<sup>1</sup> The reset value is product dependent. For more information, refer to [Section 33.6.1: MCU device ID code](#).

## 34 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 STM32F446xx microcontrollers.

### 34.1 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: 0x1FFF 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
U_ID(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
U_ID(15:0)															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **U\_ID(31:0)**: 31:0 unique ID bits

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
U_ID(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
U_ID(47:32)															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **U\_ID(63:32)**: 63:32 unique ID bits

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
U_ID(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
U_ID(79:64)															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:0 **U\_ID(95:64)**: 95:64 Unique ID bits.

## 34.2 Flash size

Base address: 0x1FFF 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, 0x0200 corresponds to 512 Kbytes.

## 34.3 Package data register

Base address: 0x1FFF7BF0

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.	Res.	PKG[2:0]			Res.						
						r	r	r							

Bits 15:11 Reserved, must be kept at reset value.

Bits 10:8 **PKG[2:0]**: Package type

0x011: LQFP144, UFBGA144 (7x7) and UFBGA144 (10x10) packages

0x010: WLCSP81 package

0x001: LQFP100 package

0x000: LQFP64 package

Bits 7:0 Reserved, must be kept at reset value.

## 35 Revision history

Table 247. Document revision history

Date	Revision	Changes
17-Mar-2015	1	<p>Initial release.</p>
12-Jan-2016	2	<p>Updated <a href="#">Section 5.1.2: Battery backup domain</a>.      Updated <a href="#">Table 19: Standby mode entry and exit</a>.      Updated <a href="#">Section 6.3.2: RCC PLL configuration register (RCC_PLLCFGR)</a>, <a href="#">Section 6.3.23: RCC PLLI2S configuration register (RCC_PLLI2SCFGR)</a> and <a href="#">Section 6.3.24: RCC PLL configuration register (RCC_PLLSAICFGR)</a>.      Updated <a href="#">Section 11.3: AHB interface</a>, <a href="#">Section 11.4.3: SDRAM address mapping</a>, <a href="#">Section 11.5.4: NOR Flash/PSRAM controller asynchronous transactions</a>, <a href="#">Section 11.5.6: NOR/PSRAM controller registers</a>, <a href="#">SRAM/NOR-Flash write timing registers 1..4 (FMC_BWTR1..4)</a>, <a href="#">FIFO status and interrupt register (FMC_SR)</a>, <a href="#">Common memory space timing register 2..4 (FMC_PMEM)</a> and <a href="#">Attribute memory space timing registers (FMC_PATT)</a>, <a href="#">SDRAM initialization</a>.      Updated <a href="#">Table 74: Programmable NAND Flash access parameters</a>.      Updated figures <a href="#">31</a>, <a href="#">42</a>, <a href="#">43</a>, <a href="#">44</a> and <a href="#">45</a> in <a href="#">Section 11</a>.      Updated footnote <a href="#">5</a> of <a href="#">Figure 52</a>, and added footnote <a href="#">2</a> to <a href="#">Figure 51</a> and footnote <a href="#">1</a> to <a href="#">Figure 90</a>.      Updated <a href="#">Section 12.5.7: QUADSPI address register (QUADSPI_AR)</a>.      Updated <a href="#">Section 13.2: ADC main features</a> and <a href="#">Section 13.13.2: ADC control register 1 (ADC_CR1)</a>.      Updated figures <a href="#">109</a>, <a href="#">138</a>, <a href="#">152</a> and <a href="#">Input capture mode</a> in <a href="#">Section 16</a>.      Updated <a href="#">Figure 182</a>, <a href="#">Section 17.4.3: TIMx slave mode control register (TIMx_SMCR)</a> and <a href="#">Input capture mode</a> in <a href="#">Section 17</a>.      Updated <a href="#">Table 115: TIMx internal trigger connections</a> and <a href="#">Input capture mode</a> in <a href="#">Section 18</a>.      Updated <a href="#">Figure 237: Watchdog block diagram</a> and <a href="#">Section 21.4: How to program the watchdog timeout</a>.      Updated <a href="#">Section 22.6.4: RTC initialization and status register (RTC_ISR)</a>.      Updated <a href="#">Section 23.7.5: Timing register (FMP12C_TIMINGR)</a>.      Updated <a href="#">Section 24.6.2: I2C Control register 2 (I2C_CR2)</a>.      Added <a href="#">Section 25.3: USART implementation</a>.      Updated tables in <a href="#">Section 25.4.4: Fractional baud rate generation</a>.      Updated figures <a href="#">304</a>, <a href="#">305</a>, <a href="#">306</a> and <a href="#">307</a>, and their footnotes in <a href="#">Section 26.3: SPI functional description</a>.      Updated <a href="#">Section 29.1: SDIO main features</a>, <a href="#">Section 29.3: SDIO functional description</a>, <a href="#">Section 29.8.1: SDIO power control register (SDIO_POWER)</a>, <a href="#">Section 29.8.2: SDIO clock control register (SDIO_CLKCR)</a> and <a href="#">Section 29.8.4: SDIO command register (SDIO_CMD)</a>.      Updated <a href="#">Section 30.7.4: Identifier filtering</a>, <a href="#">CAN filter mode register (CAN_FM1R)</a>, <a href="#">CAN filter scale register (CAN_FS1R)</a>, <a href="#">CAN filter FIFO assignment register (CAN_FFA1R)</a>, <a href="#">CAN filter activation register (CAN_FA1R)</a> and <a href="#">Section 30.9.5: bxCAN register map</a>.      Updated <a href="#">Section 31.15.5: OTG reset register (OTG_GRSTCTL)</a>.      Updated <a href="#">Section 33.6.1: MCU device ID code</a> and <a href="#">Section 33.6.3: Cortex®-M4 with FPU TAP</a>.</p>

**Table 247. Document revision history (continued)**

Date	Revision	Changes
04-Jul-2017	3	<p>Updated <a href="#">Section 1.1: List of abbreviations for registers</a>.      Updated <a href="#">Section 5.4.2: PWR power control/status register (PWR_CSR)</a>.      Replaced former <a href="#">Section 9.3.1: General description</a> with <a href="#">Section 9.3.1: DMA block diagram</a> and <a href="#">Section 9.3.1: DMA block diagram</a>.      Updated <a href="#">Section 11.1: FMC main features, SRAM/NOR-Flash chip-select timing registers 1..4 (FMC_BTR1..4)</a>, <a href="#">Common memory space timing register 2..4 (FMC_PMEM)</a>, <a href="#">Attribute memory space timing registers (FMC_PATT)</a> and <a href="#">SDRAM Control registers 1,2 (FMC_SDCR1,2)</a>.      Updated <a href="#">Table 54: FMC_BCRx bit fields</a> and <a href="#">Table 72: FMC_BCRx bit fields</a>.      Updated <a href="#">Figure 38: Mode2 write access waveforms</a> and <a href="#">Figure 51: NAND Flash controller waveforms for common memory access</a>.      Added <a href="#">Section 12.3.2: QUADSPI pins</a>.      Updated <a href="#">Section 12.3.7: QUADSPI memory-mapped mode</a>,  <a href="#">Section 12.3.13: QUADSPI error management</a> and <a href="#">Section 12.5.1: QUADSPI control register (QUADSPI_CR)</a>.      Updated notes in <a href="#">Section 13.13.7: ADC watchdog higher threshold register (ADC_HTR)</a> and <a href="#">Section 13.13.8: ADC watchdog lower threshold register (ADC_LTR)</a>.      Removed former <a href="#">Section 15.3: DCMI pins</a> and added <a href="#">Section 15.4.1: DCMI block diagram</a>.      Updated <a href="#">Table 96: DCMI external signals</a>.      Changed D, PIXCLK, HSYNC and VSYNC with, respectively, DCMI_D, DCMI_HSYNC, DCMI_VSYNC and DCMI_VSYNC in <a href="#">Section 15: Digital camera interface (DCMI)</a>.      Updated <a href="#">FMPI2C master initialization</a>, <a href="#">Section 23.7.2: Control register 2 (FMPI2C_CR2)</a>, <a href="#">Section 23.7.3: Own address 1 register (FMPI2C_OAR1)</a>, <a href="#">Section 23.7.4: Own address 2 register (FMPI2C_OAR2)</a> and <a href="#">Section 23.7.5: Timing register (FMPI2C_TIMINGR)</a>.      Updated <a href="#">Figure 246: Slave initialization flowchart</a>.      Updated <a href="#">Section 25.6.1: Status register (USART_SR)</a>.      Updated <a href="#">Section 26.1: Introduction</a>, <a href="#">Section 26.3.7: SPI configuration</a> and notes in <a href="#">Resetting the SPI<sub>x</sub>_TXCRC and SPI<sub>x</sub>_RXCRC values</a> and in <a href="#">Section 26.7.1: SPI control register 1 (SPI_CR1) (not used in I2S mode)</a>.      Added <a href="#">Section 26.6.2: I2S full-duplex</a>.      Updated <a href="#">Section 27.2: SPDIFRX main features</a>, <a href="#">Section 27.3: SPDIFRX functional description</a>, <a href="#">Section 27.5.1: Control register (SPDIFRX_CR)</a> and <a href="#">Section 27.5.3: Status register (SPDIFRX_SR)</a>.      Added <a href="#">Section 27.3.10: Symbol clock generation</a>, <a href="#">Section 27.3.11: DMA Interface</a>, <a href="#">Section 27.5.10: SPDIFRX version register (SPDIFRX_VERR)</a>, <a href="#">Section 27.5.11: SPDIFRX identification register (SPDIFRX_IDR)</a> and <a href="#">Section 27.5.12: SPDIFRX size identification register (SPDIFRX_SIDR)</a>.      Updated <a href="#">Table 168: SPDIFRX interface register map and reset values</a>.   </p>

Table 247. Document revision history (continued)

Date	Revision	Changes
04-Jul-2017	3 cont'd	<p>Added <a href="#">Section 28.3.2: SAI pins and internal signals</a> and updated <a href="#">Section 28.3.8: SAI clock generator</a>, <a href="#">Section 28.3.9: Internal FIFOs</a>, <a href="#">Section 28.5.1: Global configuration register (SAI_GCR)</a>, <a href="#">Section 28.5.2: Configuration register 1 (SAI_ACR1 / SAI_BCR1)</a>.</p> <p>Updated <a href="#">Figure 353: Functional block diagram</a> and <a href="#">Figure 359: Audio block clock generator overview</a>.</p> <p>Updated <a href="#">Table 172: Example of possible audio frequency sampling range</a> and <a href="#">Table 176: SAI interrupt sources</a>.</p> <p>Updated <a href="#">Section 31.1: Introduction</a>, <a href="#">Section 31.2.3: Peripheral-mode features</a>, <a href="#">Section 31.9: Power options</a>, <a href="#">Section 31.11.3: FIFO RAM allocation</a>, <a href="#">Section 31.15.1: OTG control and status register (OTG_GOTGCTL)</a>, <a href="#">Section 31.15.3: OTG AHB configuration register (OTG_GAHBCFG)</a>, <a href="#">Section 31.15.4: OTG USB configuration register (OTG_GUSBCFG)</a>, <a href="#">Section 31.15.5: OTG reset register (OTG_GRSTCTL)</a>, <a href="#">Section 31.15.6: OTG core interrupt register (OTG_GINTSTS)</a>, <a href="#">Section 31.15.9: OTG Receive FIFO size register (OTG_GRXFSIZ)</a>, <a href="#">Section 31.15.11: OTG non-periodic transmit FIFO/queue status register (OTG_HNPTXSTS)</a>, <a href="#">Section 31.15.13: OTG general core configuration register (OTG_GCCFG)</a>, <a href="#">Section 31.15.17: OTG device IN endpoint transmit FIFO size register (OTG_DIEPTXF<sub>x</sub>)</a> (<math>x = 1..5[\text{FS}] / 8[\text{HS}]</math>, where <math>x</math> is the FIFO_number), <a href="#">Section 31.15.37: OTG device OUT endpoint common interrupt mask register (OTG_DOEPMSK)</a>, <a href="#">Section 31.15.47: OTG device endpoint-x control register (OTG_DIEPCTL<sub>x</sub>)</a> (<math>x = 1..5[\text{FS}] / 0..8[\text{HS}]</math>, where <math>x</math> = Endpoint_number), <a href="#">Section 31.15.49: OTG device endpoint-x control register (OTG_DOEPCTL<sub>x</sub>)</a> (<math>x = 1..5[\text{FS}] / 8[\text{HS}]</math>, where <math>x</math> = Endpoint_number), <a href="#">Section 31.15.50: OTG device endpoint-x interrupt register (OTG_DIEPINT<sub>x</sub>)</a> (<math>x = 0..5[\text{FS}] / 8[\text{HS}]</math>, where <math>x</math> = Endpoint_number), <a href="#">Section 31.15.51: OTG device endpoint-x interrupt register (OTG_DOEPINT<sub>x</sub>)</a> (<math>x = 0..5[\text{FS}] / 8[\text{HS}]</math>, where <math>x</math> = Endpoint_number), <a href="#">Section 31.15.56: OTG device IN endpoint-x transfer size register (OTG_DIEPTSI<sub>x</sub>)</a> (<math>x = 1..5[\text{FS}] / 8[\text{HS}]</math>, where <math>x</math> = Endpoint_number), <a href="#">Section 31.15.57: OTG device IN endpoint transmit FIFO status register (OTG_DTXFSTS<sub>x</sub>)</a> (<math>x = 0..5[\text{FS}] / 8[\text{HS}]</math>, where <math>x</math> = Endpoint_number), <a href="#">Section 31.15.58: OTG device OUT endpoint-x transfer size register (OTG_DOEPTSI<sub>x</sub>)</a> (<math>x = 1..5[\text{FS}] / 8[\text{HS}]</math>, where <math>x</math> = Endpoint_number), <a href="#">Section 31.16.3: Device initialization</a>, <a href="#">Section 31.16.4: DMA mode</a>, <a href="#">Section 31.16.5: Host programming model</a> and <a href="#">Section 31.16.6: Device programming model</a>.</p> <p>Added <a href="#">Section 31.15.15: OTG core LPM configuration register (OTG_GLPMCFG)</a> and <a href="#">Section 31.15.53: OTG Device channel-x DMA address register (OTG_DIEPDMAX)</a> (<math>x = 0..15</math>, where <math>x</math> = Channel_number).</p> <p>Added <a href="#">Table 214: OTG_HS speeds supported</a> and <a href="#">Table 215: OTG_FS speeds supported</a>.</p> <p>Updated <a href="#">Table 216: Core global control and status registers (CSRs)</a>, <a href="#">Table 218: Device-mode control and status registers</a> and <a href="#">Table 224: OTG_FS/OTG_HS register map and reset values</a>.</p> <p>Updated <a href="#">Section 32.1: Introduction</a>.</p> <p>Added <a href="#">Section 32.3.2: HDMI-CEC block diagram</a>.</p>

# Index

## A

ADC_CCR .....	394
ADC_CDR .....	397
ADC_CR1 .....	383
ADC_CR2 .....	385
ADC_CSR .....	393
ADC_DR .....	393
ADC_HTR .....	388
ADC_JDRx .....	392
ADC_JOFRx .....	388
ADC_JSQR .....	392
ADC_LTR .....	389
ADC_SMPR1 .....	387
ADC_SMPR2 .....	387
ADC_SQR1 .....	389
ADC_SQR2 .....	390
ADC_SQR3 .....	391
ADC_SR .....	382

## C

CAN_BTR .....	1066
CAN_ESR .....	1065
CAN_FA1R .....	1076
CAN_FFA1R .....	1075
CAN_FiRx .....	1077
CAN_FM1R .....	1075
CAN_FMR .....	1074
CAN_FS1R .....	1075
CAN_IER .....	1064
CAN_MCR .....	1057
CAN_MSR .....	1059
CAN_RDHxR .....	1073
CAN_RDLxR .....	1073
CAN_RDTxR .....	1072
CAN_RF0R .....	1062
CAN_RF1R .....	1063
CAN_RIxR .....	1071
CAN_TDHzR .....	1070
CAN_TDLxR .....	1070
CAN_TDTxR .....	1069
CAN_TIxR .....	1068
CAN_TSR .....	1060
CEC_CFGR .....	1277
CEC_CR .....	1276
CEC_IER .....	1282
CEC_ISR .....	1280
CEC_RXDR .....	1280

CEC_TXDR .....	1280
CKGATENR .....	168
CRC_DR .....	88
CRC_IDR .....	89

## D

DAC_CR .....	412
DAC_DHR12L1 .....	416
DAC_DHR12L2 .....	417
DAC_DHR12LD .....	418
DAC_DHR12R1 .....	415
DAC_DHR12R2 .....	417
DAC_DHR12RD .....	418
DAC_DHR8R1 .....	416
DAC_DHR8R2 .....	417
DAC_DHR8RD .....	419
DAC_DOR1 .....	419
DAC_DOR2 .....	419
DAC_SR .....	420
DAC_SWTRIGR .....	415
DBGMCU_APB1 .....	1308
DBGMCU_APB2_FZ .....	1310
DBGMCU_CR .....	1306
DBGMCU_IDCODE .....	1291
DCMI_CR .....	434
DCMI_CWSIZE .....	445
DCMI_CWSTRT .....	445
DCMI_DR .....	446
DCMI_ESCR .....	443
DCMI_ESUR .....	444
DCMI_ICR .....	442
DCMI_IER .....	440
DCMI_MIS .....	441
DCMI_RIS .....	439
DCMI_SR .....	438
DMA_HIFCR .....	225
DMA_HISR .....	224
DMA_LIFCR .....	225
DMA_LISR .....	223
DMA_SxCR .....	226
DMA_SxFCR .....	231
DMA_SxM0AR .....	230
DMA_SxM1AR .....	230
DMA_SxNDTR .....	229
DMA_SxPAR .....	230

## Index

---

### E

EXTI_EMR .....	245
EXTI_FTSR .....	246
EXTI_IMR .....	245
EXTI_PR .....	247
EXTI_RTSR .....	246
EXTI_SWIER .....	247

### F

FLASH_ACR .....	79
FLASH_FCR .....	82
FLASH_FKEYR .....	80
FLASH_FOPTCR .....	84
FLASH_FOPTKEYR .....	80
FLASH_FSR .....	81
FMC_BCR1..4 .....	286
FMC_BTR1..4 .....	288
FMC_BWTR1..4 .....	291
FMC_ECCR .....	304
FMC_PATT .....	302
FMC_PCR .....	299
FMC_PMEM .....	301
FMC_SDCMR .....	318
FMC_SDCR1,2 .....	314
FMC_SDRTR .....	319
FMC_SDSR .....	320
FMC_SDTR1,2 .....	316
FMC_SR .....	300
FMPI2C_CR1 .....	742
FMPI2C_CR2 .....	745
FMPI2C_ICR .....	754
FMPI2C_ISR .....	752
FMPI2C_OAR1 .....	748
FMPI2C_OAR2 .....	749
FMPI2C_PPCR .....	755
FMPI2C_RXDR .....	756
FMPI2C_TIMEOUTR .....	751
FMPI2C_TIMINGR .....	750
FMPI2C_TXDR .....	756

### G

GPIOx_AFRH .....	191
GPIOx_AFRL .....	191
GPIOx_BSRR .....	189
GPIOx_IDR .....	188
GPIOx_LCKR .....	189
GPIOx_MODER .....	186
GPIOx_ODR .....	189
GPIOx_OSPEEDR .....	187
GPIOx_OTYPER .....	187

GPIOx_PUPDR .....	188
-------------------	-----

### I

I2C_CCR .....	790
I2C_CR1 .....	780
I2C_CR2 .....	782
I2C_DR .....	785
I2C_OAR1 .....	748, 784
I2C_OAR2 .....	749, 784
I2C_SR1 .....	785
I2C_SR2 .....	789
I2C_TIMEOUTR .....	751
I2C_TIMINGR .....	750
I2C_TRISE .....	791
I2Cx_CR2 .....	745
IWDG_KR .....	642
IWDG_PR .....	643
IWDG_RLR .....	644
IWDG_SR .....	644

### O

OTG_CID .....	1141
OTG_DAINT .....	1168
OTG_DAINTMSK .....	1169
OTG_DCFG .....	1160
OTG_DCTL .....	1162
OTG_DEACHINT .....	1172
OTG_DEACHINTMSK .....	1173
OTG_DIEPCTL0 .....	1173
OTG_DIEPCTLx .....	1175
OTG_DIEPEMPMSK .....	1172
OTG_DIEPINTx .....	1181
OTG_DIEPMISK .....	1166
OTG_DIEPTSIZ0 .....	1184
OTG_DIEPTSIZx .....	1184-1186
OTG_DIEPTXF0 .....	1137
OTG_DIEPTXFx .....	1145
OTG_DOEPCTL0 .....	1177
OTG_DOEPCTLx .....	1179
OTG_DOEPINTx .....	1183
OTG_DOEPMISK .....	1167
OTG_DOEPTSIZ0 .....	1185
OTG_DOEPTSIZx .....	1188
OTG_DSTS .....	1165
OTG_DTHRCTL .....	1171
OTG_DTXFSTSx .....	1187
OTG_DVBUSSID .....	1170
OTG_DVBUSPULSE .....	1170
OTG_GAHBCFG .....	1118
OTG_GCCFG .....	1140
OTG_GI2CCTL .....	1139

OTG_GINTMSK .....	1131	RCC_AHB1RSTR .....	135
OTG_GINTSTS .....	1126	RCC_AHB2ENR .....	144
OTG_GLPMCFG .....	1141	RCC_AHB2LPENR .....	152
OTG GOTGCTL .....	1114	RCC_AHB2RSTR .....	137
OTG GOTGINT .....	1117	RCC_AHB3ENR .....	145
OTG_GRSTCTL .....	1124	RCC_AHB3LPENR .....	153
OTG_GRXFSIZ .....	1136	RCC_AHB3RSTR .....	137
OTG_GRXSTSP .....	1135	RCC_APB1ENR .....	145
OTG_GRXSTSR .....	1135	RCC_APB1LPENR .....	153
OTG_GUSBCFG .....	1120	RCC_APB1RSTR .....	138
OTG_HAINT .....	1149	RCC_APB2ENR .....	148
OTG_HAINTMSK .....	1150	RCC_APB2LPENR .....	157
OTG_HCCHARx .....	1153	RCC_APB2RSTR .....	141
OTG_HCDMAx .....	1160	RCC_BDCR .....	158
OTG_HCFG .....	1146	RCC_CFGR .....	130
OTG_HCINTMSKx .....	1157	RCC_CIR .....	132
OTG_HCINTx .....	1155	RCC_CR .....	126
OTG_HCSPLTx .....	1154	RCC_CSR .....	160
OTG_HCTSIZx .....	1158	RCC_DCKCFGR .....	166
OTG_HFIR .....	1147	RCC_PLLCFGR .....	128, 162, 165
OTG_HFNUM .....	1148	RCC_SSCGR .....	161
OTG_HNPTXFSIZ .....	1137	RTC_ALRMAR .....	679
OTG_HNPTXSTS .....	1138	RTC_ALRMBR .....	680
OTG_HPRT .....	1150	RTC_ALRMBSSR .....	688
OTG_HPTXFSIZ .....	1145	RTC_BKxR .....	689
OTG_HPTXSTS .....	1148	RTC_CALIBR .....	677
OTG_PCGCCTL .....	1189	RTC_CALR .....	684
<b>P</b>		RTC_CR .....	672
PWR_CR .....	110	RTC_DR .....	671
PWR_CSR .....	112	RTC_ISR .....	674
<b>Q</b>		RTC_PRER .....	676
QUADSPI_PIR .....	351	RTC_SHIFTR .....	682
QUADSPI_PSMAR .....	350	RTC_SSR .....	681
QUADSPI_PSMKR .....	350	RTC_TR .....	670
QUADSPI_ABR .....	349	RTC_TSDR .....	683
QUADSPI_AR .....	348	RTC_TSSSR .....	684
QUADSPI_CCR .....	346	RTC_TSTR .....	683
QUADSPI_CR .....	340	RTC_WPR .....	681
QUADSPI_DCR .....	343	RTC_WUTR .....	677
QUADSPI_DLR .....	345		
QUADSPI_DR .....	349		
QUADSPI_FCR .....	345		
QUADSPI_LPTR .....	351		
QUADSPI_SR .....	344		
<b>R</b>			
RCC_AHB1ENR .....	143	<b>S</b>	
RCC_AHB1LPENR .....	150	SAI_ACLRFR .....	975
		SAI_ACR1 .....	964
		SAI_ACR2 .....	967
		SAI_ADR .....	976
		SAI_AFRCR .....	969
		SAI_AIM .....	972
		SAI_ASLOTR .....	971
		SAI_ASRR .....	973
		SAI_BCLRFR .....	975
		SAI_BCR1 .....	964

## Index

---

SAI_BCR2 .....	967
SAI_BDR .....	976
SAI_BFRCR .....	969
SAI_BIM .....	972
SAI_BSLOTR .....	971
SAI_BSR .....	973
SAI_GCR .....	964
SDIO_ARG .....	1023
SDIO_CLKCR .....	1021
SDIO_DCOUNT .....	1028
SDIO_DCTRL .....	1026
SDIO_DLEN .....	1026
SDIO_DTIMER .....	1025
SDIO_FIFO .....	1034
SDIO_FIFOCNT .....	1033
SDIO_ICR .....	1029
SDIO_MASK .....	1031
SDIO_POWER .....	1021
SDIO_RESPCMD .....	1024
SDIO_RESPx .....	1024
SDIO_STA .....	1028
SPI_CR1 .....	886
SPI_CR2 .....	888
SPI_CRCPR .....	891
SPI_DR .....	891
SPI_I2SCFGR .....	893
SPI_I2SPR .....	894
SPI_RXCRCR .....	892
SPI_SR .....	889
SPI_TXCRCR .....	892
SYSCFG_EXTICR1 .....	196
SYSCFG_EXTICR2 .....	197
SYSCFG_EXTICR3 .....	198
SYSCFG_EXTICR4 .....	198
SYSCFG_MEMRMP .....	194

## T

TIM2_OR .....	577
TIM5_OR .....	578
TIMx_ARR .....	573, 613, 624, 638
TIMx_BDTR .....	514
TIMx_CCER .....	507, 571, 612, 623
TIMx_CCMR1 .....	503, 567, 608, 620
TIMx_CCMR2 .....	506, 570
TIMx_CCR1 .....	512, 574, 614, 625
TIMx_CCR2 .....	513, 574, 614
TIMx_CCR3 .....	513, 575
TIMx_CCR4 .....	514, 575
TIMx_CNT .....	511, 573, 613, 624, 637
TIMx_CR1 .....	493, 558, 602, 617, 634
TIMx_CR2 .....	494, 560, 636

TIMx_DCR .....	516, 576
TIMx_DIER .....	498, 563, 605, 618, 636
TIMx_DMAR .....	517, 576
TIMx_EGR .....	501, 566, 608, 619, 637
TIMx_PSC .....	511, 573, 613, 624, 638
TIMx_RCR .....	512
TIMx_SMCR .....	496, 561, 604
TIMx_SR .....	500, 564, 606, 618, 637

## U

USART_BRR .....	838
USART_CR1 .....	839, 920, 923
USART_CR2 .....	841
USART_CR3 .....	842
USART_DR .....	838, 927-931
USART_GTPR .....	844
USART_SR .....	835, 924, 926

## W

WWDG_CFR .....	651
WWDG_CR .....	650
WWDG_SR .....	651

**IMPORTANT NOTICE – PLEASE READ CAREFULLY**

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2017 STMicroelectronics – All rights reserved

